


Excel 2002 Formulas

John Walkenbach

Author of *Excel 2002 Power Programming with VBA*

“Even if you already have a shelf full of Excel reference books, be sure to squeeze in a couple of inches for this one.”

—Microsoft OfficePro magazine on the previous edition

 Power Utility Pak trial
and more on CD-ROM

Visit us at mandtbooks.com

P
R
O
F
E
S
S
I
O
N
A
L

M
I
N
D
W
A
R
E[™]

 M
A
N
D
T
B
O
O
K
S[™]



Excel 2002 Formulas

Excel 2002 Formulas

John Walkenbach



M&T Books

An imprint of Hungry Minds, Inc.

Best-Selling Books • Digital Downloads • e-Books • Answer Networks •
e-Newsletters • Branded Web Sites • e-Learning

New York, NY • Cleveland, OH • Indianapolis, IN

Excel 2002 Formulas

Published by
M&T Books
An imprint of Hungry Minds, Inc.
909 Third Avenue
New York, NY 10022
www.hungryminds.com

Copyright © 2001 Hungry Minds, Inc. All rights reserved.
No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

Library of Congress Control Number: 2001089348

ISBN: 0-7645-4800-X

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/RT/QZ/QR/IN

Distributed in the United States by Hungry Minds, Inc.
Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria, and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru;

by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For general information on Hungry Minds' products and services please contact our Customer Care department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

For sales inquiries and reseller information, including discounts, premium and bulk quantity sales, and foreign-language translations, please contact our Customer Care department at 800-434-3422, fax 317-572-4002 or write to Hungry Minds, Inc., Attn: Customer Care Department, 10475 Crosspoint Boulevard, Indianapolis, IN 46256.

For information on licensing foreign or domestic rights, please contact our Sub-Rights Customer Care department at 212-884-5000.

For information on using Hungry Minds' products and services in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 317-572-3168 or fax 317-572-4168.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK. THE PUBLISHER AND AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THERE ARE NO WARRANTIES WHICH EXTEND BEYOND THE DESCRIPTIONS CONTAINED IN THIS PARAGRAPH. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ACCURACY AND COMPLETENESS OF THE INFORMATION PROVIDED HEREIN AND THE OPINIONS STATED HEREIN ARE NOT GUARANTEED OR WARRANTED TO PRODUCE ANY PARTICULAR RESULTS, AND THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY INDIVIDUAL. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES. FULFILLMENT OF EACH COUPON OFFER IS THE SOLE RESPONSIBILITY OF THE OFFEROR.

Trademarks: Professional Mindware is a trademark or registered trademark of Hungry Minds, Inc. All other trademarks are property of their respective owners. Hungry Minds, Inc. is not associated with any product or vendor mentioned in this book.

 is a trademark of
Hungry Minds[™] Hungry Minds, Inc.

 is a trademark of
Hungry Minds, Inc.

Credits

ACQUISITIONS EDITOR

Greg Croy

PROJECT EDITOR

Susan Christophersen

TECHNICAL EDITOR

Bill Manville

COPY EDITORS

Jennifer Mario, Rebekah Mancilla

SENIOR PERMISSIONS EDITOR

Carmen Krikorian

EDITORIAL MANAGER

Kyle Looper

PROJECT COORDINATOR

Nancee Reeves

GRAPHICS AND PRODUCTION

SPECIALISTS

Sean Decker

Jill Piscitelli

Kendra Span

Laurie Stevens

Brian Torwelle

Jeremy Unger

Erin Zeltner

QUALITY CONTROL TECHNICIANS

Laura Albert

Andy Hollandbeck

Carl Pierce

MEDIA DEVELOPMENT SPECIALIST

Greg Stephens

MEDIA DEVELOPMENT COORDINATOR

Marisa Pearman

PROOFREADING AND INDEXING

TECHBOOKS Production Services

About the Author

John Walkenbach is a leading authority on spreadsheet software and principal of JWalk and Associates Inc., a Southern California-based consulting firm that specializes in spreadsheet application development. John is the author of about 30 spreadsheet books, and has written more than 300 articles and reviews for a variety of publications, including *PC World*, *InfoWorld*, *PC Magazine*, *Windows*, and *PC/Computing*. He also maintains a popular Internet Web site (*The Spreadsheet Page*, www.j-walk.com/ss), and is the developer of the Power Utility Pak, an award-winning add-in for Microsoft Excel. John graduated from the University of Missouri and earned a Masters and PhD from the University of Montana.

Preface

Thanks for buying my book. If you're interested in developing killer formulas and taking Excel to a new level, this book is as good as it gets. I'm confident that you'll agree that your money was invested wisely.

Why I Wrote This Book

I approached this project with one goal in mind: To write the ultimate Excel book that would appeal to a broad base of users. That's a fairly ambitious goal. But based on the feedback I received from the first edition, I think I've accomplished it.

I've been using Excel for nearly a decade and I spend a lot of time participating in the Excel newsgroups on the Internet. As a result, I'm very familiar with the types of questions that come up time and time again. Much of the material in this book was inspired by questions on the Excel newsgroups. This book provides the answers to those questions – along with answers to questions that probably never occurred to you!

As you probably know, most bookstores offer dozens of Excel books. The vast majority of these books are general-purpose user guides that explain how to use the features available in Excel (often by simply rewording the text in the help files). A few others focus on advanced issues such as macro programming or scientific applications. None (that's right, none!) hones in on the one fundamental component of Excel that is critically important to every user: formulas. Fact is, formulas are what make a spreadsheet a spreadsheet. The more you know about formulas, the better your spreadsheets will be. It's that simple.

Excel is the spreadsheet market leader, by a long shot. This is the case not only because of Microsoft's enormous marketing clout but also because it is truly the best spreadsheet available. One area in which Excel's superiority is most apparent is formulas. Excel has some special tricks up its sleeve in the formulas department. As you'll see, Excel lets you do things with formulas that are impossible with other spreadsheets.

It's a safe bet that only about ten percent of Excel users really understand how to get the most out of worksheet formulas. In this book, I attempt to nudge you into that elite group. Are you up to it?

What You Should Know

This is *not* a book for beginning Excel users. If you have absolutely no experience with Excel, this may not be the best book for you – unless you're one of a rare breed who can learn a new software product almost instantaneously.

To get the most out of this book, you should have some background using Excel. Specifically, I assume that you know how to

- ◆ Create workbooks, insert sheets, save files, and other basic tasks
- ◆ Navigate through a workbook
- ◆ Use Excel’s menus, toolbars, and dialog boxes
- ◆ Use basic Windows features, such as file management and copy and paste techniques



If you’re an experienced spreadsheet user, but you are new to Excel, Chapter 1 presents a concise overview of what this product has to offer.

What You Should Have

To make the best use of this book, you need a copy of Microsoft Excel. When I wrote the current edition of the book, I was using Excel 2002 (which is part of Microsoft Office XP). With a few exceptions (noted in the text), the material in this book also applies to all earlier versions of Excel that are still in use.

To use the examples on the companion CD-ROM, you’ll need a CD-ROM drive. Duh! The examples on the CD-ROM are discussed further in the “About the Companion CD-ROM” section, later in this preface.



I use Excel for Windows exclusively, and do not own a Macintosh. Therefore, I can’t guarantee that all of the examples will work with Excel for Macintosh. Excel’s cross-platform compatibility is pretty good, but it’s definitely not perfect.

As far as hardware goes, the faster the better. And, of course, the more memory in your system, the happier you’ll be. And, I strongly recommend using a high-resolution video mode: at least 1024 x 768.

Conventions in This Book

Take a minute to skim this section and learn some of the typographic conventions used throughout this book.

Keyboard Conventions

You need to use the keyboard to enter formulas. In addition, you can work with menus and dialog boxes directly from the keyboard – a method you may find easier if your hands are already positioned over the keys.

FORMULA LISTINGS

Formulas usually appear on a separate line in `monospace` font. For example, I may list the following formula:

```
=VLOOKUP(StockNumber,PriceList,2,False)
```

Excel supports a special type of formula known as an *array formula*. When you enter an array formula, press `Ctrl+Shift+Enter` (not just `Enter`). Excel encloses an array formula in brackets in order to remind you that it's an array formula. When I list an array formula, I include the brackets to make it clear that it is, in fact, an array formula. For example:

```
{=SUM(LEN(A1:A10))}
```



Do not type the brackets for an array formula. Excel will put them in automatically.

VBA CODE LISTINGS

This book also contains examples of VBA code. Each listing appears in a `monospace` font; each line of code occupies a separate line. To make the code easier to read, I usually use one or more tabs to create indentations. Indentation is optional, but it does help to delineate statements that go together.

If a line of code doesn't fit on a single line in this book, I use the standard VBA line continuation sequence: a space followed by an underscore character. This indicates that the line of code extends to the next line. For example, the following two lines comprise a single VBA statement:

```
If Right(cell.Value, 1) = "!" Then cell.Value _  
    = Left(cell.Value, Len(cell.Value) - 1)
```

You can enter this code either exactly as shown on two lines, or on a single line without the trailing underscore character.

KEY NAMES

Names of keys on the keyboard appear in normal type, for example Alt, Home, PgDn, and Ctrl. When you should press two keys simultaneously, the keys are connected with a plus sign: “Press Ctrl+G to display the Go To dialog box.”

FUNCTIONS, PROCEDURES, AND NAMED RANGES

Excel’s worksheet functions appear in all uppercase, like so: “Use the SUM function to add the values in column A.”

Macro and procedure names appear in normal type: “Execute the InsertTotals procedure.” I often use mixed upper- and lowercase to make these names easier to read. Named ranges appear in italic: “Select the *InputArea* range.”

Unless you’re dealing with text inside of quotation marks, Excel is not sensitive to case. In other words, both of the following formulas produce the same result:

```
=SUM(A1:A50)  
=sum(a1:a50)
```

Excel, however, will convert the characters in the second formula to uppercase.

Mouse Conventions

The mouse terminology in this book is all standard fare: “pointing,” “clicking,” “right-clicking,” “dragging,” and so on. You know the drill.

What the Icons Mean

Throughout the book, icons appear in the left margin to call your attention to points that are particularly important.



This icon indicates a feature new to Excel 2002.



I use Note icons to tell you that something is important — perhaps a concept that may help you master the task at hand or something fundamental for understanding subsequent material.



Tip icons indicate a more efficient way of doing something, or a technique that may not be obvious. These will often impress your officemates.



These icons indicate that an example file is on the companion CD-ROM. (See the upcoming “About the Companion CD-ROM” section.)



I use Caution icons when the operation that I’m describing can cause problems if you’re not careful.



I use the Cross Reference icon to refer you to other chapters that have more to say on a particular topic.

How This Book Is Organized

There are hundreds of ways to organize this material, but I settled on a scheme that divides the book into five main parts. In addition, I’ve included a few appendixes that provide supplemental information that you may find helpful.

Part I: Basic Information

This part is introductory in nature, and consists of Chapters 1 through 3. Chapter 1 sets the stage with a quick and dirty overview of Excel. This chapter is designed for readers who are new to Excel, but who have used other spreadsheet products. In Chapter 2, I cover the basics of formulas. This chapter is absolutely essential reading in order to get the most out of this book. Chapter 3 deals with names. If you thought names were just for cells and ranges, you’ll see that you’re missing out on quite a bit.

Part II: Using Functions in Your Formulas

This part consists of Chapters 4 through 10. Chapter 4 covers the basics of using worksheet functions in your formulas. I get more specific in subsequent chapters. Chapter 5 deals with manipulating text, Chapter 6 covers dates and times, and Chapter 7 explores various counting techniques. In Chapter 8, I discuss various types of lookup formulas. Chapter 9 deals with databases and lists, and Chapter 10 covers a variety of miscellaneous calculations such as unit conversions and rounding.

Part III: Financial Formulas

Part III consists of three chapters (Chapters 11 through 13) that deal with creating financial formulas. You'll find lots of useful formulas that you can adapt to your needs.



Most of the material in Chapters 11 through 13 was contributed by Norman Harker. Norman is a Senior Lecturer in Real Estate at the University of Western Sydney (Australia).

Part IV: Array Formulas

This part consists of Chapters 14 and 15. The majority of Excel users know little or nothing about array formulas—a topic that happens to be dear to me. Therefore I devote an entire part to this little-used yet extremely powerful feature.

Part V: Miscellaneous Formula Techniques

This part consists of Chapters 16 through 21. They cover a variety of topics—some of which, on the surface, may appear to have nothing to do with formulas. Chapter 16 demonstrates that a circular reference can be a good thing. In Chapter 17, you'll see why formulas can be important when you work with charts, and Chapter 18 covers formulas as they relate to pivot tables. Chapter 19 contains some very interesting (and useful) formulas that you can use in conjunction with Excel's conditional formatting and data validation features. Chapter 20 covers a topic that I call “megaformulas.” A megaformula is a huge formula that takes the place of several intermediary formulas. And what do you do when your formulas don't work correctly? Consult Chapter 21 for some debugging techniques.

Part VI: Developing Custom Worksheet Functions

This part consists of Chapters 22 through 25. This is the part that explores Visual Basic for Applications (VBA), the key to creating custom worksheet functions. Chapter 22 introduces VBA and the VB Editor, and Chapter 23 provides some necessary background on custom worksheet functions. Chapter 24 covers programming concepts, and Chapter 25 provides a slew of worksheet function examples that you can use as-is or customize for your own needs.

Appendixes

What's a computer book without appendixes? This book has five appendixes. In the appendixes, you'll find secrets about importing 1-2-3 files, a quick reference guide

to Excel's worksheet functions, tips on using custom number formats, and a handy guide to Excel resources on the Internet. The final appendix describes all the files on the CD-ROM.

How to Use This Book

You can use this book any way you please. If you choose to read it cover to cover while lounging on a sunny beach in Maui, that's fine with me. More likely, you'll want to keep it within arm's reach while you toil away in your dimly lit cubicle.

Due to the nature of the subject matter, the chapter order is often immaterial. Most readers will probably skip around, picking up useful tidbits here and there. The material contains many examples, designed to help you identify a relevant formula quickly. If you're faced with a challenging task, you may want to check the index first to see whether the book specifically addresses your problem.

About the Companion CD-ROM

The inside back cover of this book contains a CD-ROM that consists of three basic elements:

- ◆ Example workbooks that demonstrate concepts presented in the text.
- ◆ A trial copy of my Power Utility Pak 2000 add-in.
- ◆ A demo copy of my Sound-Proof 2000 add-in. Sound-Proof is a handy auditing tool that uses Microsoft Agent technology to read the contents of cells. You may prefer this to Excel 2002's text-to-speech feature.

The example workbook files on the companion CD-ROM are not compressed, so you can access them directly from the CD (installation not required). Power Utility Pak and Sound-Proof, however, *do* require installation. Refer to Appendix E for details.



All CD-ROM files are read-only. Therefore, if you open a file from the CD-ROM and make any changes to it, you'll need to save it to your hard drive. Also, if you copy a file from the CD-ROM to your hard drive, the file retains its read-only attribute. To change this attribute after copying a file, right-click the file-name or icon and select Properties from the shortcut menu. In the Properties dialog box, click the General tab and remove the check mark from the Read-only check box.

About the Power Utility Pak Offer

Toward the back of the book, you'll find a coupon that you can redeem for a discounted copy of my award-winning Power Utility Pak – a collection of useful Excel utilities, plus many new worksheet functions. I developed this package using VBA exclusively.

You can also use this coupon to purchase the complete VBA source code for a nominal fee. Studying the code is an excellent way to pick up some useful programming techniques. You can take the product for a test drive by installing the shareware version from the companion CD-ROM.



Power Utility Pak requires Excel 97 for Windows or later.

You can always download the most current version of the Power Utility Pak from my Web site:

<http://www.j-walk.com/ss>

Reach Out

I'm always interested in getting feedback on my books. The best way to provide this feedback is via email. Send your comments and suggestions to:

author@j-walk.com

Unfortunately, I'm not able to reply to specific questions. Posting your question to one of the Excel newsgroups is, by far, the best way to get such assistance. See Appendix D for specifics.

Also, when you're out surfing the Web, don't overlook my Web site ("The Spreadsheet Page"):

<http://www.j-walk.com/ss/>

Now, without further ado, it's time to turn the page and expand your horizons.

Acknowledgments

Thanks to everyone who purchased the first edition of this book. I'm especially grateful to those who took the time to provide me with valuable feedback and suggestions. I've incorporated many of the reader suggestions into this new edition.

I am also grateful to Norman Harker, Senior Lecturer in Real Estate at the University of Western Sydney (Australia). After reading the first edition, Norman pointed out that the single chapter on financial formulas was the weakest part of the book. Consequently, the financial formulas portion of the book has been beefed up significantly, and Norman provided the bulk of the contents of Chapters 11–13.

I would also like to thank Bill Manville for his superb technical editing skills. This is my second project with Bill, and I'm convinced that he is among the best technical editors in the business. He corrected many of my mistakes, made lots of useful suggestions, and re-wrote dozens of my formulas to make them perform better.

Finally, I wish to thank the folks at Hungry Minds for publishing this book. It is certainly not your "typical" Excel book, and publishing it was a risky venture. The risk paid off, however, as evidenced by the fact that it was selected for a second edition. Special thanks to Susan Christophersen, my project editor. She made my job much easier.

Contents at a Glance

	Preface	vii
	Acknowledgments	xv
Part I	Basic Information	
<hr/>		
Chapter 1	Excel in a Nutshell	3
Chapter 2	Basic Facts about Formulas	29
Chapter 3	Working with Names	57
Part II	Using Functions in Your Formulas	
<hr/>		
Chapter 4	Introducing Worksheet Functions	97
Chapter 5	Manipulating Text	113
Chapter 6	Working with Dates and Times	139
Chapter 7	Counting and Summing Techniques	179
Chapter 8	Lookups	211
Chapter 9	Databases and Lists	237
Chapter 10	Miscellaneous Calculations	269
Part III	Financial Formulas	
<hr/>		
Chapter 11	Introducing Financial Formulas	293
Chapter 12	Discounting and Depreciation Financial Functions	329
Chapter 13	Advanced Uses of Financial Functions and Formulas	351
Part IV	Array Formulas	
<hr/>		
Chapter 14	Introducing Arrays	375
Chapter 15	Performing Magic with Array Formulas	397
Part V	Miscellaneous Formula Techniques	
<hr/>		
Chapter 16	Intentional Circular References	425
Chapter 17	Charting Techniques	441
Chapter 18	Pivot Tables	489
Chapter 19	Conditional Formatting and Data Validation	513
Chapter 20	Creating Megaformulas	541
Chapter 21	Tools and Methods for Debugging Formulas	559

Part VI	Developing Custom Worksheet Functions	
<hr/>		
Chapter 22	Introducing VBA	587
Chapter 23	Function Procedure Basics	599
Chapter 24	VBA Programming Concepts	619
Chapter 25	VBA Custom Function Examples	653
Appendixes		
<hr/>		
Appendix A	Working with Imported 1-2-3 Files	699
Appendix B	Excel Function Reference	717
Appendix C	Using Custom Number Formats	733
Appendix D	Additional Excel Resources	759
Appendix E	What's on the CD-ROM	765
	Index	777
	End-User License Agreement	825
	CD-ROM Installation Instructions	828

Contents

	Preface	vii
	Acknowledgments	xv
Part I	Basic Information	
<hr/>		
Chapter 1	Excel in a Nutshell	3
	The History of Excel	4
	It Started with VisiCalc	4
	Then Came Lotus	4
	Microsoft Enters the Picture	4
	Excel Versions	5
	The Object Model Concept	7
	The Workings of Workbooks	7
	Worksheets	9
	Chart Sheets	10
	XLM Macro Sheets	10
	Dialog Sheets	10
	Excel's User Interface	11
	Menus	11
	Shortcut Menus	11
	Smart Tags	12
	Dialog Boxes	12
	Toolbars	13
	Drag-and-Drop	13
	Keyboard Shortcuts	14
	Customized On-screen Display	14
	Data Entry	14
	Object and Cell Selecting	14
	Cell Formatting	16
	Numeric Formatting	16
	Stylistic Formatting	16
	Worksheet Formulas and Functions	17
	Objects on the Draw Layer	18
	Shapes	18
	Diagrams	18
	Linked Picture Objects	19
	Maps	19
	Dialog Box Controls	19
	Charts	20

	Customization in Excel	20
	Macros	20
	Toolbars	21
	Add-in Programs	21
	Analysis Tools	21
	Database Access	22
	Outlines	23
	Scenario Management	23
	Analysis ToolPak	24
	Pivot Tables	24
	Auditing Capabilities	24
	Solver Add-in	24
	Protection Options	25
	Protecting Formulas from Being Overwritten	25
	Protecting a Workbook's Structure	26
Chapter 2	Basic Facts about Formulas	29
	Entering and Editing Formulas	29
	Formula Elements	29
	Entering a Formula	30
	Pasting Names	32
	Spaces and Line Breaks	32
	Formula Limits	32
	Sample Formulas	33
	Editing Formulas	34
	Using Operators in Formulas	35
	Reference Operators	36
	Sample Formulas That Use Operators	36
	Operator Precedence	38
	Nested Parentheses	39
	Calculating Formulas	40
	Cell and Range References	42
	Creating an Absolute Reference	42
	Referencing Other Sheets or Workbooks	44
	Making an Exact Copy of a Formula	45
	Converting Formulas to Values	46
	Hiding Formulas	48
	Errors in Formulas	49
	Dealing with Circular References	51
	Goal Seeking	53
	A Goal-Seeking Example	53
	More about Goal Seeking	54
Chapter 3	Working with Names	57
	What's in a Name?	57
	Methods for Creating Cell and Range Names	58
	Creating Names Using the Define Name Dialog Box	58
	Creating Names Using the Name Box	59

Creating Names Automatically	61
Naming Entire Rows and Columns	63
Names Created by Excel	64
Creating Multisheet Names	65
A Name's Scope	66
Creating Worksheet-Level Names	67
Combining Worksheet- and Workbook-Level Names	67
Referencing Names from Another Workbook	68
Working with Range and Cell Names	68
Creating a List of Names	68
Using Names in Formulas	69
Using the Intersection Operators with Names	70
Using the Range Operator with Names	72
Referencing a Single Cell in a Multicell Named Range	73
Applying Names to Existing Formulas	73
Applying Names Automatically when Creating a Formula	74
Unapplying Names	74
Deleting Names	75
Deleting Named Cells or Ranges	75
Redefining Names	76
Changing Names	76
Viewing Named Ranges	76
Using Names in Charts	77
How Excel Maintains Cell and Range Names	77
Inserting a Row or Column	77
Deleting a Row or Column	77
Cutting and Pasting	78
Potential Problems with Names	78
Name Problems When Copying Sheets	78
Name Problems when Deleting Sheets	79
The Secret to Understanding Names	80
Naming Constants	81
Naming Text Constants	82
Using Worksheet Functions in Named Formulas	83
Using Cell and Range References in Named Formulas	84
Using Named Formulas with Relative References	85
Advanced Techniques That Use Names	88
Using the INDIRECT Function with a Named Range	88
Using the INDIRECT Function to Create a Named Range with a Fixed Address	89
Using Arrays in Named Formulas	90
Creating a Dynamic Named Formula	91

Part II Using Functions in Your Formulas

Chapter 4	Introducing Worksheet Functions	97
	What Is a Function?	97
	Simplify Formulas	98
	Perform Otherwise Impossible Calculations	98
	Speed Up Editing Tasks	98
	Provide Decision-Making Capability	99
	More about Functions	99
	Function Argument Types	99
	Names as Arguments	100
	Full-Column or Full-Row as Arguments	101
	Literal Values as Arguments	102
	Expressions as Arguments	102
	Other Functions as Arguments	102
	Arrays as Arguments	103
	Ways to Enter a Function into a Formula	103
	Entering a Function Manually	104
	Using the Insert Function Dialog Box to Enter a Function	104
	More Tips for Entering Functions	106
	Function Categories	109
	Financial Functions	109
	Date & Time Functions	109
	Math & Trig Functions	109
	Statistical Functions	109
	Lookup and Reference Functions	110
	Database Functions	110
	Text Functions	110
	Logical Functions	110
	Information Functions	110
	Engineering Functions	110
	User-Defined Functions	111
	Other Function Categories	111
	Analysis ToolPak Functions	112
Chapter 5	Manipulating Text	113
	A Few Words about Text	113
	How Many Characters in a Cell?	113
	Numbers as Text	114
	Text Functions	115
	Determining Whether a Cell Contains Text	115
	Working with Character Codes	117
	Determining Whether Two Strings Are Identical	119
	Joining Two or More Cells	120
	Displaying Formatted Values as Text	121

	Displaying Formatted Currency Values as Text	122
	Repeating a Character or String	123
	Creating a Text Histogram	123
	Padding a Number	124
	Removing Excess Spaces and Nonprinting Characters	125
	Counting Characters in a String	126
	Changing the Case of Text	126
	Extracting Characters from a String	127
	Replacing Text with Other Text	127
	Finding and Searching within a String	128
	Searching and Replacing within a String	129
	Advanced Text Formulas	130
	Counting Specific Characters in a Cell	130
	Counting the Occurrences of a Substring in a Cell	130
	Expressing a Number as an Ordinal	131
	Determining a Column Letter for a Column Number	132
	Extracting a Filename from a Path Specification	132
	Extracting the First Word of a String	133
	Extracting the Last Word of a String	133
	Extracting All but the First Word of a String	133
	Extracting First Names, Middle Names, and Last Names	134
	Removing Titles from Names	135
	Counting the Number of Words in a Cell	135
	Custom VBA Text Functions	136
	Working with Dates and Times	139
	How Excel Handles Dates and Times	139
	Understanding Date Serial Numbers	140
	Entering Dates	141
	Understanding Time Serial Numbers	142
	Entering Times	144
	Formatting Dates and Times	145
	Problems with Dates	147
	Date-Related Functions	149
	Displaying the Current Date	150
	Displaying Any Date	151
	Generating a Series of Dates	152
	Converting a Non-Date String to a Date	153
	Calculating the Number of Days between Two Dates	153
	Calculating the Number of Work Days between Two Dates	154
	Offsetting a Date Using Only Work Days	156
	Calculating the Number of Years between Two Dates	156
	Calculating a Person's Age	156
	Determining the Day of the Year	157
	Determining the Day of the Week	158
	Determining the Date of the Most Recent Sunday	160
Chapter 6		

	Determining the First Day of the Week after a Date	160
	Determining the nth Occurrence of a Day of the Week in a Month	160
	Counting the Occurrences of a Day of the Week	161
	Expressing a Date as an Ordinal Number	162
	Calculating Dates of Holidays	163
	Determining the Last Day of a Month	165
	Determining Whether a Year Is a Leap Year	165
	Determining a Date's Quarter	166
	Converting a Year to Roman Numerals	166
	Creating a Calendar in a Range	166
	Time-Related Functions	167
	Displaying the Current Time	168
	Displaying Any Time	169
	Summing Times That Exceed 24 Hours	170
	Calculating the Difference between Two Times	172
	Converting from Military Time	174
	Converting Decimal Hours, Minutes, or Seconds to a Time	174
	Adding Hours, Minutes, or Seconds to a Time	175
	Converting between Time Zones	175
	Rounding Time Values	176
	Working with Non-Time-of-Day Values	177
Chapter 7	Counting and Summing Techniques	179
	Counting and Summing Worksheet Cells	179
	Counting or Summing Records in Databases and Pivot Tables	181
	Basic Counting Formulas	182
	Counting the Total Number of Cells	183
	Counting Blank Cells	183
	Counting Nonblank Cells	184
	Counting Numeric Cells	185
	Counting Nontext Cells	185
	Counting Text Cells	185
	Counting Logical Values	185
	Error Values in a Range	185
	Advanced Counting Formulas	186
	Counting Cells Using the COUNTIF Function	186
	Counting Cells Using Multiple Criteria	188
	Counting the Most Frequently Occurring Entry	191
	Counting the Occurrences of Specific Text	192
	Counting the Number of Unique Values	193
	Creating a Frequency Distribution	195
	Summing Formulas	201
	Summing All Cells in a Range	201
	Computing a Cumulative Sum	202
	Summing the "Top n" Values	204

	Conditional Sums Using a Single Criterion	204
	Summing Only Negative Values	206
	Summing Values Based on a Different Range	206
	Summing Values Based on a Text Comparison	207
	Summing Values Based on a Date Comparison	207
	Conditional Sums Using Multiple Criteria	208
	Using And Criteria	208
	Using Or Criteria	209
	Using And and Or Criteria	209
	Using VBA Functions to Count and Sum	210
Chapter 8	Lookups	211
	What Is a Lookup Formula?	211
	Functions Relevant to Lookups	212
	Basic Lookup Formulas	213
	The VLOOKUP Function	213
	The HLOOKUP Function	215
	The LOOKUP Function	216
	Combining the MATCH and INDEX Functions	217
	Specialized Lookup Formulas	219
	Looking Up an Exact Value	220
	Looking Up a Value to the Left	221
	Performing a Case-Sensitive Lookup	222
	Choosing among Multiple Lookup Tables	223
	Determining Letter Grades for Test Scores	224
	Calculating a Grade Point Average	225
	Performing a Two-Way Lookup	226
	Performing a Two-Column Lookup	228
	Determining the Address of a Value within a Range	229
	Looking Up a Value Using the Closest Match	230
	Looking Up a Value Using Linear Interpolation	231
Chapter 9	Databases and Lists	237
	Worksheet Lists or Databases	237
	Using AutoFiltering	240
	AutoFiltering Basics	240
	Counting and Summing Filtered Data	242
	Copying and Deleting Filtered Data	243
	Using Advanced Filtering	244
	Setting Up a Criteria Range	246
	Filtering a List	247
	Specifying Advanced Filter Criteria	248
	Specifying a Single Criterion	249
	Specifying Multiple Criteria	253
	Specifying Computed Criteria	255
	Using Database Functions with Lists	258
	Summarizing a List with a Data Table	261
	Creating Subtotals	264

Chapter 10	Miscellaneous Calculations	269
	Unit Conversions	269
	Using the Unit Conversion Tables	269
	Converting Metric Units	270
	Distance Conversions	272
	Weight Conversions	272
	Liquid Measurement Conversions	272
	Surface Conversions	272
	Volume Conversions	272
	Force Conversions	272
	Energy Conversions	272
	Time Conversions	272
	Temperature Conversions	277
	Solving Right Triangles	277
	Area, Surface, Circumference, and Volume Calculations	280
	Calculating the Area and Perimeter of a Square	280
	Calculating the Area and Perimeter of a Rectangle	281
	Calculating the Area and Perimeter of a Circle	281
	Calculating the Area of a Trapezoid	281
	Calculating the Area of a Triangle	281
	Calculating the Surface and Volume of a Sphere	282
	Calculating the Surface and Volume of a Cube	282
	Calculating the Surface and Volume of a Cone	282
	Calculating the Volume of a Cylinder	282
	Calculating the Volume of a Pyramid	283
	Solving Simultaneous Equations	283
	Rounding Numbers	285
	Basic Rounding Formulas	286
	Rounding to the Nearest Multiple	287
	Rounding Dollar Values	287
	Working with Fractional Dollars	288
	Using the INT and TRUNC Functions	288
	Rounding to an Even or Odd Integer	289
	Rounding to n Significant Digits	290

Part III Financial Formulas

Chapter 11	Introducing Financial Formulas	293
	Excel's Basic Financial Functions	293
	Signing of Money Flows Convention	295
	Accumulation, Discounting, and Amortization Functions	296
	Simple Accumulation Problems	297
	Complex Accumulation Problems	301
	Simple Discounting Problems	303

	Complex Discounting Problems	307
	Amortization Problems	308
	Converting Interest Rates	313
	Methods of Quoting Interest Rates	313
	Converting Interest Rates Using the Financial Functions Add-in	314
	Effective Cost of Loans	317
	Impact of Fees and Charges upon Effective Interest	317
	“Flat” Rate Loans	319
	Interest-Free Loans	319
	“Annual Payments / 12” Loan Costs	320
	Calculating the Interest and Principal Components	320
	Using the IPMT and PPMT Functions	321
	Using the CUMIPMT and CUMPRINC Functions	323
	Matching Different Interest and Payment Frequencies	324
	Limitations of Excel’s Financial Functions	325
	Deferred Start to a Series of Regular Payments	326
	Valuing a Series of Regular Payments	326
Chapter 12	Discounting and Depreciation	
	Financial Functions	329
	Using the NPV Function	329
	Definition of NPV	330
	NPV Function Examples	330
	Using the NPV Function to Calculate Accumulated Amounts	337
	Using the IRR Function	339
	Multiple Rates of IRR and the MIRR Function	342
	Using the FVSCHEDULE Function	346
	Depreciation Calculations	347
Chapter 13	Advanced Uses of Financial Functions and Formulas	351
	Creating Dynamic Financial Schedules	351
	Creating Amortization Schedules	352
	A Simple Amortization Schedule	352
	A Detailed Amortization Schedule	355
	A Variable Loan Rate Amortization Schedule	356
	Summarizing Loan Options Using a Data Table	358
	Creating a One-Way Data Table	358
	Creating a Two-Way Data Table	360
	Accumulation Schedules	361
	Discounted Cash Flow Schedules	363
	Credit Card Calculations	365
	XIRR and XNPV Functions	366
	Variable Rate Analysis	369
	Creating Indices	370

Part IV Array Formulas

Chapter 14	Introducing Arrays	375
	Introducing Array Formulas	375
	A Multicell Array Formula	376
	A Single-Cell Array Formula	377
	Creation of an Array Constant	378
	Array Constant Elements	379
	Understanding the Dimensions of an Array	379
	One-Dimensional Horizontal Arrays	379
	One-Dimensional Vertical Arrays	380
	Two-Dimensional Arrays	380
	Naming Array Constants	381
	Working with Array Formulas	383
	Entering an Array Formula	383
	Selecting an Array Formula Range	383
	Editing an Array Formula	384
	Expanding or Contracting a Multicell Array Formula	385
	Using Multicell Array Formulas	386
	Creating an Array from Values in a Range	386
	Creating an Array Constant from Values in a Range	386
	Performing Operations on an Array	387
	Using Functions with an Array	388
	Transposing an Array	388
	Generating an Array of Consecutive Integers	389
	Using Single-Cell Array Formulas	390
	Counting Characters in a Range	391
	Summing the Three Smallest Values in a Range	392
	Counting Text Cells in a Range	392
	Eliminating Intermediate Formulas	394
	Using an Array in Lieu of a Range Reference	395
Chapter 15	Performing Magic with Array Formulas	397
	Working with Single-Cell Array Formulas	397
	Summing a Range That Contains Errors	398
	Counting the Number of Error Values in a Range	398
	Summing Based on a Condition	399
	Summing the n Largest Values in a Range	402
	Computing an Average That Excludes Zeros	402
	Determining Whether a Particular Value Appears in a Range	403
	Counting the Number of Differences in Two Ranges	404
	Returning the Location of the Maximum Value in a Range	404
	Finding the Row of a Value's nth Occurrence in a Range	405
	Returning the Longest Text in a Range	405

Determining Whether a Range Contains Valid Values 406

Summing the Digits of an Integer 406

Summing Rounded Values 408

Summing Every nth Value in a Range 409

Removing Non-Numeric Characters from a String 410

Determining the Closest Value in a Range 410

Returning the Last Value in a Column 410

Returning the Last Value in a Row 412

Ranking Data with an Array Formula 412

Creating a Dynamic Crosstab Table 413

Working with Multicell Array Formulas 414

Returning Only Positive Values from a Range 414

Returning Nonblank Cells from a Range 415

Reversing the Order of the Cells in a Range 415

Sorting a Range of Values Dynamically 416

Returning a List of Unique Items in a Range 416

Displaying a Calendar in a Range 417

Returning an Array from a Custom VBA Function 418

Part V **Miscellaneous Formula Techniques**

Chapter 16 **Intentional Circular References 425**

What Are Circular References? 425

 Correcting an Accidental Circular Reference 426

 Understanding Indirect Circular References 427

Intentional Circular References 428

How Excel Determines Calculation
 and Iteration Settings 430

Circular Reference Examples 431

 Time Stamping a Cell Entry 432

 Calculating an All-Time-High Value 432

 Generating Unique Random Integers 433

 Solving a Recursive Equation 435

 Solving Simultaneous Equations Using
 a Circular Reference 436

Potential Problems with Intentional
 Circular References 438

Chapter 17 **Charting Techniques 441**

Representing Data in Charts 441

 Understanding the SERIES Formula 441

 Creating Links to Cells 445

 Charting Progress toward a Goal 448

 Creating a Gantt Chart 449

 Creating a Comparative Histogram 451

	Creating a Box Plot	453
	Plotting Every nth Data Point	455
	Updating a Data Series Automatically	457
	Plotting the Last n Data Points	458
	Plotting Data Interactively	460
	Plotting Based on the Active Row	460
	Selecting Data from a Combo Box	461
	Plotting Functions with One Variable	462
	Plotting Functions with Two Variables	466
	Creating Awesome Designs	468
	Working with Trendlines	469
	Linear Trendlines	470
	Nonlinear Trendlines	474
	Useful Chart Tricks	479
	Storing Multiple Charts on a Chart Sheet	479
	Viewing an Embedded Chart in a Window	480
	Changing a Worksheet Value by Dragging a Data Point	480
	Using Animated Charts	481
	Creating a “Gauge” Chart	482
	Creating a “Clock” Chart	483
	Drawing with an XY Chart	486
Chapter 18	Pivot Tables	489
	About Pivot Tables	489
	A Pivot Table Example	490
	Data Appropriate for a Pivot Table	493
	Creating a Pivot Table	495
	Step 1: Specifying the Data Location	495
	Step 2: Specifying the Data	496
	Step 3: Completing the Pivot Table	497
	Grouping Pivot Table Items	503
	Creating a Calculated Field or Calculated Item	506
	Creating a Calculated Field in a Pivot Table	507
	Inserting a Calculated Item into a Pivot Table	509
Chapter 19	Conditional Formatting and Data Validation	513
	Conditional Formatting	513
	Specifying Conditional Formatting	514
	Formatting Types You Can Apply	515
	Specifying Conditions	516
	Working with Conditional Formats	518
	Conditional Formatting Formulas	521
	Using Custom Functions in Conditional Formatting Formulas	530

	Data Validation	533
	Specifying Validation Criteria	534
	Types of Validation Criteria You Can Apply	536
	Using Formulas for Data Validation Rules	537
	Using Data Validation Formulas to Accept Only Specific Entries	538
Chapter 20	Creating Megaformulas	541
	What Is a Megaformula?	541
	Creating a Megaformula: A Simple Example	542
	Megaformula Examples	544
	Using a Megaformula to Remove Middle Names	544
	Using a Megaformula to Return a String's Last Space Character Position	548
	Using a Megaformula to Determine the Validity of a Credit Card Number	552
	The Pros and Cons of Megaformulas	557
Chapter 21	Tools and Methods for Debugging Formulas	559
	Formula Debugging?	559
	Formula Problems and Solutions	560
	Mismatched Parentheses	561
	Cells Are Filled with #####	562
	Blank Cells Are Not Blank	563
	Formulas Returning an Error	563
	Absolute/Relative Reference Problems	567
	Operator Precedence Problems	568
	Formulas Are Not Calculated	569
	Actual versus Displayed Values	570
	Floating Point Number Errors	571
	"Phantom Link" Errors	572
	Circular Reference Errors	572
	Excel's Auditing Tools	572
	Identifying Cells of a Particular Type	573
	Viewing Formulas	573
	Tracing Cell Relationships	575
	Tracing Error Values	577
	Fixing Circular Reference Errors	577
	Using Excel 2002's Background Error Checking Feature	578
	Using Excel 2002's Formula Evaluator	580
	Third-Party Auditing Tools	580
	Power Utility Pak	581
	Spreadsheet Detective	582
	Excel Auditor	582

Part VI	Developing Custom Worksheet Functions	
Chapter 22	Introducing VBA	587
	About VBA	587
	Introducing the Visual Basic Editor	588
	Activating the VB Editor	588
	The VB Editor Components	589
	Using the Project Window	590
	Using Code Windows	593
	Entering VBA Code	595
	Saving Your Project	598
Chapter 23	Function Procedure Basics	599
	Why Create Custom Functions?	599
	An Introductory VBA Function Example	600
	About Function Procedures	602
	Declaring a Function	602
	Choosing a Name for Your Function	603
	Using Functions in Formulas	604
	Using Function Arguments	605
	Using the Insert Function Dialog Box	605
	Adding a Function Description	606
	Specifying a Function Category	607
	Testing and Debugging Your Functions	609
	Using VBA's MsgBox Statement	610
	Using Debug.Print Statements in Your Code	612
	Calling the Function from a Sub Procedure	613
	Setting a Breakpoint in the Function	616
	Creating Add-Ins	616
Chapter 24	VBA Programming Concepts	619
	An Introductory Example Function Procedure	619
	Using Comments in Your Code	622
	Using Variables, Data Types, and Constants	622
	Defining Data Types	623
	Declaring Variables	624
	Using Constants	626
	Using Strings	627
	Using Dates	627
	Using Assignment Expressions	628
	Using Arrays	629
	Declaring an Array	630
	Declaring Multidimensional Arrays	630
	Using VBA's Built-in Functions	631
	Controlling Execution	633
	The If-Then Construct	633
	The Select Case Construct	635

	Looping Blocks of Instructions	636
	The On Error Statement	641
	Using Ranges	643
	The For Each-Next Construct	643
	Referencing a Range	644
	Some Useful Properties of Ranges	646
	The Set Keyword	649
	The Intersect Function	650
	The Union Function	651
	The UsedRange Property	651
Chapter 25	VBA Custom Function Examples	653
	Simple Functions	653
	Does a Cell Contain a Formula?	654
	Returning a Cell's Formula	654
	Is the Cell Hidden?	654
	Returning a Worksheet Name	655
	Returning a Workbook Name	656
	Returning the Application's Name	657
	Returning Excel's Version Number	657
	Returning Cell Formatting Information	657
	Determining a Cell's Data Type	659
	A Multifunctional Function	660
	Generating Random Numbers	662
	Generating Random Numbers That Don't Change	663
	Selecting a Cell at Random	663
	Calculating Sales Commissions	664
	A Function for a Simple Commission Structure	665
	A Function for a More Complex Commission Structure	666
	Text Manipulation Functions	668
	Reversing a String	668
	Scrambling Text	669
	Returning an Acronym	669
	Does the Text Match a Pattern?	670
	Does a Cell Contain Text?	671
	Extracting the nth Element from a String	672
	Spelling Out a Number	674
	Counting and Summing Functions	674
	Counting Cells Between Two Values	675
	Counting Visible Cells in a Range	675
	Summing Visible Cells in a Range	676
	Date Functions	677
	Calculating the Next Monday	677
	Calculating the Next Day of the Week	678
	Which Week of the Month?	678
	Working with Dates Before 1900	679

Returning the Last Nonempty Cell	
in a Column or Row	680
The LASTINCOLUMN Function	680
The LASTINROW Function	681
Multisheet Functions	681
Returning the Maximum Value Across All Worksheets	682
The SHEETOFFSET Function	683
Advanced Function Techniques	685
Returning an Error Value	685
Returning an Array from a Function	687
Returning an Array of Nonduplicated Random Integers	689
Randomizing a Range	691
Using Optional Arguments	693
Using an Indefinite Number of Arguments	694

Appendixes

Appendix A	Working with Imported 1-2-3 Files	699
	About 1-2-3 Files	699
	Lotus 1-2-3 Formulas	700
	Calculation Order	701
	Text in Calculations	702
	Logical Values	702
	Date Problems	703
	Database Criteria	703
	Lotus 1-2-3 Function Compatibility	704
	Function Equivalents	704
	Converting Database Functions	714
Appendix B	Excel Function Reference	717
	Excel Functions by Category	717
Appendix C	Using Custom Number Formats	733
	About Number Formatting	733
	Automatic Number Formatting	733
	Formatting Numbers Using Toolbar Buttons	734
	Using Shortcut Keys to Format Numbers	735
	Using the Format Cells Dialog Box to Format Numbers	736
	Creating a Custom Number Format	738
	About Custom Number Formats	738
	Parts of a Number Format String	739
	Custom Number Format Codes	740
	Custom Number Format Examples	742
	Scaling Values	742
	Hiding Zeros	746
	Displaying Leading Zeros	747
	Formatting Percentages	747

	Displaying Fractions	748
	Displaying N/A for Text	749
	Displaying Text in Quotes	749
	Repeating Text	749
	Displaying a Negative Sign on the Right	750
	Conditional Number Formatting	750
	Coloring Values	751
	Formatting Dates and Times	752
	Displaying Text with Numbers	753
	Displaying a Zero with Dashes	753
	Using Special Symbols	754
	Suppressing Certain Types of Entries	755
	Filling a Cell with a Repeating Character	756
	Displaying Leading Dots	757
Appendix D	Additional Excel Resources	759
	Microsoft Technical Support	759
	Support Options	759
	Microsoft Knowledge Base	759
	Microsoft Excel Home Page	760
	Microsoft Office Tools on the Web	760
	Internet Newsgroups	760
	Spreadsheet Newsgroups	761
	Microsoft Newsgroups	761
	Searching Newsgroups	762
	Internet Web sites	763
	The Spreadsheet Page	763
	Excel Web Source	764
	Stephen Bullen's Excel Page	764
	Spreadsheet FAQ	764
Appendix E	What's on the CD-ROM	765
	CD-ROM Overview	765
	Chapter Examples	766
	Chapter 5	766
	Chapter 6	766
	Chapter 7	767
	Chapter 8	767
	Chapter 9	768
	Chapter 10	768
	Chapter 11	768
	Chapter 12	769
	Chapter 13	769
	Chapter 15	770
	Chapter 16	770
	Chapter 17	771
	Chapter 18	772

Chapter 19	772
Chapter 20	772
Chapter 25	773
Power Utility Pak	773
Registering Power Utility Pak	774
Installing the trial version	774
Uninstalling Power Utility Pak	774
Sound-Proof 2000	775
Installing the demo version	775
Uninstalling Sound-Proof	776
Electronic Version of Excel 2002 Formulas	776
Adobe Acrobat Reader	776
Index.....	777
End-User License Agreement.....	825
CD-ROM Installation Instructions	828

Part I

Basic Information

CHAPTER 1

Excel in a Nutshell

CHAPTER 2

Basic Facts about Formulas

CHAPTER 3

Working with Names

Chapter 1

Excel in a Nutshell

IN THIS CHAPTER

- ◆ A brief history of Excel
- ◆ The object model concept in Excel
- ◆ The workings of workbooks
- ◆ The user interface
- ◆ The two types of cell formatting
- ◆ Worksheet formulas and functions
- ◆ Objects on the worksheet's invisible drawer layer
- ◆ Macros, toolbars, and add-ins for Excel customization
- ◆ Analysis tools
- ◆ Protection options

MICROSOFT EXCEL HAS BEEN REFERRED TO as “the best application ever written for Windows.” You may or may not agree with that statement, but you can’t deny that Excel is one of the *oldest* Windows products and has undergone many reincarnations and face-lifts over the years. Cosmetically, the current version – Excel 2002 – barely even resembles the original version (which, by the way, was written for the Macintosh). However, many of Excel’s key elements have remained intact over the years, with significant enhancements, of course.

This chapter presents a concise overview of the features available in the more recent versions of Excel, with specific emphasis on Excel 2002. It sets the stage for the subsequent chapters and provides a transition for those who have used other spreadsheet products and are moving up to Excel. Hard-core Lotus 1-2-3 users, for example, usually need some help to start thinking in Excel’s terms.



If you’re an old hand at Excel, you may want to ignore this chapter or just skim through it quickly.

The History of Excel

You probably weren't expecting a history lesson when you bought this book, but you may find this information interesting. At the very least, this section provides fodder for the next office trivia match.

Spreadsheets comprise a huge business, but most of us tend to take this software for granted. In the pre-spreadsheet days, people relied on clumsy mainframes or calculators and spent hours doing what now takes minutes.

It Started with VisiCalc

Dan Bricklin and Bob Frankston conjured up VisiCalc, the world's first electronic spreadsheet, back in the late 1970s when personal computers were unheard of in the office environment. They wrote VisiCalc for the Apple II computer, an interesting machine that seems like a toy by today's standards. VisiCalc caught on quickly, and many forward-looking companies purchased the Apple II for the sole purpose of developing their budgets with VisiCalc. Consequently, VisiCalc is often credited for much of Apple II's initial success.

Then Came Lotus

When the IBM PC arrived on the scene in 1982, thus legitimizing personal computers, VisiCorp wasted no time porting VisiCalc to this new hardware environment. Envious of VisiCalc's success, a small group of computer enthusiasts at a start-up company in Cambridge, Massachusetts, refined the spreadsheet concept. Headed by Mitch Kapor and Jonathon Sachs, the company designed a new product and launched the software industry's first full-fledged marketing blitz. Released in January 1983, Lotus Development Corporation's 1-2-3 proved an instant success. Despite its \$495 price tag (yes, people really paid that much for software), it quickly outsold VisiCalc and rocketed to the top of the sales charts, where it remained for many years. Lotus 1-2-3 was, perhaps, the most popular application ever.

Microsoft Enters the Picture

Most people don't realize that Microsoft's experience with spreadsheets extends back to the early 1980s. In 1982, Microsoft released its first spreadsheet – MultiPlan. Designed for computers running the CP/M operating system, the product was subsequently ported to several other platforms, including Apple II, Apple III, XENIX, and MS-DOS. MultiPlan essentially ignored existing software user-interface standards. Difficult to learn and use, it never earned much of a following in the United States. Not surprisingly, Lotus 1-2-3 pretty much left MultiPlan in the dust.

Excel partly evolved from MultiPlan, first surfacing in 1985 on the Macintosh. Like all Mac applications, Excel was a graphics-based program (unlike the character-based MultiPlan). In November 1987, Microsoft released the first version of Excel for

Windows (labeled Excel 2 to correspond with the Macintosh version). Excel didn't catch on right away, but as Windows gained popularity, so did Excel. Lotus eventually released a Windows version of 1-2-3, and Excel had additional competition from Quattro Pro – originally a DOS program developed by Borland International, then sold to Novell, and then sold again to Corel (its current owner).

Excel Versions

Excel 2002 is actually Excel 10 in disguise. You may think that this name represents the tenth version of Excel. Think again. Microsoft may be a successful company, but their version-naming techniques can prove quite confusing. As you'll see, Excel 2002 actually represents the eighth Windows version of Excel. In the following sections, I briefly describe the major Windows versions of Excel.

EXCEL 2

The original version of Excel for Windows, Excel 2 first appeared in late 1987. It was labeled Version 2 to correspond to the Macintosh version (the original Excel). Because Windows wasn't in widespread use at the time, this version included a *runtime* version of Windows – a special version with just enough features to run Excel and nothing else. This version appears quite crude by today's standards, as shown in Figure 1-1.

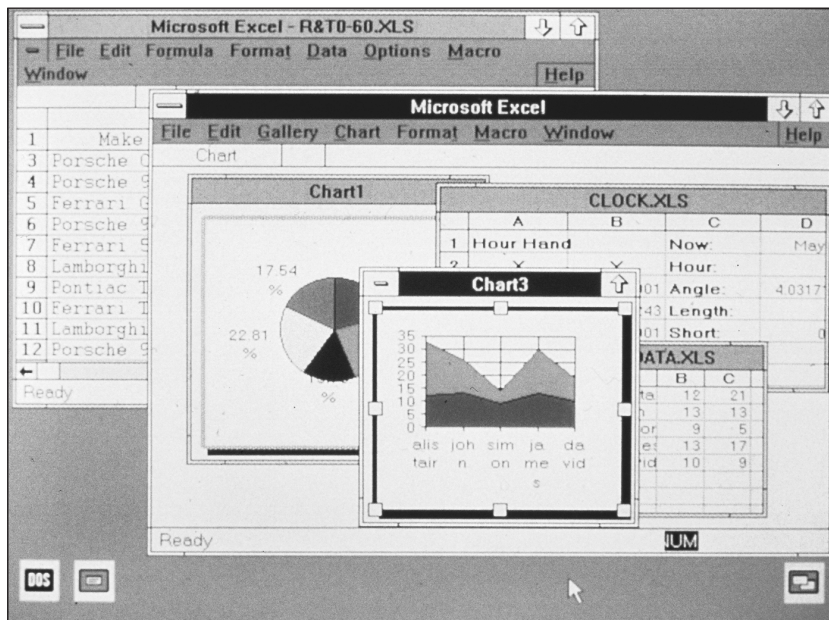


Figure 1-1: The original Excel 2 for Windows. Excel has come a long way since its original version. (Photo courtesy of Microsoft Corporation)

EXCEL 3

At the end of 1990, Microsoft released Excel 3 for Windows. This version offered a significant improvement in both appearance and features. It included toolbars, drawing capabilities, worksheet outlining, add-in support, 3-D charts, workgroup editing, and lots more.

EXCEL 4

Excel 4 hit the streets in the spring of 1992. This version made quite an impact on the marketplace as Windows increased in popularity. It boasted lots of new features and “usability” enhancements that made it easier for beginners to get up to speed quickly.

EXCEL 5

In early 1994, Excel 5 appeared on the scene. This version introduced tons of new features, including multisheet workbooks and the new Visual Basic for Applications (VBA) macro language. Like its predecessor, Excel 5 took top honors in just about every spreadsheet comparison published in the trade magazines.

EXCEL 95

Excel 95 (also known as Excel 7) shipped in the summer of 1995. On the surface, it resembled Excel 5 (this version included only a few major new features). But Excel 95 proved to be significant because it presented the first version to use more advanced 32-bit code. Excel 95 and Excel 5 use the same file format.

EXCEL 97

Excel 97 (also known as Excel 8) probably offered the most significant upgrade ever. The toolbars and menus took on a great new look, online help moved a dramatic step forward, and the number of rows available in a worksheet quadrupled. And if you’re a macro developer, you may have noticed that Excel’s programming environment (VBA) moved up several notches on the scale. Excel 97 also introduced a new file format.

EXCEL 2000

Excel 2000 (also known as Excel 9) was released in June of 1999. Excel 2000 offered several minor enhancements, but the most significant advancement was the ability to use HTML as an alternative file format. Excel 2000 still supported the standard binary file format, of course, which is compatible with Excel 97.

EXCEL 2002

The most recent version, Excel 2002 (also known as Excel 10) was released in June of 2001. It is sold as part of Microsoft Office XP. This version offers several new features, most of which are fairly minor and are designed to appeal to novice users. Perhaps the most significant new feature is the capability to save your work when Excel crashes, and also recover corrupt workbook files that you may have abandoned long ago. Excel 2002 also adds background formula error checking and a new formula-debugging tool; both features are relevant to this book.



Many of these versions of Excel also included *sub-versions*. For example, Microsoft released two service releases (SR-1 and SR-2) for Excel 97. These service releases correct various problems with the software.

The Object Model Concept

If you've dealt with computers for any length of time, you've undoubtedly heard the term *object-oriented programming*. An *object* essentially represents a software element that a programmer can manipulate. When using Excel, you may find it useful to think in terms of objects, even if you have no intention of becoming a programmer. An object-oriented approach can often help you keep the various elements in perspective.

Excel objects include the following:

- ◆ Excel itself
- ◆ An Excel workbook
- ◆ A worksheet in a workbook
- ◆ A range in a worksheet
- ◆ A button on a worksheet
- ◆ A ListBox control on a UserForm (a custom dialog box)
- ◆ A chart sheet
- ◆ A chart on a chart sheet
- ◆ A chart series in a chart

Notice that something of an *object hierarchy* exists here: The Excel object contains workbook objects, which contain worksheet objects, which contain range objects. This hierarchy is called Excel's *object model*. Other Microsoft Office products have their own object model. The object model concept proves to be vitally important when developing VBA macros. Even if you don't create macros, you may find it helpful to think in terms of objects.

The Workings of Workbooks

One of the most common Excel objects is a *workbook*. Everything that you do in Excel takes place in a workbook, which is stored in a file with an .xls extension.

Where Are the VBA Module Sheets?

In Excel 5 and Excel 95, a VBA module appeared in a workbook as a separate sheet. A VBA module, as you may know, holds VBA code. In Excel 97 and later versions, VBA modules still store with a workbook, but they no longer show up as a separate sheet. Rather, you work with VBA modules in the Visual Basic Editor (VB Editor). To view or edit a VBA module, activate the VB Editor by pressing Alt+F11. See Part VI of this book for more information about VBA.



Beginning with Excel 2000, you can also use HTML as a “native” file format for Excel. Because this file must store lots of information needed to recreate the workbook, you’ll find that the HTML files generated by Excel are very bloated. So unless you have a real need to save your work in HTML by using this feature, you should use the normal XLS file format.

An Excel workbook can hold any number of sheets (limited only by memory). The four types of sheets are:

- ◆ Worksheets
- ◆ Chart sheets
- ◆ XLM macro sheets (obsolete, but still supported)
- ◆ Dialog sheets (obsolete, but still supported)

You can open as many workbooks as you want (each in its own window), but only one workbook is the *active workbook* at any given time. Similarly, only one sheet in a workbook is the *active sheet*. To activate a different sheet, click its corresponding tab at the bottom of the window, or press Ctrl+PgUp (for the next sheet) or Ctrl+PgDn (for the previous sheet). To change a sheet’s name, double-click its Sheet tab and enter the new text for the name. Right-clicking a tab brings up a shortcut menu with some additional sheet-manipulation options.

You can also hide the window that contains a workbook by using the **Window** → **Hide** command. A hidden workbook window remains open, but not visible. A single workbook can display in multiple windows (select **Window** → **New Window**). Each window can display a different sheet.

Worksheets

The most common type of sheet is a worksheet – which you normally think of when you think of a spreadsheet. Every Excel worksheet has 256 columns and 65,536 rows. And to answer a common question, the number of rows and columns is permanently fixed; you can't change it. Despite what must amount to thousands of requests from users, Microsoft refuses to increase the number of rows and columns in a workbook. You can hide unneeded rows and columns to keep them out of view, but you can't increase the number of rows or columns.



Versions prior to Excel 97 support only 16,384 rows in a worksheet.

Having access to more cells isn't the *real* value of using multiple worksheets in a workbook. Rather, multiple worksheets are valuable because they enable you to organize your work better. Back in the old days, when a spreadsheet file consisted of a single worksheet, developers wasted a lot of time trying to organize the worksheet to hold their information efficiently. Now, you can store information on any number of worksheets and still access it instantly.

You have complete control over the column widths and row heights and you can even hide rows and columns (as well as entire worksheets). You can display the contents of a cell vertically (or at an angle) and even wrap around to occupy multiple lines.

How Big Is a Worksheet?

It's interesting to stop and think about the actual size of a worksheet. Do the arithmetic ($256 \times 65,536$), and you'll see that a worksheet has 16,777,216 cells. Remember that this is in just one worksheet. A single workbook can hold more than one worksheet.

If you're using the standard VGA video mode with the default row heights and column widths, you can see 9 columns and 18 rows (or 162 cells) at a time. This works out to be less than 0.001 percent of the entire worksheet. In other words, nearly 104,000 VGA screens of information reside within a single worksheet.

If you entered a single digit into each cell at the relatively rapid clip of one cell per second, it would take you about 194 days, nonstop, to fill up a worksheet. To print the results of your efforts would require more than 36,000 sheets of paper – a stack about six feet high.



By default, every new workbook starts out with three worksheets. You can easily add a new sheet when necessary, so you really don't need to start with three sheets. You may want to change this default to a single sheet. To change this option, use the Tools → Options command, click the General tab, and change the setting for Sheets in new workbook.

Chart Sheets

A chart sheet normally holds a single chart. Many users ignore chart sheets, preferring to use “embedded charts,” which are stored on the worksheet's draw layer. Using chart sheets is optional, but they make it a bit easier to print a chart on a page by itself, and they prove especially useful for presentations. I discuss embedded charts (or *floating charts* on a worksheet) later in this chapter.

XLM Macro Sheets

An XLM macro sheet (also known as an MS Excel 4 macro sheet) is essentially a worksheet, but it has some different defaults. More specifically, an XLM macro sheet displays formulas rather than the results of formulas. Also, the default column width is larger than in a normal worksheet.

As the name suggests, an XLM macro sheet is designed to hold XLM macros. As you may know, the XLM macro system consists of a holdover from previous versions (version 4.0 or earlier) of Excel. However, Excel 2002 continues to support XLM macros for compatibility reasons, but it no longer provides the option of recording an XLM macro. This book doesn't cover the XLM macro system; instead, it focuses on the more powerful VBA macro system.

Dialog Sheets

In Excel 5 and Excel 95, you can create a custom dialog box by inserting a special dialog sheet. When you open a workbook that contains an Excel 5/95 dialog sheet, the dialog sheet appears as a sheet in the workbook. Excel 97 and later versions still support these dialog sheets, but they provide a much better alternative: UserForms. You can work with UserForms in the VB Editor.



If, for compatibility purposes, you need to insert an Excel 5/95 dialog sheet in later versions of Excel, you won't find the command to do so on the Insert menu. You can only add an Excel 5/95 dialog sheet by right-clicking any Sheet tab and selecting Insert from the shortcut menu. Then, in the Insert dialog box, click the MS Excel 5.0 Dialog icon.

Excel's User Interface

A *user interface* (UI) is the means by which an end user communicates with a computer program. A UI includes elements, such as menus, dialog boxes, toolbars, and keystroke combinations, as well as features such as drag-and-drop. For the most part, Excel uses the standard Windows UI to accept commands.

Menus

Beginning with Excel 97, Excel's UI deviates from the standard Windows UI by providing non-standard Windows menus. The menus in Excel 2000 and Excel 97 are actually toolbars in disguise—the icons that accompany some menu items are a dead give-away.

Excel's menu system is relatively straightforward. Excel contains two different menu bars—one for an active worksheet, the other for an active chart sheet or embedded chart. Consistent with Windows conventions, inappropriate menu commands are dimmed (“grayed out”) and commands that open a dialog box are followed by an ellipsis (three dots). Where appropriate, the menus list any available shortcut key combinations (for example, the Edit menu lists Ctrl+Z as the shortcut key for Edit → Undo).

Several menu items are *cascading menus*, and as such, lead to submenus that have additional commands (Edit → Fill represents a cascading menu, for example). A small arrow on the right of the menu item text indicates cascading menus.



An end user or developer can customize the entire menu system. To do so, choose the View → Toolbars → Customize command. You must understand that menu changes made by using this technique are “permanent.” In other words, the menu changes will remain in effect even if you close Excel and restart it. This differs greatly from the Menu Editor found in Excel 5 and Excel 95, which is not available in Excel 97 and later versions.

Shortcut Menus

Excel also features dozens of shortcut menus. These menus appear when the user right-clicks after selecting one or more objects. The shortcut menus are context-sensitive. In other words, the menu that appears depends on the location of the mouse pointer when you right-click. You can right-click just about anything—a cell, a row or column border, a workbook title bar, a toolbar, and so on.

Smart Tags

A Smart Tag is a small icon that appears automatically in your worksheet. Clicking a Smart Tag reveals several clickable options.



Smart Tags are available only in Excel 2002.

For example, if you copy and paste a range of cells, Excel generates a Smart Tag that appears below the pasted range (see Figure 1-2). Excel 2002 features several other Smart Tags, and additional Smart Tags can be provided by third-party providers.

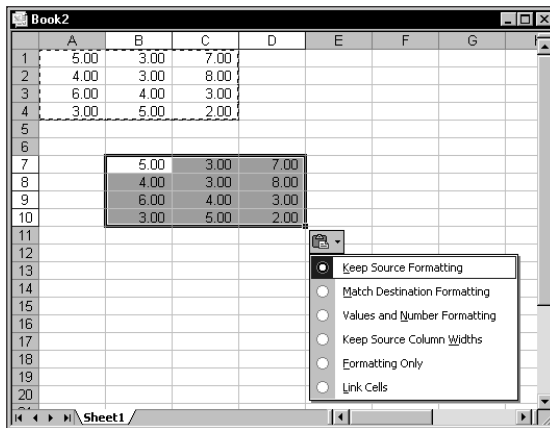


Figure 1-2: This Smart Tag appears when you paste a copied range.

Dialog Boxes

Most of the menu commands in Excel display a dialog box, in which you can clarify your intentions. These dialog boxes remain quite consistent in terms of how they operate. Some of Excel's dialog boxes use a notebook tab metaphor, which makes a single dialog box function as several different dialog boxes. Tabbed dialog boxes provide access to many options without overwhelming you. The Options dialog box (choose Tools → Options) presents an example of a tabbed dialog box in Excel 2002 (see Figure 1-3).

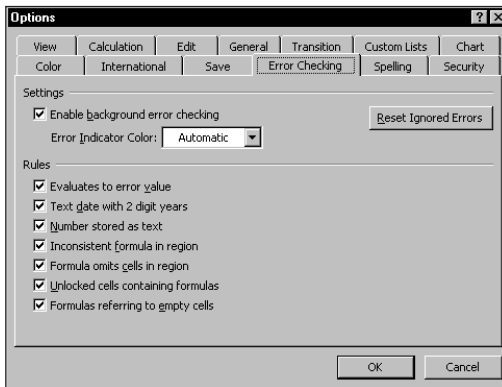


Figure 1-3: The Options dialog box represents a type of tabbed dialog box.

Toolbars

Excel 2002 ships with 54 predefined toolbars (including the two toolbars that function as menus). These toolbars typically appear automatically, when appropriate. For example, if you activate a chart, the Chart toolbar displays.

You can *dock* toolbars (position them along any edge of the screen) or make them *float*. By default, Excel displays the Standard and Formatting toolbars directly below the menu bar.

Drag-and-Drop

Excel's drag-and-drop UI feature enables you to freely drag objects that reside on the draw layer to change their position. Pressing Ctrl while dragging duplicates the selected objects.

Excel also permits drag-and-drop actions on cells and ranges. You can easily drag a cell or range to a different position. And pressing Ctrl while dragging copies the selected range.



Cell drag-and-drop is optional; you can disable it in the Edit tab of the Options dialog box.

Keyboard Shortcuts

Excel has many keyboard shortcuts. For example, you can press Ctrl+C to copy a selection. If you're a newcomer to Excel or if you just want to improve your efficiency, then do yourself a favor and check out the online help (search for *Keyboard Shortcuts*). The help file contains tables that summarize useful keyboard commands and shortcuts.

Customized On-screen Display

Excel offers a great deal of flexibility regarding on-screen display (status bar, formula bar, toolbars, and so on). For example, by choosing View → Full Screen, you can get rid of everything except the menu bar, thereby maximizing the amount of visible information. In addition, by using the View tab in the Options dialog box, you can customize what displays in a worksheet window (for example, you can hide scroll bars and grid lines).

Data Entry

Data entry in Excel is quite straightforward. Excel interprets each cell entry as one of the following:

- ◆ A value (including a date or a time)
- ◆ Text
- ◆ A Boolean value (TRUE or FALSE).
- ◆ A formula

Formulas always begin with an equal sign (=). Excel accommodates habitual 1-2-3 users, however, and accepts an “at” symbol (@), a plus sign (+), or a minus sign (-) as the first character in a formula. It automatically adjusts the entry after you press Enter.

Object and Cell Selecting

Generally, selecting objects in Excel conforms to standard Windows practices. You can select a range of cells by using the keyboard (using the Shift key, along with the arrow keys), or by clicking and dragging the mouse. To select a large range, click a cell at any corner of the range, scroll to the opposite corner of the range, and press Shift while you click the opposite corner cell.

You can use Ctrl+* (Ctrl asterisk) to select an entire table. And when a large range is selected, you can use Ctrl+. (Ctrl period) to move among the four corners of the range.

Clicking an object placed on the draw layer selects the object. An exception occurs if the object has a macro assigned to it. In such a case, clicking the object

Data-Entry Tips

The following list of data-entry tips can help those moving up to Excel from another spreadsheet.

- ◆ To enter data without pressing the arrow keys, enable the Move selection after entering an option in the Edit tab of the Options dialog box (which you access from the Tools → Options command). You can also choose the direction that you want to go.
- ◆ You may find it helpful to select a range of cells before entering data. If you do so, you can use the Tab key to move only within the selected cells.
- ◆ To enter the same data in all cells within a range, select the range, enter the information into the active cell, and then press Ctrl+Enter.
- ◆ To copy the contents of the active cell to all other cells in a selected range, press F2 and then Ctrl+Enter.
- ◆ To fill a range with increments of a single value, press Ctrl while you drag the fill handle at the lower-right corner of the cell.
- ◆ To create a custom AutoFill list, use the Custom Lists tab of the Options dialog box.
- ◆ To copy a cell without incrementing, drag the fill handle at the lower-right corner of the selection; or press Ctrl+D to copy down or Ctrl+R to copy to the right.
- ◆ To make text easier to read, you can enter carriage returns in a cell. To enter a carriage return, press Alt+Enter. Carriage returns cause a cell's contents to wrap within the cell.
- ◆ To enter a fraction, enter 0, a space, and then the fraction (using a slash). Excel formats the cell using the Fraction number format.
- ◆ To automatically format a cell with the currency format, type a dollar sign before the value.
- ◆ To enter a value in percent format, type a percent sign after the value. You can also include your local thousand separator symbol to separate thousands (for example, 123,434).
- ◆ To insert the current date, Press Ctrl+semicolon. To enter the current time into a cell, press Ctrl+Shift+semicolon.
- ◆ To set up a cell or range so that it only accepts entries of a certain type (or within a certain value range), use the Data → Validation command.

executes the macro. To select multiple objects or noncontiguous cells, press Ctrl while you select the objects or cells.

Cell Formatting

Excel provides two types of cell formatting—numeric formatting and stylistic formatting.

Numeric Formatting

Numeric formatting refers to how a value appears in the cell. In addition to choosing from an extensive list of predefined formats, you can create your own custom number formats in the Number tab of the Format Cells dialog box (choose Format → Cells).

Excel applies some numeric formatting automatically, based on the entry. For example, if you precede a value with your local currency symbol (such as a dollar sign), Excel applies Currency number formatting.



Refer to Appendix C for additional information about creating custom number formats.

The number format doesn't affect the actual value stored in the cell. For example, suppose that a cell contains the value 3.14159. If you apply a format to display two decimal places, the number appears as 3.14. When you use the cell in a formula, however, the actual value (3.14159) — not the displayed value — is used.

Stylistic Formatting

Stylistic formatting refers to the cosmetic formatting (colors, shading, fonts, borders, and so on) that you apply in order to make your work look good. The Format Cells dialog box (see Figure 1-4) is your one-stop shopping place for formatting cells and ranges.

Many toolbar buttons offer direct access to common formatting options, regardless of whether you work with cells, drawn objects, or charts. For example, you can use the Fill Color toolbar button to change the background color of a cell, change the fill color of a drawn text box, or change the color of a bar in a chart. Access the Format dialog box for the full range of formatting options.

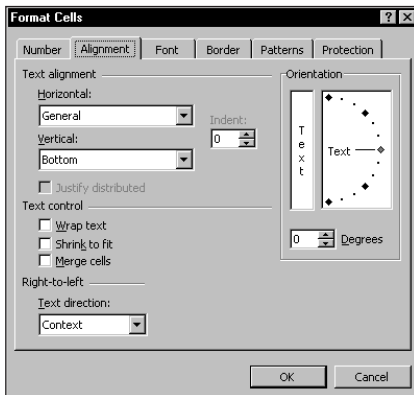


Figure 1-4: Use the Format Cells dialog box to apply stylistic formatting.

Each type of object has its own Format dialog box. You can easily get to the correct dialog box and format an object by selecting the object, right-clicking, and then choosing Format .xxx (where .xxx is the selected object) from the shortcut menu. Alternatively, you can press Ctrl+1. Either of these actions leads to a tabbed dialog box that holds all the formatting options for the selected object.

Don't overlook Excel's conditional formatting feature. This handy tool enables you to specify formatting that appears only when certain conditions are met. For example, you can make the cell's interior red if the cell contains a negative number.



Chapter 19 describes how to create conditional formatting formulas that greatly enhance this feature.

Worksheet Formulas and Functions

Formulas, of course, make a spreadsheet a spreadsheet. Excel's formula-building capability is as good as it gets. You will discover this as you explore subsequent chapters in this book.

Worksheet functions allow you to perform calculations or operations that would otherwise be impossible. Excel provides a huge number of built-in functions, and you can access even more functions (many of them quite esoteric) by attaching the Analysis ToolPak add-in.



See Chapter 4 for more information about worksheet functions.

All spreadsheets allow you to define names for cells and ranges, but Excel handles names in some unique ways. A *name* represents an identifier that enables you to refer to a cell, range, value, or formula. Using names makes your formulas easier to create and read.



I devote Chapter 3 entirely to names.

Objects on the Draw Layer

As I mentioned earlier in this chapter, each worksheet has an invisible draw layer, which holds shapes, diagrams, charts, maps, pictures, and controls (such as buttons and list boxes). I discuss some of these items in the following sections.

Shapes

You can insert AutoShapes from the Drawing toolbar. You can choose from a huge assortment of shapes. After you place a shape on your worksheet, you can modify the shape by selecting it and dragging its handles. In addition, you can apply drop shadows, text, or 3-D effects to the shape. Also, you can group multiple shapes into a single drawing object, which you'll find easier to size or position.

Diagrams

The Insert → Diagram command displays the Diagram Gallery dialog box, shown in Figure 1-5. You can choose from six diagrams, and each is highly customizable.



The Diagram Gallery is new to Excel 2002.

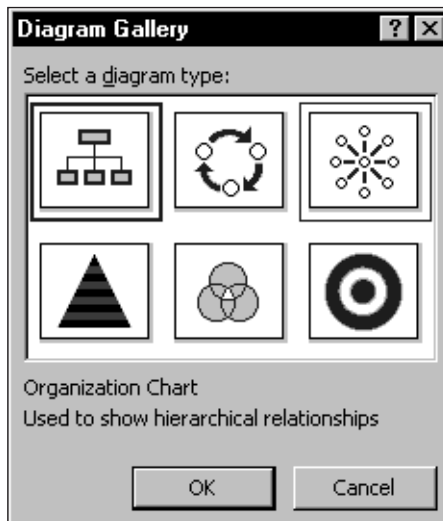


Figure 1-5: Excel 2002 supports several types of diagrams.

Linked Picture Objects

For some reason, the designers of Excel make the *linked picture object* rather difficult to generate. To use this object, copy a range and then press Shift and select the Edit → Paste Picture Link command (which appears on the Edit menu only when you press Shift). This command originally accommodated users who wanted to print a noncontiguous selection of ranges. Users could “take pictures” of the ranges and then paste the pictures together in a single area, which they could then print.

Maps

If you work with geographic data, you may like the ability to insert a map into a worksheet by using the Insert → Map command. Unfortunately, this feature was removed from Excel 2002. Maps that were created with earlier versions still appear in Excel 2002, but they can't be modified.

Dialog Box Controls

Many of the controls that are used in custom dialog boxes can be placed directly on the draw layer of a worksheet. Doing this can greatly enhance the usability of some worksheets and eliminate the need to create custom dialog boxes. Figure 1-6 shows a worksheet with some dialog box controls added to the draw layer.



Dialog box controls come from two sources: The Forms toolbar, or the Control Toolbox toolbar. Controls from the Control Toolbox toolbar consist of ActiveX controls, and are available only in Excel 97 or later.

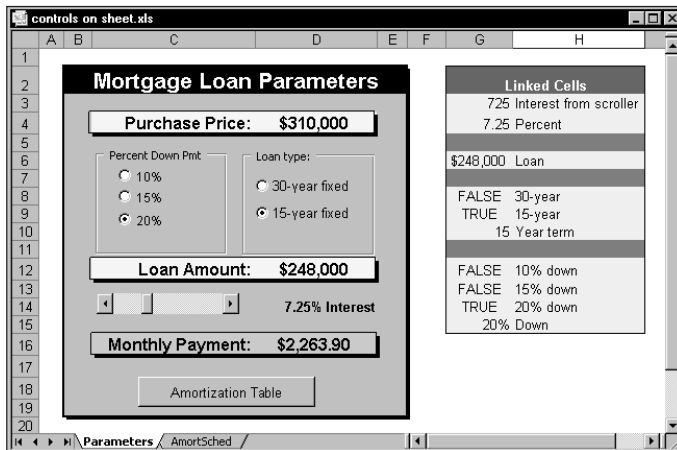


Figure 1-6: Excel enables you to add many controls directly to the draw layer of a worksheet.

Charts

Excel, of course, has excellent charting capabilities. As I mentioned earlier in this chapter, you can store charts on a chart sheet or you can float them on a worksheet.

Excel offers extensive chart customization options. If a chart is free-floating, just click a chart element to select it (or double-click it to display its formatting dialog box). Right-clicking a chart element displays a shortcut menu.

You can easily create a free-floating chart by selecting the data to be charted and then using the Chart Wizard to walk you through the steps to create a chart that meets your needs.



Chapter 17 contains additional information about charts.

Customization in Excel

This section describes various features that enable you to customize Excel. They include macros, toolbars, and add-in programs.

Macros

Excel's VBA programming language provides a powerful tool that can make Excel perform otherwise impossible feats. You can classify the procedures that you create with VBA into two general types:

- ◆ Macros that automate various aspects of Excel.
- ◆ Macros that serve as custom functions that you can use in worksheet formulas.



Part VI of this book describes how to use and create custom worksheet functions using VBA.

Toolbars

As I noted earlier, Excel includes many toolbars. You can, if you're so inclined, create new toolbars that contain existing toolbar buttons, or new buttons that execute macros.

Use the View → Toolbars → Customize command to customize toolbars or create new ones. You can also write VBA code to manipulate toolbars.

Add-in Programs

An *add-in* is a program attached to Excel that gives it additional functionality. For example, you can store custom worksheet functions in an add-in. To attach an add-in, use the Tools → Add-Ins command.

Excel ships with quite a few add-ins (including the Analysis ToolPak). In addition to these add-ins, you can purchase or download many third-party add-ins from online services. My Power Utility Pak represents an example of an add-in. You can access a trial version on the CD-ROM included with this book.



Chapter 23 describes how to create your own add-ins that contain custom worksheet functions.

Analysis Tools

Excel is certainly no slouch when it comes to analysis. After all, most people use a spreadsheet for analysis. Many analysis tasks can be handled with formulas, but Excel offers many other options, which I discuss in the following sections.

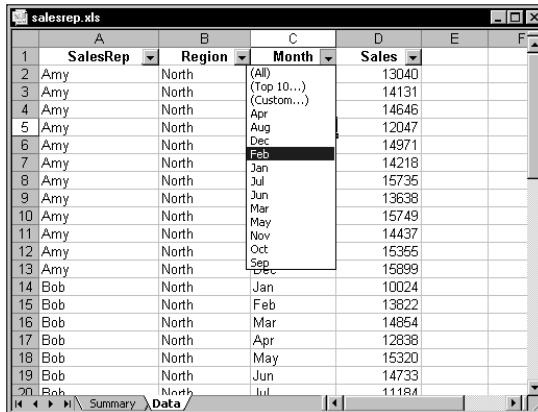
Database Access

Over the years, most spreadsheets have enabled users to work with simple flat database tables (even the original version of 1-2-3 contained this feature). Excel's database features fall into two main categories:

- ◆ **Worksheet databases.** The entire database stores in a worksheet, limiting the size of the database. In Excel, a worksheet database can have no more than 65,535 records (because there are 65,536 rows; the top row holds the field names) and 256 fields (because there are 256 columns).
- ◆ **External databases.** The data stores in one or more disk files and you can access it as needed.

Generally, when the cell pointer resides within a worksheet database, Excel recognizes it and displays the field names whenever possible. For example, if you move the cell pointer within a worksheet database and choose the Data → Sort command, Excel enables you to select the sort keys by choosing field names from a drop-down list.

A particularly useful feature, Excel's AutoFilter, enables you to display only the records that you want to see. When AutoFilter mode is on, you can filter the data by selecting values from pull-down lists (which appear in place of the field names when you choose the Data → Filter → AutoFilter command). Rows that don't qualify are temporarily hidden. See Figure 1-7 for an example.



	A	B	C	D	E	F
1	SalesRep	Region	Month	Sales		
2	Amy	North	(All)	13040		
3	Amy	North	(Top 10...)	14131		
4	Amy	North	(Custom...)	14646		
5	Amy	North	Apr	12047		
6	Amy	North	Aug	14971		
7	Amy	North	Feb	14218		
8	Amy	North	Jan	15735		
9	Amy	North	Jun	13638		
10	Amy	North	Mar	15749		
11	Amy	North	May	14437		
12	Amy	North	Nov	15355		
13	Amy	North	Oct	15899		
14	Bob	North	Dec	10024		
15	Bob	North	Jan	13822		
16	Bob	North	Feb	14854		
17	Bob	North	Mar	12838		
18	Bob	North	Apr	15320		
19	Bob	North	May	14733		
20	Bob	North	Jun	11184		

Figure 1-7: Excel's AutoFilter feature makes it easy to view only the database records that meet your criteria.

If you prefer, you can use the traditional spreadsheet database techniques that involve criteria ranges. To do so, choose the Data → Filter → Advanced Filter command.



Chapter 9 provides additional details regarding worksheet lists and databases.

Excel can automatically insert (or remove) subtotal formulas in a table that is set up as a database. It also creates an outline from the data so that you can view only the subtotals, or any level of detail that you desire.

Outlines

A worksheet outline often serves as an excellent way to work with hierarchical data, such as budgets. Excel can create an outline automatically by examining the formulas in your worksheet. After you've created an outline, you can collapse or expand the outline to display various levels of details. Figure 1-8 shows an example of a worksheet outline.

1	A	B	C	D	E	I	M	Q
2	State	Jan	Feb	Mar	Q1 Total	Q2 Total	Q3 Total	Q4 Total
6	West Total	5,170	6,527	5,081	16,778	18,242	18,314	19,138
7	New York	1,429	1,316	1,993	4,738	4,763	3,915	4,289
8	New Jersey	1,735	1,406	1,224	4,365	4,316	4,051	4,526
9	Massachusetts	1,099	1,233	1,110	3,442	4,155	4,882	5,376
10	Florida	1,705	1,792	1,225	4,722	4,630	5,062	4,734
11	East Total	5,968	5,747	5,552	17,267	17,864	17,910	18,925
12	Kentucky	1,109	1,078	1,155	3,342	4,626	4,306	4,526
13	Oklahoma	1,309	1,045	1,641	3,995	5,364	4,338	4,449
14	Missouri	1,511	1,744	1,414	4,669	4,556	4,203	4,279
15	Illinois	1,539	1,493	1,211	4,243	3,623	4,889	4,330
16	Kansas	1,973	1,560	1,243	4,776	4,007	4,322	4,299
17	Central Total	7,441	6,920	6,664	17,683	17,550	17,752	17,357
18	Grand Total	18,579	19,194	17,297	51,728	53,656	53,976	55,420

Figure 1-8: Excel can automatically insert subtotal formulas and create outlines.

Scenario Management

Scenario management is the process of storing input values that drive a model. For example, if you have a sales forecast, you may create scenarios such as best case, worst case, and most likely case.

If you seek the ultimate in scenario-management features, 1-2-3's Version Manager is probably your best bet. Unlike Version Manager, Excel's Scenario Manager can only handle simple scenario-management tasks. However, it is definitely easier than trying to keep track of different scenarios manually.

Analysis ToolPak

The Analysis ToolPak add-in provides 19 special-purpose analysis tools (primarily statistical in nature) and many specialized worksheet functions. These tools make Excel suitable for small- to medium-scale statistical analysis.

Pivot Tables

One of Excel's most powerful tools is its pivot tables. A *pivot table* enables you to display summarized data in just about any possible way. Data for a pivot table comes from a worksheet database or an external database and stores in a special cache, which enables Excel to recalculate data rapidly after a pivot table is altered.



Chapter 18 contains additional information about pivot tables.

Excel 2000 and later versions also support the pivot chart feature. Pivot charts enable you to link a chart to a pivot table.

Auditing Capabilities

Excel also offers useful auditing capabilities that help you identify errors or track the logic in an unfamiliar spreadsheet. To access this feature, select Tools → Formula Auditing (or Tools → Auditing, in versions prior to Excel 2002).



Excel 2002 includes background formula auditing. I cover this topic and other auditing features in Chapter 21.

Solver Add-in

For specialized linear and nonlinear problems, Excel's Solver add-in calculates solutions to what-if scenarios based on adjustable cells, constraint cells, and, optionally, cells that must be maximized or minimized.

Protection Options

Excel offers a number of different protection options. For example, you can protect formulas from being overwritten or modified, protect a workbook's structure, and protect your VBA code.

Protecting Formulas from Being Overwritten

In many cases, you may want to protect your formulas from being overwritten or modified. To do so, perform the following steps:

1. Select the cells that *may* be overwritten.
2. Select Format → Cells, and click the Protection tab of the Format Cells dialog box.
3. In the Protection tab, remove the checkmark from the Locked check box.
4. Click OK to close the Format Cells dialog box.
5. Select Tools → Protection → Protect Sheet to display the Protect Sheet dialog box, as shown in Figure 1-9. If you use a version prior to Excel 2002, this dialog box looks different.
6. In the Protect Sheet dialog box, specify a password if desired, and click OK.



By default, all cells are Locked. This has no effect, however, unless you have a protected worksheet.



Protection options in Excel 2002 are much more flexible. When you protect a worksheet, the Protect Sheet dialog box lets you choose which elements won't be protected. For example, you can allow users to sort data or use AutoFiltering on a protected sheet (tasks that weren't possible with earlier versions).

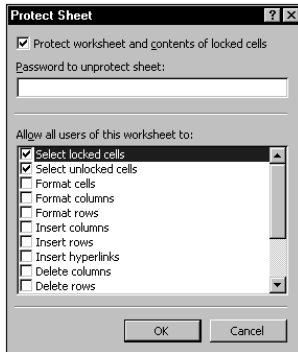


Figure 1-9: The Protect Sheet dialog box in Excel 2002

You can also hide your formulas so they won't appear in Excel's formula bar when the cell is activated. To do so, select the formula cells and make sure that the Hidden check box is checked in the Protection tab of the Format Cells dialog box.

Protecting a Workbook's Structure

When you protect a workbook's structure, you can't add or delete sheets. Use the Tools → Protection → Protect Workbook command to display the Protect Workbook dialog box, as shown in Figure 1-10. Make sure that you check the Structure check box. If you also check the Windows check box, the window can't be moved or resized.



Figure 1-10: The Protect Workbook dialog box



It's important to keep in mind that Excel is not really a secure application. The protection features, even when used with a password, are intended to prevent casual users from accessing various components of your workbook. Anyone who really wants to defeat your protection can probably do so by using readily available password-cracking utilities.

Summary

This chapter provided a general overview of the features available in Excel, and primarily focuses on newcomers to Excel. The next chapter gets into the meat of the book and provides an introduction to Excel formulas.

Chapter 2

Basic Facts about Formulas

IN THIS CHAPTER

- ◆ How to enter, edit, and paste names into formulas
- ◆ The various operators used in formulas
- ◆ How Excel calculates formulas
- ◆ Cell and range references used in formulas
- ◆ How to make an exact copy of a formula
- ◆ How to convert formulas to values
- ◆ How to prevent formulas from being viewed
- ◆ The types of formula errors
- ◆ Circular reference messages and correction techniques
- ◆ Excel's goal-seeking feature

THIS CHAPTER SERVES AS A BASIC INTRODUCTION to using formulas in Excel. Although I direct its focus on newcomers to Excel, even veteran Excel users may find some new information here.

Entering and Editing Formulas

This section describes the basic elements of a formula. It also explains various ways of entering and editing your formulas.

Formula Elements

A formula entered into a cell can consist of five element types:

- ◆ **Operators:** These include symbols such as + (for addition) and * (for multiplication)
- ◆ **Cell references:** These include named cells and ranges and can refer to cells in the current worksheet, cells in another worksheet in the same workbook, or even cells in a worksheet in another workbook.

- ◆ **Values or strings:** Examples include 7.5 or “*Year-End Results.*”
- ◆ **Worksheet functions and their arguments:** These include functions such as SUM or AVERAGE and their arguments.
- ◆ **Parentheses:** These control the order in which expressions within a formula are evaluated.

Entering a Formula

When you type an equal sign into an empty cell, Excel assumes that you are entering a formula (a formula always begins with an equal sign). Excel’s accommodating nature also permits you to begin your formula with a minus sign or a plus sign. However, Excel always inserts the leading equal sign after you enter the formula.

As a concession to former 1-2-3 users, Excel also enables you to use an “at” symbol (@) to begin a formula that starts with a function. For example, Excel accepts either of the following formulas:

```
=SUM(A1:A200)
```

```
@SUM(A1:A200)
```

However, after you enter the second formula, Excel replaces the at symbol with an equal sign. You can enter a formula into a cell in one of two ways: enter it manually, or enter it by pointing to cell references. I discuss each of these methods in the following sections.

ENTERING FORMULAS MANUALLY

Entering a formula manually involves, well, entering a formula manually. You simply activate a cell and type an equal sign (=) followed by the formula. As you type, the characters appear in the cell as well as in the formula bar. You can, of course, use all the normal editing keys when entering a formula. After you insert the formula, press Enter.



An exception to this is when you enter an array formula. When you enter an array formula, press Ctrl+Shift+Enter rather than just Enter. I discuss array formulas in Part IV.

After you press Enter, the cell displays the result of the formula. The formula, itself, appears in the formula bar when the cell is activated.

ENTERING FORMULAS BY POINTING

The other method of entering a formula still involves some manual typing, but you can simply point to the cell references instead of entering them manually. For example, to enter the formula $=A1+A2$ into cell A3, follow these steps:

1. Move the cell pointer to cell A3.
2. Type an equal sign (=) to begin the formula. Notice that Excel displays *Enter* in the left side of the status bar.
3. Press the up arrow twice. As you press this key, notice that Excel displays a faint moving border around the cell and that the cell reference (A1) appears in cell A3 and in the formula bar. Also notice that Excel displays *Point* in the status bar.

If you prefer, you can use your mouse and click cell A1.

4. Type a plus sign (+). The faint border disappears and *Enter* reappears in the status bar.
5. Press the up arrow one more time. A2 adds to the formula.

If you prefer, you can use your mouse and click cell A2.

6. Press Enter to end the formula. As with entering the formula manually, the cell displays the result of the formula, and the formula appears in the formula bar when the cell is activated.

If you prefer, you can use your mouse and click the check mark icon next to the formula bar.

Pointing to cell addresses rather than entering them manually is usually less tedious, and almost always more accurate.



When you create a formula that refers to other cells, the cell that contains the formula has the same number format as the first cell to which it refers. The only exception: if the first cell reference is formatted as a percentage.

Excel 97 and Excel 2000 include the Formula Palette feature that you can use when entering or editing formulas (see Figure 2-1). To access the Formula Palette, click on the Edit Formula button in the edit line (it has an image of an equal sign). The Formula Palette enables you to enter formulas manually or use the pointing techniques described previously.



The Formula Palette was removed from Excel 2002.

Pasting Names

As I discuss in Chapter 3, you can assign a name to a cell or range. If your formula uses named cells or ranges, you can type the name in place of the address or choose the name from a list and have Excel insert the name for you automatically.

To insert a name into a formula, select the Insert → Name → Paste command (or Press F3) to display the Paste Name dialog box. Excel displays its Paste Name dialog box with all the names listed, as shown in Figure 2-1. Select the name and click OK. Or, you can double-click the name, which inserts the name into the formula and closes the dialog box.

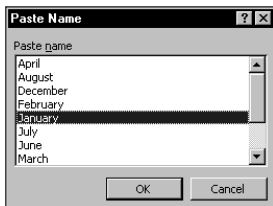


Figure 2-1: The Paste Name dialog box enables you to insert a name while entering a formula.

Spaces and Line Breaks

Normally, you enter a formula without using any spaces. However, you can use spaces (and even line breaks) within your formulas. Doing so has no effect on the formula's result, but may make the formula easier to read. To enter a line break in a formula, press Alt+Enter. Figure 2-2 shows a formula that contains spaces and line breaks.

Formula Limits

A formula can consist of up to 1,024 characters. If you need to create a formula that exceeds this limit, you must break the formula up into multiple formulas. You also can opt to create a custom function (using VBA).

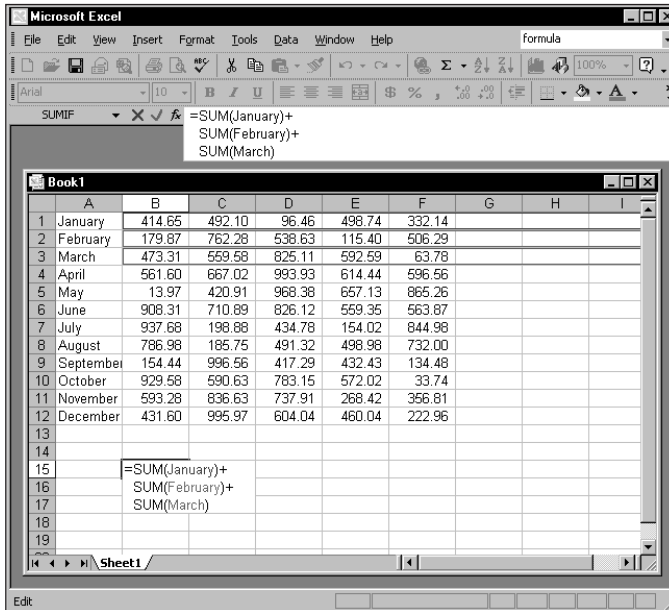


Figure 2-2: This formula contains spaces and line breaks.



Part IV focuses on creating custom functions.

Sample Formulas

If you follow the above instructions for entering formulas, you can create a variety of formulas. This section provides a look at some sample formulas.

- ◆ The following formula multiplies 150 times .01, and returns 1.5. This formula uses only literal values, so it doesn't prove very useful (you can simply enter the value 1.5 instead of the formula).

```
=150*.01
```

- ◆ This formula adds the values in cells A1 and A2:

```
=A1+A2
```

- ◆ The next formula subtracts the value in the cell named *Expenses* from the value in the cell named *Income*.

```
=Income-Expenses
```

- ◆ The following formula uses the SUM function to add the values in the range A1:A12.

=SUM(A1:A12)

- ◆ The next formula compares cell A1 with cell C12 by using the = operator. If the values in the two cells are identical, the formula returns TRUE; otherwise it returns FALSE.

=A1=C12

- ◆ This final formula subtracts the value in cell B3 from the value in cell B2 and then multiplies the result by the value in cell B4:

=(B2-B3)*B4

Editing Formulas

If you make changes to your worksheet, you may need to edit formulas. Or, the formula may return one of the error values described later in this chapter, and you need to edit the formula to correct the error. You can edit your formulas just as you edit any other cell.

There are several ways to get into cell edit mode:

1. Double-click the cell. This enables you to edit the cell contents directly in the cell. This technique works only if the Edit directly in cell option is in effect. You can change this option in the Edit tab of the Options dialog box.
2. Press F2. This enables you to edit the cell contents directly in the cell. If the Edit directly in cell option is not turned on, the editing will occur in the formula bar.
3. Select the formula cell that you want to edit and then click in the formula bar. This enables you to edit the cell contents in the formula bar.
4. Click the Edit Formula button (it has an equal sign icon) in the edit line to access the Formula Palette.



Excel 2002 does not have an Edit Formula button.

Using the Formula Bar as a Calculator

If you simply need to perform a calculation, you can use the formula bar as a calculator. For example, enter the following formula into any cell:

```
=(145*1.05)/12
```

Because this formula always returns the same result, you might prefer to store the formula's result rather than the formula. To do so, press F2 to edit the cell. Then press F9 followed by Enter. Excel stores the formula's result (12.6875), rather than the formula. This technique also works if the formula uses cell references.

You'll find that this technique is most useful when you use worksheet functions. For example, to enter the square root of 221 into a cell, enter =SQRT(221), press F9, and press Enter. Excel enters the result: 14.8660687473185. You also can use this technique to evaluate just part of a formula. Consider this formula:

```
=(145*1.05)/A1
```

If you want to convert just the expression within the parentheses to a value, get into cell edit mode and select the part that you want to evaluate. In this example, select 145*1.05. Then, press F9 followed by Enter. Excel converts the formula to the following:

```
=(152.25)/A1
```

When you edit a formula, you can select multiple characters by dragging the mouse over them or by holding down Shift while you use the arrow keys. You can also press Home or End to select from the cursor position to the beginning or end of the formula. If you use Ctrl+Shift, pressing the arrow keys allows you to select "words" within the formula.



Suppose you have a lengthy formula that contains an error, and Excel won't let you enter it because of the error. In this case, you can convert the formula to text and tackle it again later. To convert a formula to text, just remove the initial equal sign (=). To try the formula again, insert the initial equal sign to convert the cell contents back to a formula.

Using Operators in Formulas

As previously discussed, an operator is the basic element of a formula. An *operator* is a symbol that represents an operation. Excel supports the following operators:

+	Addition
-	Subtraction
/	Division
*	Multiplication
%	Percent
&	Text concatenation
^	Exponentiation
=	Logical comparison (equal to)
>	Logical comparison (greater than)
<	Logical comparison (less than)
>=	Logical comparison (greater than or equal to)
<=	Logical comparison (less than or equal to)
<>	Logical comparison (not equal to)

You can, of course, use as many operators as you need. Formulas can prove quite complex.

Reference Operators

Excel supports another class of operators known as *reference operators*. Reference operators, described in the following list, work with cell references.

:	(colon)	Range operator. Produces one reference to all the cells between two references.
,	(comma)	Union operator. This combines multiple cell or range references into one reference.
	(single space)	Intersection operator. This produces one reference to cells common to two references.

Sample Formulas That Use Operators

These examples of formulas use various operators:

- ◆ The following formula joins (*concatenates*) the two literal text strings to produce a new text string: *Part-23A*:
`= "Part- " & "23A"`
- ◆ The next formula concatenates the contents of cell A1 with cell A2:
`=A1&A2`

Usually, concatenation is used with text, but concatenation works with values as well. For example, if cell A1 contains 123 and cell A2 contains 456, the preceding formula would return the value 123456. Note that, technically, the result is a text string. However, this text string functions as a numeric value.

- ◆ The following formula uses the exponentiation operator to raise 6 to the third power, to produce a result of 216.

```
=6^3
```

- ◆ A more useful form of the above formula uses a cell reference instead of the literal value. Note this example that raises the value in cell A1 to the third power:

```
=A1^3
```

- ◆ This formula returns the cube root of 216 (which is 6):

```
=216^(1/3)
```

- ◆ The next formula returns TRUE if the value in cell A1 is less than the value in cell A2. Otherwise, it returns FALSE.

```
=A1<A2
```

Logical comparison operators also work with text. If A1 contains *Alpha* and A2 contains *Gamma*, the formula returns TRUE because Alpha comes before Gamma in alphabetical order.

- ◆ The following formula returns TRUE if the value in cell A1 is less than or equal to the value in cell A2. Otherwise, it returns FALSE.

```
=A1<=A2
```

- ◆ The next formula returns TRUE if the value in cell A1 does not equal the value in cell A2. Otherwise, it returns FALSE.

```
=A1<>A2
```

- ◆ Unlike some other spreadsheets (such as 1-2-3), Excel doesn't have logical AND or OR operators. Rather, you use functions to specify these types of logical operators. For example, this formula returns TRUE if cell A1 contains either 100 or 1000:

```
=OR(A1=100,A1=1000)
```

This last formula returns TRUE only if both cell A1 and cell A2 contain values less than 100:

```
=AND(A1<100,A2<100)
```

Operator Precedence

You can (and should) use parentheses in your formulas to control the order in which the calculations occur. As an example, consider the following formula that uses references to named cells.

```
=Income-Expenses*TaxRate
```

The goal is to subtract expenses from income and then multiply the result by the tax rate. If you enter the above formula, you discover that Excel computes the wrong answer. Rather, the formula multiplies expenses by the tax rate and then subtracts the result from the income. The correct way to write this formula is:

```
=(Income-Expenses)*TaxRate
```

To understand how this works, you need to be familiar with a concept called *operator precedence*—the set of rules that Excel uses to perform its calculations. Table 2-1 lists Excel's operator precedence. Operations with a lower precedence number are performed before operations with a higher precedence number.

Use parentheses to override Excel's built-in order of precedence. Returning to the previous example, the formula without parentheses is evaluated using Excel's standard operator precedence. Because multiplication has a higher precedence, the *Expense* cell multiplies by the *TaxRate* cell. Then, this result is subtracted from *Income*—producing an incorrect calculation.

The correct formula uses parentheses to control the order of operations. Expressions within parentheses always get evaluated first. In this case, *Expenses* is subtracted from *Income*, and the result multiplies by *TaxRate*.

TABLE 2-1 OPERATOR PRECEDENCE IN EXCEL FORMULAS

Symbol	Operator	Precedence
-	Negation	1
%	Percent	2
^	Exponentiation	3
* and /	Multiplication and division	4
+ and -	Addition and subtraction	5
&	Text concatenation	6
=, <, >, <=, >=, and <>	Comparison	7

Nested Parentheses

You can also *nest* parentheses in formulas. Nesting means putting parentheses inside of parentheses. If you do so, Excel evaluates the most deeply nested expressions first and works its way out. The following example of a formula uses nested parentheses.

```
=((B2*C2)+(B3*C3)+(B4*C4))*B6
```

This formula has four sets of parentheses. Three sets are nested inside the fourth set. Excel evaluates each nested set of parentheses and then sums the three results. This sum is then multiplied by the value in B6.

It's a good idea to make liberal use of parentheses in your formulas, even when they aren't necessary. Using parentheses clarifies the order of operations and makes the formula easier to read. For example, if you want to add 1 to the product of two cells, the following formula performs will do the job:

```
=A1*A2+1
```

You may find it much clearer, however, to use the following formula (with superfluous parentheses):

```
=(A1*A2)+1
```

Every left parenthesis, of course, must have a matching right parenthesis. If you have many levels of nested parentheses, you might find it difficult to keep them straight. If the parentheses don't match, Excel pops up a message telling you and won't permit you to enter the formula.



Fortunately, Excel lends a hand in helping you match parentheses. When you enter or edit a formula that has parentheses, pay attention to the text. When the cursor moves over a parenthesis, Excel momentarily displays the parenthesis and its closing parenthesis in bold. This lasts for less than a second, so watch carefully.

In some cases, if your formula contains mismatched parentheses, Excel may propose a correction to your formula (Excel 97 introduced this Formula AutoCorrect feature). Figure 2-3 shows an example of Excel's AutoCorrect feature in action.

Don't Hard-Code Values

When you create a formula, think twice before using a literal value in the formula. For example, if your formula calculates 7.5 percent sales tax, you may be tempted to enter a formula such as:

```
=A1*.075
```

A better approach is to insert the sales tax rate into a cell and use the cell reference in place of the literal value. This makes it easier to modify and maintain your worksheet. For example, if the sales tax rate changes to 7.75 percent, you need to modify every formula that uses the old value. If the tax rate is stored in a cell, you simply change one cell and all the formulas automatically get updated.



It is tempting to simply accept the correction proposed in the dialog box, but be careful. In many cases, the proposed formula, although syntactically correct, isn't the formula that you want. In Figure 2-3, I omitted the closing parentheses after January. Excel proposed this correction:

```
=SUM(January/SUM(Total))
```

In fact, the correct formula is:

```
=SUM(January)/SUM(Total)
```

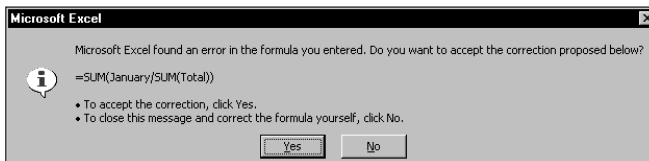


Figure 2-3: Excel's Formula AutoCorrect feature often suggests a correction to an erroneous formula.

Calculating Formulas

You've probably noticed that the formulas in your worksheet get calculated immediately. If you change any cells that the formula uses, the formula displays a new result with no effort on your part. This occurs when Excel's Calculation mode is set to Automatic. In this mode (the default mode), Excel follows certain rules when calculating your worksheet:

- ◆ When you make a change (enter or edit data or formulas, for example), Excel calculates immediately those formulas that depend on new or edited data.
- ◆ If working on a lengthy calculation, Excel temporarily suspends calculation when you need to perform other worksheet tasks; it resumes when you finish.
- ◆ Formulas are evaluated in a natural sequence. For instance, if a formula in cell D12 depends on the result of a formula in cell D11, cell D11 is calculated before D12.

Sometimes, however, you may want to control when Excel calculates formulas. For example, if you create a worksheet with thousands of complex formulas, you'll find that things can slow to a snail's pace while Excel does its thing. In this case, you can set Excel's calculation mode to Manual. Do this in the Calculation panel of the Options dialog box. (Select Tools → Options to display this dialog box.)

When you work in Manual calculation mode, Excel displays *Calculate* in the status bar when you have any uncalculated formulas. You can use the following shortcut keys to recalculate the formulas:

- ◆ **F9**: Calculates the formulas in all open workbooks.
- ◆ **Shift+F9**: Calculates only the formulas in the active worksheet. It does not calculate other worksheets in the same workbook.
- ◆ **Ctrl+Alt+F9**: Forces a complete recalculation of all open workbooks. Use it if Excel (for some reason) doesn't seem to return correct calculations.
- ◆ **Ctrl+Shift+Alt+F9**: Rechecks all of the dependent formulas, and then forces a recalculation of all open workbooks.



The Ctrl+Shift+Alt+F9 key sequence works only in Excel 2002.



Excel's Calculation mode isn't specific to a particular worksheet. When you change Excel's Calculation mode, it affects all open workbooks — not just the active workbook. Also, the initial Calculation mode is set by the Calculation mode saved with the first workbook you open.

Cell and Range References

Most formulas reference one or more cells by using the cell or range address (or name if it has one). Cell references come in four styles; the dollar sign differentiates them:

- ◆ **Relative:** The reference is fully relative. When the formula is copied, the cell reference adjusts to its new location. Example: A1
- ◆ **Absolute:** The reference is fully absolute. When the formula is copied, the cell reference does not change. Example: \$A\$1
- ◆ **Row Absolute:** The reference is partially absolute. When the formula is copied, the column part adjusts, but the row part does not change. Example: A\$1
- ◆ **Column Absolute:** The reference is partially absolute. When the formula is copied, the row part adjusts, but the column part does not change. Example: \$A1

Creating an Absolute Reference

When you create a formula by pointing to cells, all cell and range references are relative. To change a reference to an absolute reference, you must do so manually by adding the dollar signs. Or, when you enter a cell or range address, you can use the F4 key to cycle among all possible reference modes.

If you think about it, you may realize that the only reason you would ever need to change a reference is if you plan to copy the formula. Figure 2-4 demonstrates this. Note the formula in cell C4:

```
=C$3*$B4
```

This formula calculates the area for various widths (listed in column B) and lengths (listed in Row 3). After you enter the formula, it can then be copied down and across. Because the formula uses absolute references to row 3 and column B, each copied formula produces the correct result. If the formula uses relative references, copying the formula causes the references to adjust and produce the wrong results.

A1 vs. R1C1 Notation

Normally, Excel uses what is referred to as A1 notation. Each cell address consists of a column letter and a row number. However, Excel also supports R1C1 notation. In this system, cell A1 is referred to as cell R1C1, cell A2 as R2C1, and so on.

To change to R1C1 notation, select Tools → Options to get the Options dialog box, click the General tab, and place a check mark next to R1C1 reference style. Now, notice that the column letters all change to numbers. And, all of the cell and range references in your formulas also adjust.

Look at the following examples of formulas using standard notation and R1C1 notation. The formula is assumed to be in cell B1 (also known as R1C2).

Standard	R1C1
=A1+1	=RC[-1]+1
=\$A\$1+1	=R1C1+1
=\$A1+1	=RC1+1
=A\$1+1	=R1C[-1]+1
=SUM(A1:A10)	=SUM(RC[-1]:R[9]C[-1])
=SUM(\$A\$1:\$A\$10)	=SUM(R1C1:R10C1)

If you find R1C1 notation confusing, you're not alone. R1C1 notation isn't too bad when you're dealing with absolute references. But when relative references are involved, the brackets can drive you nuts.

The numbers in brackets refer to the relative position of the references. For example, R[-5]C[-3] specifies the cell that appears five rows above and three columns to the left. Conversely, R[5]C[3] references the cell that appears five rows below and three columns to the right. If you omit the brackets, it specifies the same row or column. For example, R[5]C refers to the cell five rows below in the same column.

Although you probably won't use R1C1 notation as your standard system, it *does* have at least one good use. R1C1 notation makes it very easy to spot an erroneous formula. When you copy a formula, every copied formula is exactly the same in R1C1 notation. This remains true regardless of the types of cell references you use (relative, absolute, or mixed). Therefore, you can switch to R1C1 notation and check your copied formulas. If one looks different from its surrounding formulas, it's probably incorrect.

Continued

A1 vs. R1C1 Notation *(Continued)*

If you're using Excel 2002, however, you can take advantage of the new background formula auditing feature. This feature can flag potentially incorrect formulas. I discuss this feature in Chapter 21.

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3			1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
4	1.0	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50	5.00	
5	1.5	1.50	2.25	3.00	3.75	4.50	5.25	6.00	6.75	7.50	
6	2.0	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	
7	2.5	2.50	3.75	5.00	6.25	7.50	8.75	10.00	11.25	12.50	
8	3.0	3.00	4.50	6.00	7.50	9.00	10.50	12.00	13.50	15.00	
9	3.5	3.50	5.25	7.00	8.75	10.50	12.25	14.00	15.75	17.50	
10	4.0	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00	20.00	
11	4.5	4.50	6.75	9.00	11.25	13.50	15.75	18.00	20.25	22.50	
12	5.0	5.00	7.50	10.00	12.50	15.00	17.50	20.00	22.50	25.00	
13											

Figure 2-4: An example of using non-relative references in a formula

Referencing Other Sheets or Workbooks

A formula can use references to cells and ranges that are in a different worksheet. To refer to a cell in a different worksheet, precede the cell reference with the sheet name followed by an exclamation point. Note this example of a formula that uses a cell reference in a different worksheet (Sheet2):

```
=Sheet2!A1+1
```

You can also create link formulas that refer to a cell in a different workbook. To do so, precede the cell reference with the workbook name (in square brackets), the worksheet name, and an exclamation point like this:

```
= [Budget.xls]Sheet1!A1+1
```

If the workbook name in the reference includes one or more spaces, you must enclose it (and the sheet name) in single quotation marks. For example:

```
= '[Budget Analysis.xls]Sheet1'!A1+A1
```

If the linked workbook is closed, you must add the complete path to the workbook reference. For example:

```
= 'C:\MSOffice\Excel\[Budget Analysis.xls]Sheet1'!A1+A1
```

Although you can enter link formulas directly, you also can create the reference by using normal pointing methods discussed earlier. To do so, make sure you have an open source file. Normally, you can create a formula by pointing to results in relative cell references. But, when you create a reference to a workbook by pointing, Excel creates *absolute* cell references (if you plan to copy the formula to other cells, you must edit the formula to make the references relative).



Working with links can be tricky and may cause some unexpected problems. For example, if you use the File → Save As command to make a backup copy of the source workbook, you automatically change the link formulas to refer to the new file (not usually what you want). You also can mess up your links by renaming the source workbook file.

Making an Exact Copy of a Formula

When you copy a formula, Excel adjusts the formula's cell references when you paste it to a different location. This is usually exactly what you want. Sometimes, however, you may want to make an exact copy of the formula. You can do this by converting the cell references to absolute values, as discussed earlier – but this isn't always desirable.

A better approach is to select the formula while in edit mode and then copy it to the Clipboard as text. There are several ways to do this. Here I present a step-by-step example of how to make an exact copy of the formula in A1 and copy it to A2:

Using Links to Recover Data in a Corrupt File

At some point, you may find one of your Excel workbooks damaged or corrupt. If you cannot load a corrupt workbook, you can write a link formula to recover all or part of the data (but not the formulas). You can do this because you do not need to have the source file in a link formula open. If your corrupt file is named `Badfile.xls`, for example, open a blank workbook and enter the following formula into cell A1 to attempt to recover the data from Sheet1:

```
=[Badfile.xls]Sheet1!A1
```

Copy this formula down and to the right to recover as much information as you can. As a better approach, however, you can maintain a backup of your important files.

If you use Excel 2002, corrupt workbooks are less of a problem because this version can often repair such files.

1. Double-click cell A1 to activate edit mode (or, press F2).
2. Press End, followed by Shift+Home to select all of the formula text. Or, you can drag the mouse to select the entire formula.
3. Click the Copy button on the Standard toolbar (or, press Ctrl+C). This copies the selected text to the Clipboard.
4. Press Enter to end edit mode.
5. Activate cell A2.
6. Click the Paste button on the Standard toolbar (or, press Ctrl+V). This operation pastes an exact copy of the formula text into cell A2.

You also can use this technique to copy just *part* of a formula to use in another formula. Just select the part of the formula that you want to copy by dragging the mouse or by using the Shift+arrow keys. Then use any of the available techniques to copy the selection to the Clipboard. You can then paste the text to another cell.

Formulas (or parts of formulas) copied in this manner won't have their cell references adjusted when you paste them to a new cell. This is because you copy the formulas as text, not as actual formulas.

Another technique for making an exact copy of a formula is to edit the formula and remove its initial equal sign. This converts the formula to text. Then, copy the "non-formula" to a new location. Finally, edit both the original and the copied formula by inserting the initial equal sign.

Converting Formulas to Values

If you have a range of formulas that always produce the same result (i.e., dead formulas), you may want to convert them to values. You can use the Edit→Paste Special command to do this.

Suppose that range A1:A10 contains formulas that calculate a result and that never changes. To convert these formulas to values:

1. Select A1:A10.
2. Click the Copy button on the Standard toolbar (or, press Ctrl+C).
3. Select the Edit→Paste Special command. Excel displays its Paste Special dialog box.
4. Select the Values option button and then click OK.
5. Press Enter or Esc to cancel paste mode.



If you're using Excel 2002, you can take advantage of a Smart Tag. In Step 3 in the preceding list, select Edit → Paste (or press Ctrl+V). A Smart Tag will appear at the lower right corner of the range. Click the Smart Tag and choose Values Only (see Figure 2-5).

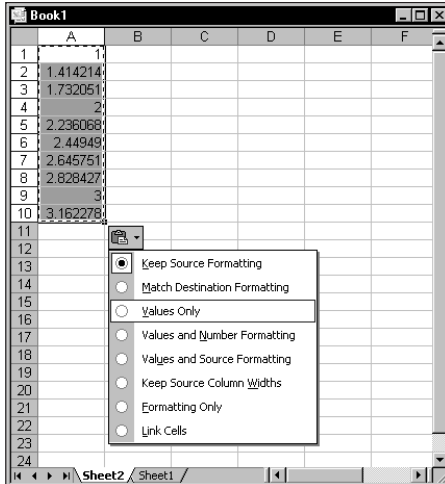


Figure 2-5: in Excel 2002, a Smart Tag appears after pasting data.

This technique is very useful when you use formulas as a means to convert cells. For example, assume you have a list of names (in uppercase) in column A. You want to convert these names to proper case. In order to do so, you need to create formulas in a separate column; then convert the formulas to values and replace the original values in column A. The following steps illustrate how to do this.

1. Insert a new column after column A.
2. Insert the following formula into cell B1:


```
=PROPER(A1)
```
3. Copy the formula down column B, to accommodate the number of entries in column A. Column B then displays the values in column A, but in proper case.
4. Select all the names in column B.
5. Click the Copy button on the Standard toolbar.
6. Select cell A1.

When to Use AutoFill Rather Than Formulas

Excel's AutoFill feature provides a quick way to copy a cell to adjacent cells. AutoFill also has some other uses that may even substitute for formulas in some cases. I'm surprised to find that many experienced Excel users don't take advantage of the AutoFill feature, which can save a lot of time.

For example, if you need a list of values from 1 to 100 to appear in A1:A100, you can do it with formulas. You enter 1 in cell A1, the formula `=A1+1` into cell A2 and then copy the formula to the 98 cells below.

You also can use AutoFill to create the series for you without using a formula. To do so, enter 1 into cell A1 and 2 into cell A2. Select A1:A2 and drag the fill handle down to cell A100. (The fill handle is the small square at the lower right corner of the active cell.) When you use AutoFill in this manner, Excel analyzes the selected cells and uses this information to complete the series. If cell A1 contains 1 and cell A2 contains 3, Excel recognizes this pattern and fills in 5, 7, 9, and so on. This also works with decreasing series (10, 9, 8, and so on) and dates. If there is no discernible pattern in the selected cells, Excel performs a linear regression and fills in values on the calculated trend line.

Excel also recognizes common series names such as months and days of the week. If you enter Monday into a cell and then drag its fill handle, Excel fills in the successive days of the week. You also can create custom AutoFill lists using the Custom Lists panel of the Options dialog box. Finally, if you drag the fill handle with the right mouse button, Excel displays a shortcut menu to enable you to select an AutoFill option.

7. Select the **E**dit → Paste Special command. Excel displays its Paste Special dialog box.
8. Select the Values option button and then click OK.
9. Press Enter or Esc to cancel paste mode.
10. Delete column B.

Hiding Formulas

In some cases, you may not want others to see your formulas. For example, you may have a special formula you developed that performs a calculation proprietary to your company. You can use the Format Cells dialog box to hide the formulas contained in these cells.

To prevent one or more formulas from being viewed:

1. Select the formula or formulas.
2. Choose Format → Cells. In the Format Cells dialog box, click the Protection tab.
3. Place a check mark next to the Hidden check box, as shown in Figure 2-6.
4. Use the Tools → Protection → Protect Sheet command to protect the worksheet. To prevent others from unprotecting the sheet, make sure you specify a password in the Protect Sheet dialog box.

By default, all cells are “locked.” Protecting a sheet prevents any locked cells from being changed. Therefore, you should unlock any cells that require user input before protecting your sheet.



Be aware that several password-cracking utilities are available. Therefore, this technique of hiding your formulas does not ensure that no one can view your formulas.

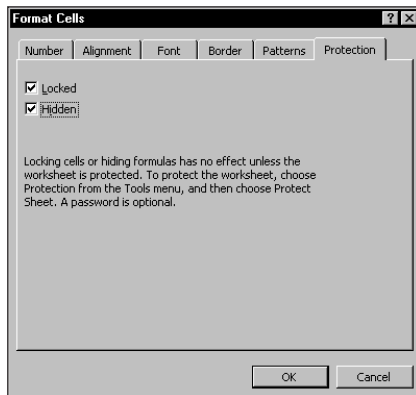


Figure 2-6: Use the Format Cells dialog box to change the Hidden status of a cell.

Errors in Formulas

It's not uncommon to enter a formula only to find that the formula returns an error. Table 2-2 lists the types of error values that may appear in a cell that has a formula.

Formulas may return an error value if a cell that they refer to has an error value. This is known as the ripple effect: A single error value can make its way to lots of other cells that contain formulas that depend on that cell.

TABLE 2-2 EXCEL ERROR VALUES

Error Value	Explanation
#DIV/0!	The formula attempts to divide by zero (an operation not allowed on this planet). This also occurs when the formula attempts to divide by an empty cell.
#NAME?	The formula uses a name that Excel doesn't recognize. This can happen if you delete a name used in the formula or if you misspell a function.
#N/A	The formula refers (directly or indirectly) to a cell that uses the NA function to signal unavailable data. This error also occurs if a lookup function does not find a match.
#NULL!	The formula uses an intersection of two ranges that don't intersect. (I describe this concept later in the chapter.)
#NUM!	A problem occurs with a value; for example, you specify a negative number where a positive number is expected.
#REF!	The formula refers to an invalid cell. This happens if the cell has been deleted from the worksheet.
#VALUE!	The formula includes an argument or operand of the wrong type. An <i>operand</i> refers to a value or cell reference that a formula uses to calculate a result.



If the entire cell fills with hash marks (#####), this usually means that the column isn't wide enough to display the value. You can either widen the column or change the number format of the cell. The cell will also fill with hash marks if it contains a formula that returns an invalid date or time.



In Excel 2002, formulas that return an error display a Smart Icon. You can click this Smart Icon to get more information about the error or to trace the calculation steps that led to the error. Refer to Chapter 21 for more information about this feature.

Dealing with Circular References

When you enter formulas, you may occasionally see a message from Excel like the one shown in Figure 2-7. This indicates that the formula you just entered will result in a *circular reference*.

A circular reference occurs when a formula refers to its own value, either directly or indirectly. For example, if you enter `=A1+A2+A3` into cell A3, this produces a circular reference because the formula in cell A3 refers to cell A3. Every time the formula in A3 is calculated, it must be calculated again because A3 has changed. The calculation would go on forever. In other words, the answer never gets resolved.

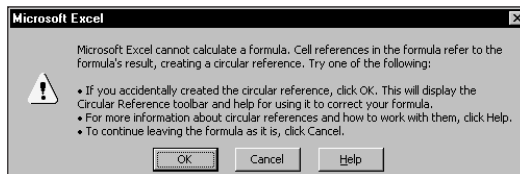


Figure 2-7: Excel's way of telling you that your formula contains a circular reference

When you enter a formula that contains a circular reference, Excel displays a dialog box with three options:

- ◆ Click OK to attempt to locate the circular reference.
- ◆ Click Cancel to enter the formula as is.
- ◆ Click Help to read more about circular references in the online help.

Normally, you'll want to correct any circular references, so you should choose OK. When you do so, Excel displays its Circular Reference toolbar (see Figure 2-8). On the Circular Reference toolbar, click the first cell in the Navigate Circular Reference drop-down list box, and then examine the cell's formula. If you cannot determine whether the cell is the cause of the circular reference, click the next cell in the Navigate Circular Reference drop-down list box. Continue to review the formulas until the status bar no longer displays *Circular*.

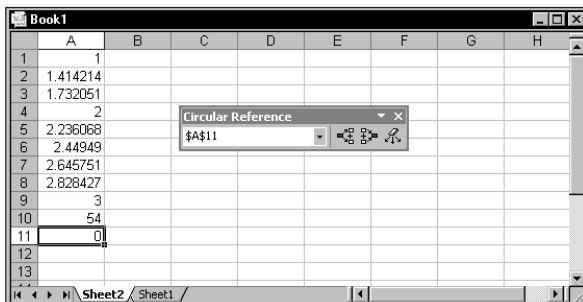


Figure 2-8: The Circular Reference toolbar



There are a few situations in which you may want to use a circular reference intentionally. Refer to Chapter 16 for some examples.

If you ignore the circular reference message (by clicking Cancel), Excel enables you to enter the formula and displays a message in the status bar reminding you that a circular reference exists. In this case, the message reads *Circular: A3*. If you activate a different worksheet or workbook, the message simply displays *Circular* (without the cell reference).



Excel doesn't warn you about a circular reference if you have the *Iteration* setting turned on. You can check this in the Options dialog box (in the Calculation panel). If *Iteration* is on, Excel performs the circular calculation the number of times specified in the *Maximum iterations* field (or until the value changes by less than .001 — or whatever other value appears in the *Maximum change* field). You should, however, keep the *Iteration* setting off so that you'll be warned of circular references. Generally, a circular reference indicates an error that you must correct.

Usually, the cause of a circular reference is quite obvious and is, therefore, easy to identify and correct. Sometimes, however, you will encounter indirect circular references. In other words, a formula may refer to a formula that refers to a formula that refers back to the original formula. In some cases, it may require you to do a bit of detective work to reach the problem.

Goal Seeking

Many spreadsheets contain formulas that enable you to ask questions, such as, “What would be the total profit if sales increase by 20 percent?” If you set up your worksheet properly, you can change the value in one cell to see what happens to the profit cell.

Goal seeking serves as a useful feature that works in conjunction with your formulas. If you know what a formula result *should* be, Excel can tell you which values of one or more input cells you need to produce that result. In other words, you can ask a question such as, “What sales increase is needed to produce a profit of \$1.2 million?”

Single-cell goal seeking (also known as *backsolving*) represents a rather simple concept. Excel determines what value in an input cell produces a desired result in a formula cell. You can best understand how this works by walking through an example.

A Goal-Seeking Example

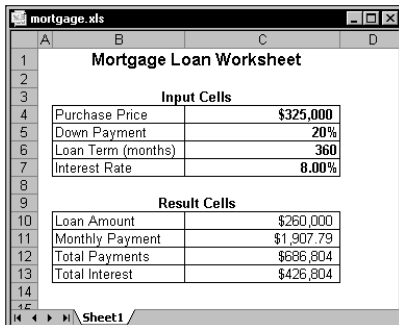
Figure 2-9 shows a mortgage loan worksheet that has four input cells (C4:C7) and four formula cells (C10:C13). The formulas calculate various values using the input cell. The formulas are:

C10: $=(1-C5)*C4$

C11: $=PMT(C7/12,C6,-C10)$

C12: $=C11*C6$

C13: $=C12-C10$



Mortgage Loan Worksheet	
Input Cells	
Purchase Price	\$325,000
Down Payment	20%
Loan Term (months)	360
Interest Rate	8.00%
Result Cells	
Loan Amount	\$260,000
Monthly Payment	\$1,907.79
Total Payments	\$696,804
Total Interest	\$426,804

Figure 2-9: This worksheet presents a good demonstration of goal seeking.

Imagine that you're in the market for a new home and you know that you can afford \$1,200 per month in mortgage payments. You also know that a lender can issue a fixed-rate mortgage loan for 8.00 percent, based on an 80 percent loan-to-value (a 20 percent down payment). The question is, "What is the maximum purchase price you can handle?" In other words, what value in cell C4 causes the formula in cell C11 to result in \$1,200? You can plug values into cell C4 until C11 displays \$1,200. A more efficient approach lets Excel determine the answer.

To answer this question, select Tools → Goal Seek. Excel responds with the Goal Seek dialog box shown in Figure 2-10. Completing this dialog box resembles forming a sentence. Set cell C11 to 1200 by changing cell C4. Enter this information in the dialog box by either typing the cell references or by pointing with the mouse. Click OK to begin the goal-seeking process.

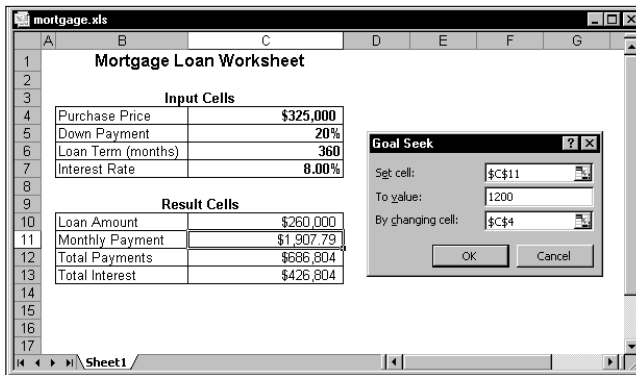


Figure 2-10: The Goal Seek dialog box

Almost immediately, Excel announces that it has found the solution and displays the Goal Seek Status box. This box tells you the target value and what Excel came up with. In this case, Excel found an exact value. The worksheet now displays the found value in cell C4 (\$204,425). As a result of this value, the monthly payment amount is \$1,200. Now, you have two options:

- ◆ Click OK to replace the original value with the found value.
- ◆ Click Cancel to restore your worksheet to its original form before you chose Tools → Goal Seek.

More about Goal Seeking

If you think about it, you may realize that Excel can't always find a value that produces the result you're looking for—sometimes a solution doesn't exist. In such a case, the Goal Seek Status box informs you of that fact (see Figure 2-11). Other

times, however, Excel may report that it can't find a solution, even though you believe one exists. In this case, you can adjust the current value of the changing cell to a value closer to the solution, and then reissue the command. If that fails, double-check your logic, and make sure that the formula cell does indeed depend on the specified changing cell.

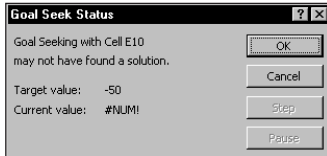


Figure 2-11: The Goal Seek Status box tells you if Excel can't find a solution to your goal-seeking problem.

Like all computer programs, Excel has limited precision. To demonstrate this, enter $=A1^2$ into cell A2. Then, select **T**ools → **G**oal Seek to find the value in cell A1 that causes the formula to return 16. Excel returns a value of 4.00002269 – close to the square root of 16, but certainly not exact. You can adjust the precision in the Calculation panel of the Options dialog box (make the Maximum change value smaller).

In some cases, multiple values of the input cell produce the same desired result. For example, the formula $=A1^2$ returns 16 if cell A1 contains either -4 or $+4$. If you use goal seeking when two solutions exist, Excel gives you the solution that has the same sign as the current value in the cell, or the solution that is nearest to the current value in the cell.

Perhaps the main limitation of the **T**ools → **G**oal Seek command is that it can find the value for only one input cell. For example, it can't tell you what purchase price *and* what down payment percent result in a particular monthly payment. If you want to change more than one variable at a time, use Solver.

Summary

This chapter provided an introduction to Excel formulas and covered the various elements that comprise a formula. The chapter also discussed related topics such as relative and absolute references, converting formulas to values, formula errors, and circular references.

The next chapter covers how to work with names in Excel.

Chapter 3

Working with Names

IN THIS CHAPTER

- ◆ An overview and the advantages of using names in Excel
- ◆ Various ways to create cell and range names
- ◆ How to create names that extend across multiple worksheets
- ◆ The difference between workbook- and worksheet-level names
- ◆ How to perform common operations with range and cell names
- ◆ How Excel maintains cell and range names
- ◆ Potential problems that may crop up when you use names
- ◆ The secret behind names and examples of named constants and named formulas
- ◆ Examples of advanced techniques that use names

MOST INTERMEDIATE AND ADVANCED Excel users are familiar with the concept of named cells or ranges. Naming cells and ranges is an excellent practice and offers several important advantages. As you'll see in this chapter, Excel supports other types of names – and the power of this concept may surprise you.

What's in a Name?

You can think of a *name* as an identifier for something in a workbook. This “something” can consist of a cell, a range, a chart, a shape, and so on. If you provide a name for a range, you can then use that name in your formulas. For example, suppose your worksheet contains daily sales information stored in the range B2:B200. Further, assume that cell C1 contains a sales commission rate. The following formula returns the sum of the sales, multiplied by the commission rate:

```
=SUM(B2:B200)*C1
```

This formula works fine, but its purpose is not at all clear. To help clarify the formula, you can define one descriptive name for the daily sales range and another descriptive name for cell C1. For example, assume that the range B2:B200 is named

DailySales and cell C1 is named *CommissionRate*. You can then rewrite the formula to use the names instead of the actual range addresses:

```
=SUM(DailySales)*CommissionRate
```

As you can see, using names instead of cell references makes the formula “self-documenting,” and much easier to understand.

Using named cells and ranges offers a number of advantages:

- ◆ Names make your formulas more understandable and easier to use, especially for people who didn’t create the worksheet. Obviously, a formula such as `=Income-Taxes` is more intuitive than `=D20-D40`.
- ◆ When entering formulas, a descriptive range name (such as *Total_Income*) is easier to remember than a cell address (such as AC21). And typing a name is less error-prone than entering a cell or range address.
- ◆ You can quickly move to areas of your worksheet either by using the Name box, located at the left side of the formula bar (click the arrow for a drop-down list of defined names) or by choosing Edit → Go To (or F5) and specifying the range name.
- ◆ When you select a named cell or range, its name appears in the Name box.
- ◆ You may find that creating formulas is easier if you use named cells. You can paste a cell or range name into a formula by using the Insert → Name → Paste command (or F3).
- ◆ Macros are easier to create and maintain when you use range names rather than cell addresses.

Methods for Creating Cell and Range Names

Excel provides several ways to create names for cells and ranges. I discuss these methods in this section, along with other relevant information that pertains to names.

Creating Names Using the Define Name Dialog Box

To create a name for a cell or range, start by selecting the cell or range that you want to name. Then select Insert → Name → Define (or press Ctrl+F3). Excel displays the Define Name dialog box, shown in Figure 3-1.

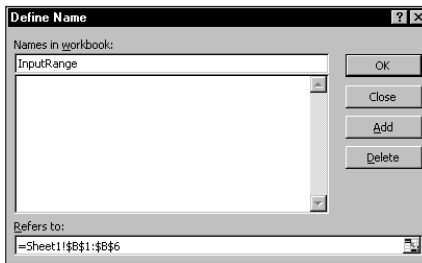


Figure 3-1: Use the Define Name dialog box to create names for cells or ranges.

Type a name in the field labeled Names in the workbook (or use the name that Excel proposes, if any). The selected cell or range address appears in the Refers to field. Verify that the address listed is correct and then click OK to add the name to your worksheet and close the dialog box. Or, click the Add button to continue adding names to your worksheet. If you do this, you must specify the Refers to range either by typing an address (make sure to begin with an equal sign) or by pointing to it in the worksheet.



A single cell or range can have any number of names. I can't think of a good reason to use more than one name, but Excel does permit it. If a cell or range has multiple names, the Name box always displays the first name when you select the cell or range.

A name can also refer to a noncontiguous range of cells. You can select a noncontiguous range by pressing the Ctrl key while you select various cells or ranges with the mouse.



If you try to edit the contents of the Refers to field manually, you'll find that this field is in "point" mode. You can't use keys such as End and Home to edit the field's contents. To switch from point mode to normal edit mode, press F2. Then you can use the standard editing keys when the Refers to field is activated.

Creating Names Using the Name Box

A faster way to create a name involves accessing the Name box. The Name box is the drop-down box to the left of the formula bar. Select the cell or range to name, and then click the Name box and type the name. Press Enter to create the name. If

Rules for Naming Names

Although Excel is quite flexible about the names that you can define, it does have some rules:

- ◆ Names can't contain any spaces. You might want to use an underscore or a period character to simulate a space (such as *Annual_Total* or *Annual.Total*).
- ◆ You can use any combination of letters and numbers, but the name must begin with a letter or underscore. A name can't begin with a number (such as *3rdQuarter*) or look like a cell reference (such as *Q3*).
- ◆ You cannot use symbols, except for underscores and periods. Although not documented, I've found that Excel also permits a backslash (\) and question mark (?) as long as they don't appear as the first character in a name.
- ◆ Names are limited to 255 characters. Trust me – you should not use a name anywhere near this length. In fact, doing so defeats the purpose of naming ranges.
- ◆ You can use single letters (except for R or C), but generally I do not recommend this because it also defeats the purpose of using meaningful names.
- ◆ Names are not case sensitive. The name *AnnualTotal* is the same as *annual-total*. Excel stores the name exactly as you type it when you define it, but it doesn't matter how you capitalize the name when you use it in a formula.

Excel also uses a few names internally for its own use. Although you can create names that override Excel's internal names, you should avoid doing so unless you know what you're doing. Generally, avoid using the following names: *Print_Area*, *Print_Titles*, *Consolidate_Area*, *Database*, *Criteria*, *Extract*, *FilterDatabase*, and *Sheet_Title*.

a name already exists, you can't use the Name box to change the range to which that name refers. Attempting to do so simply selects the original range. You must use the Define Name dialog box to change the reference for a name.



When you type a name in the Name box, you *must* press Enter to actually record the name. If you type a name and then click in the worksheet, Excel won't create the name.

The Name box serves double-duty by also providing a quick way to activate a named cell or range, as shown in Figure 3-2. To select a named cell or range, click the Name box and choose the name. This selects the named cell or range. Oddly, the

Name box does not have a keyboard shortcut. In other words, you can't access the Name box by using the keyboard; you must use a mouse. After you click the Name box, however, you can use the direction keys and Enter to choose a name.

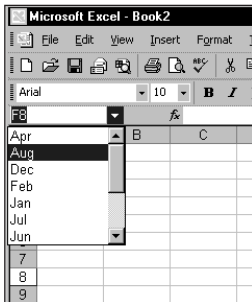


Figure 3-2: The Name box provides a quick way to activate a named cell or range.

Creating Names Automatically

You may have a worksheet containing text that you want to use for names of adjacent cells or ranges. Figure 3-3 shows an example of such a worksheet. In this case, you might want to use the text in column A to create names for the corresponding values in column B. Excel makes this very easy to do.

A screenshot of an Excel worksheet titled 'Book2'. The worksheet contains a table with the following data:

	A	B	C	D	E
1	Month	Sales			
2	Jan	25,984			
3	Feb	28,973			
4	Mar	21,983			
5	Apr	32,744			
6	May	31,982			
7	Jun	27,008			
8	Jul	31,982			
9	Aug	33,321			
10	Sep	27,440			
11	Oct	26,123			
12	Nov	29,831			
13	Dec	18,732			
14					

Figure 3-3: Excel makes it easy to create names by using text in adjacent cells.

To create names by using adjacent text, start by selecting the name text and the cells that you want to name (these can consist of individual cells or ranges of cells). The names must be adjacent to the cells that you're naming (a multiple selection is allowed). Then choose Insert → Name → Create (or Ctrl+Shift+F3). Excel displays the Create Names dialog box, shown in Figure 3-4.

The check marks in this dialog box are based on Excel's analysis of the selected range. For example, if Excel finds text in the first row of the selection, it proposes that you create names based on the top row. If Excel doesn't guess correctly, you can change the check boxes. Click OK and Excel creates the names. Note that when Excel creates names using text in cells, it does not include those text cells in the named range.

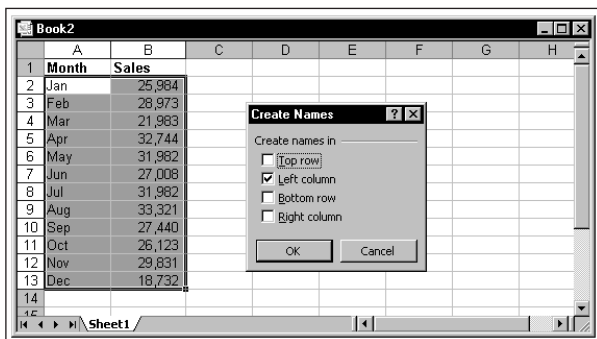


Figure 3-4: The Create Names dialog box

If the text in a cell would result in an invalid name, Excel modifies the name to make it valid. For example, if a cell contains the text *Net Income* (invalid for a name because it contains a space), Excel converts the space to an underscore character and creates the name *Net_Income*. If Excel encounters a value or a formula instead of text, however, it doesn't convert it to a valid name. It simply doesn't create a name.



Double-check the names that Excel creates. Sometimes, the Insert → Name → Create command works counter-intuitively. Figure 3-5 shows a small table of text and values. If you select the entire table, choose Insert → Name → Create, and accept Excel's suggestions (Top row and Left column options). You'll find that the name *Products* doesn't refer to A2:A6, as you may expect, but instead refers to B2:C6. If the upper-left cell of the selection contains text and you choose the Top row and Left column options, Excel uses that text for the name of the entire set of data — excluding the top row and left column. So, before you accept the names that Excel creates, take a minute to make sure that they refer to the correct ranges.

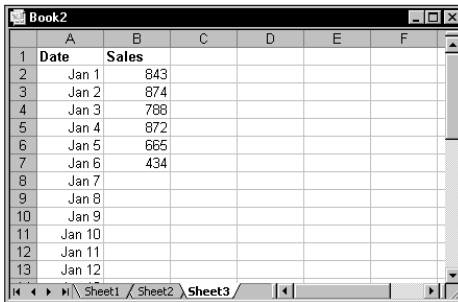


	A	B	C	D	E
1	Products	Quantity	Price		
2	Pencils	433	0.29		
3	Paper	109	3.89		
4	Erasers	41	0.79		
5	Pens	209	1.29		
6	Notepads	89	3.29		
7					
8					

Figure 3-5: Creating names from the data in this table may produce unexpected results.

Naming Entire Rows and Columns

Sometimes it makes sense to name an entire row or column. Often, a worksheet is used to store information that you enter over a period of time. The sheet in Figure 3-6 is an example of such a worksheet. If you create a name for the data in column B, you need to modify the name's reference each day you add new data. The solution is to name the entire column.



	A	B	C	D	E	F
1	Date	Sales				
2	Jan 1	843				
3	Jan 2	874				
4	Jan 3	788				
5	Jan 4	872				
6	Jan 5	665				
7	Jan 6	434				
8	Jan 7					
9	Jan 8					
10	Jan 9					
11	Jan 10					
12	Jan 11					
13	Jan 12					

Figure 3-6: This worksheet, which tracks daily sales, uses a named range that consists of an entire column.

For example, you might name column B *DailySales*. If this range were on Sheet3, its reference would appear like this:

```
=Sheet3!$B:$B
```

After defining the name, you can use it in a formula. The following formula, for example, returns the sum of all values in column B:

```
=SUM(DailySales)
```

Names Created by Excel

Excel creates some names on its own. For example, if you set a print area for a sheet, Excel creates the name *Print_Area*. If you set repeating rows or columns for printing, you also have a worksheet-level name called *Print_Titles*. When you execute a query that returns data to a worksheet, Excel assigns a name to the data that is returned. Also, many of the add-ins that ship with Excel create hidden names (see the “Hidden Names” sidebar).

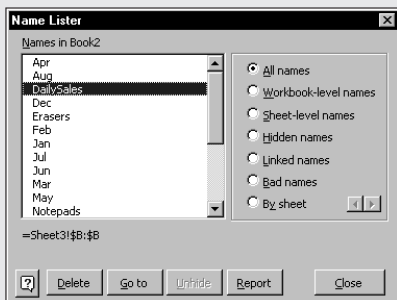
You can modify the reference for any of the names that Excel creates automatically, but make sure that you understand the consequences.

Hidden Names

Some Excel macros and add-ins create hidden names. These names exist in a workbook, but don't appear in the Define Name dialog box or the Name box. For example, the Solver add-in creates a number of hidden names. Normally, you can just ignore these hidden names. However, sometimes these hidden names create problems. If you copy a sheet to another workbook, the hidden names are also copied, and they may create a link that is very difficult to track down.

Excel doesn't make it very easy to work with names. For example, you have no way of viewing a *complete* list of names defined in a workbook. When you use the Define Name dialog box, it lists only the worksheet-level names in the active worksheet. And it never displays hidden names.

If you'd like a better tool to help you work with names, you can use the Name Lister utility, which is part of the Power Utility Pak. This utility displays a list of all names, and you can filter the list in a number of ways – for example, you can show only sheet-level names, or show only linked names. The utility is also useful for identifying and deleting “bad” names – names that refer to an invalid range. I included a trial version of the Power Utility Pak on the companion CD-ROM.



Creating Multisheet Names

Names can extend into the third dimension; in other words, they can extend across multiple worksheets in a workbook. You can't simply select the multisheet range and enter a name in the Name box, however. You must use the Define Name dialog box to create a multisheet name. The format for a multisheet reference looks like this:

```
FirstSheet:LastSheet!RangeReference
```

In Figure 3-7, a multisheet name (*DataCube*), defined for A1:C3, extends across Sheet1, Sheet2, and Sheet3.

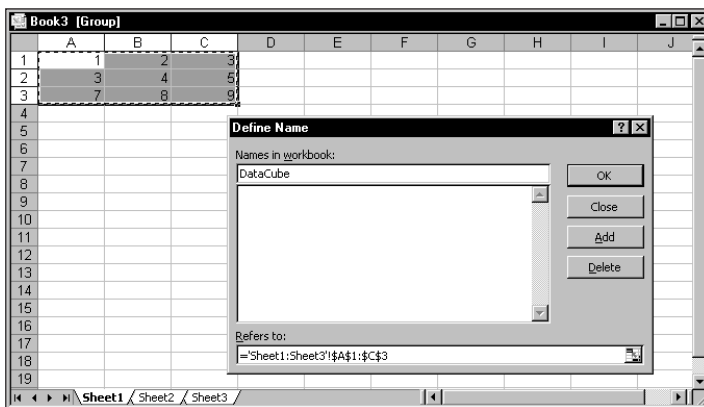


Figure 3-7: Creating a multisheet name

You can, of course, simply type the multisheet range reference into the Refers to field. But if you want to create the name by pointing to the range, you'll find it a bit tricky. Even if you begin by selecting a multisheet range, Excel does not use this selected range address in the Define Name dialog box.

Follow this step-by-step procedure to create a name called *DataCube* that refers to the range A1:C3 across three worksheets (Sheet1, Sheet2, and Sheet3):

1. Activate Sheet1.
2. Choose Insert → Name → Define (or press Ctrl+F3) to display the Define Name dialog box.
3. Type *DataCube* in the Names in workbook field.
4. Activate the Refers to field, and press Del to delete the range reference.

5. Select the range A1:C3 in Sheet1. The following appears in the Refers to field:
`=Sheet1!A1:C3`
6. Press Shift and then click the Sheet tab for Sheet3. You'll find that Excel inexplicably changes the range reference to a single cell. At this point, the following appears in the Refers to field:
`= 'Sheet1:Sheet3'!A1`
7. Reselect the range A1:C3 in Sheet1. The following appears in the Refers to field:
`= 'Sheet1:Sheet3'!A1:C3`
8. Since the Refers to field now has the correct multisheet range address, click OK to close the Define Name dialog box.

After you define the name, you can use it in your formulas. For example, the following formula returns the sum of the values in the range named *DataCube*.

```
=SUM(DataCube)
```



Multisheet names do not appear in the Name box or in the Go To dialog box (which appears when you select Edit → Go To). In other words, Excel enables you to define the name, but it doesn't give you a way to automatically select the cells to which the name refers.

If you insert a new worksheet into a workbook that uses multisheet names, the multisheet names will include the new worksheet – as long as the sheet resides between the first and last sheet in the name's definition. In the preceding example, a worksheet inserted between Sheet1 and Sheet2 will be included in the *DataCube* range. But a worksheet inserted before Sheet1 or after Sheet 3 will not be included.

If you delete the first or last sheet included in a multisheet name, Excel changes the name's range in the Refers to field automatically. In the preceding example, deleting Sheet1 causes the Refers to range of *DataCube* to change to:

```
= 'Sheet2:Sheet3'!$A$1:$C$3
```

A Name's Scope

Normally, when you name a cell or range, you can use that name in all worksheets in the workbook. For example, if you create a name called *RegionTotal* that refers

to the cell A1 on Sheet1, you can use this name in any formula in any worksheet. This is referred to as a workbook-level name (or a global name). By default, all cell and range names are workbook-level names.

Creating Worksheet-Level Names

What if you have several worksheets in a workbook and you want to use the same name (such as *RegionTotal*) on each sheet? In this case, you need to create worksheet-level names (sometimes referred to as local names).

To define a worksheet-level name *RegionTotal*, activate the worksheet in which you want to define the name and choose Insert → Name → Define. The Define Name dialog box then appears. In the Names in workbook field, precede the worksheet-level name with the worksheet name, followed by an exclamation point. For example, to define the name *RegionTotal* on Sheet2, activate Sheet2 and enter the following in the Names in workbook field of the Define Name dialog box:

```
Sheet2!RegionTotal
```

If the worksheet name contains at least one space, enclose the worksheet name in single quotation marks, like this:

```
'Marketing Dept'!RegionTotal
```

You can also create a worksheet-level name by using the Name box. Select the cell or range you want named, click in the Name box, and type the name. Make sure you precede the name with the sheet's name and an exclamation point (as shown above). Press Enter to create the name.

When you write a formula that uses a worksheet-level name on the sheet in which you defined it, you don't need to include the worksheet name in the range name (the Name box won't display the worksheet name either). If you use the name in a formula on a *different* worksheet, however, you must use the entire name (sheet name, exclamation point, and name).



Only the worksheet-level names on the current sheet appear in the Name box. Similarly, only worksheet-level names in the current sheet appear in the list when you open the Paste Name or Define Name dialog boxes.

Combining Worksheet- and Workbook-Level Names

Using worksheet-level names can be a bit confusing because Excel lets you define worksheet-level names even if the workbook contains the same name as a workbook-level name. In such a case, the worksheet-level name takes precedence over the

workbook-level name, but only in the worksheet in which you defined the sheet-level name.

For example, you can define a workbook-level name of *Total* for a cell on Sheet1. You can also define a worksheet-level name of *Sheet2!Total*. When Sheet2 is active, *Total* refers to the worksheet-level name. When any other sheet is active, *Total* refers to the workbook-level name. Confusing? Probably. To make your life easier, I recommend that you simply avoid using the same name at the workbook level and worksheet level.

Referencing Names from Another Workbook

Chapter 2 described how to use links to reference cells or ranges in other workbooks. The same rules apply when using names defined in another workbook.

For example, the following formula uses a range named *MonthlySales*, defined in a workbook named *Budget.xls* (which is assumed to be open):

```
=AVERAGE(Budget.xls!MonthlySales)
```

Working with Range and Cell Names

Once you create range or cell names, you can work with them in a variety of ways. This section describes how to perform common operations with range and cell names.

Creating a List of Names

If you create a large number of names, you may need to know the ranges that each name refers to, particularly if you're trying to track down errors or document your work.

You might want to create a list of all names (and their corresponding addresses) in the workbook. To create a list of names, first move the cell pointer to an empty area of your worksheet (the two-column name list, created at the active cell position, overwrites any information at that location). Use the Insert → Name → Paste command (or press F3). Excel displays the Paste Name dialog box (see Figure 3-8) that lists all the defined names. To paste a list of names, click the Paste List button.

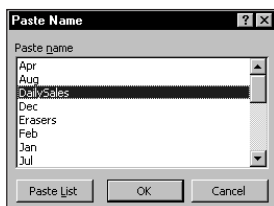


Figure 3-8: The Paste Name dialog box



The list of names does not include worksheet-level names that appear in sheets other than the active sheet.

The list of names pasted to your worksheet occupies two columns. The first column contains the names, and the second column contains the corresponding range addresses. The range addresses in the second column consist of text strings that look like formulas. You can convert such a string to an actual formula by editing the cell (press F2, then press Enter). The string then converts to a formula. If the name refers to a single cell, the formula displays the cell's current value. If the name refers to a range, the formula returns a #VALUE! error.

Using Names in Formulas

After you define a name for a cell or range, you can use it in a formula. If the name is a workbook-level name (the default type), you can use the name in any sheet in the workbook. Just enter the name in place of the cell reference. For example, the following formula calculates the sum of the values in the range named *UnitsSold*:

```
=SUM(UnitsSold)
```

When you write a formula that uses a worksheet-level name on the sheet in which it's defined, you don't need to include the worksheet name in the range name. If you use the name in a formula on a different worksheet, however, you must use the entire name (sheet name, exclamation point, and name). For example, if the name *UnitsSold* represents a worksheet-level name defined on Sheet1, the following formula (on a sheet other than Sheet1) calculates the total of the *UnitsSold* range:

```
=SUM(Sheet1!UnitsSold)
```

As you type a formula, you can select Insert → Name → Paste (or simply press F3) to display the Paste Name dialog box. Select a name from the list, click OK, and Excel inserts that name into your formula. As I previously mentioned, the Paste Name dialog box lists all workbook-level names, plus worksheet-level names for the active sheet only.

If you use a nonexistent name in a formula, Excel displays a #NAME? error, indicating that it cannot find the name you are trying to use. Often, this means that you misspelled the name.

Using the Intersection Operators with Names

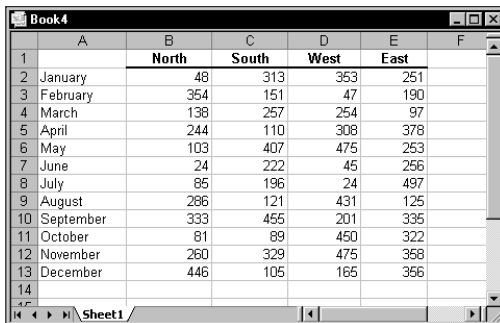
Excel's range intersection operator is a single space character. The following formula, for example, displays the sum of the cells at the intersection of two ranges: B1:C20 and A8:D8:

```
=SUM(B1:C20 A8:D8)
```

The intersection of these two ranges consists of two cells: B8 and C8.

The intersection operator also works with named ranges. Figure 3-9 shows a worksheet containing named ranges that correspond to the row and column labels. For example, the name *January* refers to B2:E2 and the name *North* refers to B2:B13. The following formula returns the contents of the cell at the intersection of the *January* range and the *North* range:

```
=January North
```



The screenshot shows a worksheet window titled 'Book4' with a grid of data. The columns are labeled 'North', 'South', 'West', and 'East' in row 1, and the rows are labeled with months from 'January' to 'December' in column 2. The data values are as follows:

	North	South	West	East
January	48	313	353	251
February	354	151	47	190
March	138	257	254	97
April	244	110	308	378
May	103	407	475	253
June	24	222	45	256
July	85	196	24	497
August	286	121	431	125
September	333	455	201	335
October	81	89	450	322
November	260	329	475	358
December	446	105	165	356

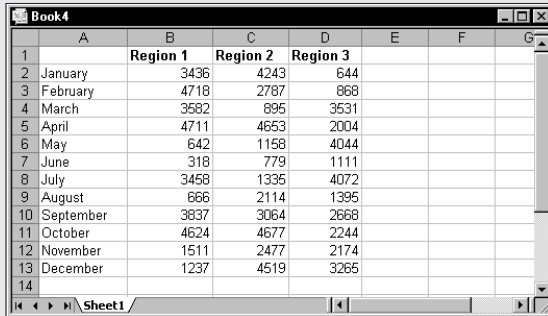
Figure 3-9: This worksheet contains named ranges that correspond to row and column labels.

Using a space character to separate two range references or names is known as *explicit intersection* because you explicitly tell Excel to determine the intersection of the ranges. Excel, however, can also perform *implicit intersections*. An implicit intersection occurs when Excel chooses a value from a multicell range based on the row or column of the formula that contains the reference. An example should clear this up. Figure 3-10 shows a worksheet that contains a range (B3:B8) named *MyData*. Cell D5 contains the simple formula shown here:

```
=MyData
```

Natural Language Formulas? Just Say No!

Beginning with Excel 97, you can use worksheet labels in your formulas, even if you haven't officially defined the names. Microsoft calls this "natural language formulas." For example, the workbook, shown in the accompanying figure, contains no defined names.



	A	B	C	D	E	F	G
1		Region 1	Region 2	Region 3			
2	January	3436	4243	644			
3	February	4718	2787	868			
4	March	3582	895	3531			
5	April	4711	4653	2004			
6	May	642	1158	4044			
7	June	318	779	1111			
8	July	3458	1335	4072			
9	August	686	2114	1395			
10	September	3837	3064	2668			
11	October	4624	4677	2244			
12	November	1511	2477	2174			
13	December	1237	4519	3265			
14							

Excel, however, can interpret the row and column labels. For example, the following formula returns the sum of the values in the row labeled January:

```
=SUM(January)
```

You can also make use of the column labels. The following formula, for instance, returns the sum of the values for Region 1:

```
=SUM(Region 1)
```

You can even use multiple labels in a formula. This next formula returns 2787, the value at the intersection of February and Region 2:

```
=February Region 2
```

Using natural language formulas may seem like an easy way to get the benefits of names without going through the trouble of defining names. However, this feature sometimes does not work as advertised. Formulas that use these "pseudonyms" sometimes do not get calculated when the data changes. Even worse, two identical formulas may return different results! Another problem is that, unlike a real named range, you really have no way of determining how Excel interprets a particular label. Finally, Excel imposes a limit of 32,764 natural language formulas; try to use more and Excel will probably crash.

I strongly recommend that you simply ignore this feature and use real names instead. To disable natural language formulas, select Tools → Options. In the Options dialog box that appears, click the Calculation tab, and uncheck the Accept labels in formulas

Continued

Natural Language Formulas? Just Say No! (Continued)

option. This setting is stored with each workbook, so if you open a file that uses natural languages formulas, you may want to turn it off for that file. When you turn this feature off, Excel scans your formula and converts any labels to actual cell references.

The screenshot shows an Excel spreadsheet with columns A through F and rows 1 through 10. Column B contains the values 6, 12, 18, 24, 36, and 48 in rows 3 through 8. Cell D5 contains the value 18. The spreadsheet title is 'Book5' and the sheet name is 'Sheet1'.

	A	B	C	D	E	F
1						
2						
3		6				
4		12				
5		18		18		
6		24				
7		36				
8		48				
9						
10						

Figure 3-10: Range B3:B8 in this worksheet is named *MyData*. Cell D5 demonstrates an implicit intersection.

Notice that cell D5 displays the value from *MyData* that corresponds to the formula's row. Similarly, if you enter the same formula into any other cell in rows 3 through 8, the formula displays the corresponding value from *MyData*. Excel performs an implicit intersection using the *MyData* range and the row that contains the formula. It's as if the following formula is being evaluated:

```
=MyData 5:5
```

If you enter the formula in a row not occupied by *MyData*, the formula returns an error because the implicit intersection returns nothing.

By the way, implicit intersections are not limited to named ranges. In the preceding example, you get the same result if cell D5 contains the following formula (which doesn't use a named range):

```
=$B$2:$B$8
```

Using the Range Operator with Names

You can also use the range operator, which is a colon (:), to work with named ranges. Refer back to Figure 3-9. For example, this formula returns the sum of the values for North through West for January through March (nine cells):

```
=SUM((North January):(West March))
```

Referencing a Single Cell in a Multicell Named Range

You can use Excel's INDEX function to return a single value from a multicell range. Assume that range A1:A50 is named *DataRange*. The following formula displays the second value (the value in A2) in *DataRange*:

```
=INDEX(DataRange,2)
```

The second and third arguments for the INDEX function are optional, although at least one of them must always be specified. The second argument (used in the preceding formula) is used to specify the row offset within the *DataRange* range.

If *DataRange* consists of multiple cells in a single row, use a formula like the following one. This formula omits the second argument for the INDEX function, but uses the third argument that specifies the column offset with the *DataRange* range:

```
=INDEX(DataRange,,2)
```

If the range consists of multiple rows and columns, use both the second and third arguments for the INDEX function. For example, this formula returns the value in the fourth row and fifth column of a range named *DataRange*:

```
=INDEX(DataRange,4,5)
```

Applying Names to Existing Formulas

When you create a name for a cell or range, Excel does not scan your formulas automatically and replace the cell references with your new name. You can, however, tell Excel to “apply” names to a range of formulas.

Select the range that contains the formulas that you want to convert. Then choose Insert → Name → Apply. The Apply Names dialog box will appear, as shown in Figure 3-11. In the Apply Names dialog box, select which names you want applied to the formulas. Only those names that you select will be applied to the formulas.

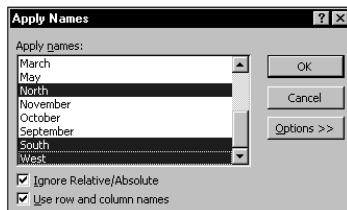


Figure 3-11: The Apply Names dialog box



To apply names to all the formulas in the worksheet, select a single cell before you choose Insert → Name → Apply.

The Ignore Relative/Absolute check box controls how Excel substitutes the range name for the actual address. A cell or range name is usually defined as an absolute reference. If the Ignore Relative/Absolute check box is checked, Excel applies the name only if the reference in the formula matches exactly. In most cases, you will want to ignore the type of cell reference when applying names.

If the Use row and column names check box is checked, Excel takes advantage of the intersection operator when applying names. Excel uses the names of row and column ranges that refer to the cells if it cannot find the exact names for the cells. Excel uses the intersection operator to join the names. Clicking the Options button displays some additional options that are available only when you have the Use row and column names check box checked.

Applying Names Automatically when Creating a Formula

When you insert a cell or range reference into a formula by pointing, Excel automatically substitutes the cell or range name if it has one.



This behavior occurs only in Excel 97 and later.

In some cases, this feature can be very useful. In other cases, it can be annoying; you may prefer to use an actual cell or range reference instead of the name. Unfortunately, you cannot turn off this feature. If you prefer to use a regular cell or range address, you need to type the cell or range reference manually (don't use the pointing technique).

Unapplying Names

Excel does not provide a direct method for unapplying names. In other words, you cannot replace a name in a formula with the name's actual cell reference automatically. However, you can take advantage of a trick described here. You need to change Excel's Transition formula entry option so it emulates 1-2-3. Select Tools → Options, and click the Transition tab in the Options dialog box. Place a check mark next to Transition formula entry, and click OK.

Next, press F2 to edit a formula that contains one or more cell or range names. The formula displays the actual range references instead of the names (the formula bar, however, continues to show the range names). Press Enter to end cell editing. Next, go back to the Options dialog box and remove the check mark from the Transition formula entry check box. You'll find that the edited cell no longer uses names.



The Power Utility Pak includes a utility that enables you to unapply names in selected formulas. The companion CD-ROM contains a trial version of the Power Utility Pak.

Deleting Names

If you no longer need a defined name, you can delete it. Deleting a range name deletes the name only. It *does not* delete the contents of the range. Choose Insert → Name → Define to display the Define Name dialog box. Choose the name that you want to delete from the list and then click the Delete button.



Be extra careful when deleting names. If the name is used in a formula, deleting the name causes the formula to become invalid (it will display #NAME?). It would be very helpful if Excel simply replaced all references to the name with the actual cell or range reference of the deleted name — but it doesn't. However, you can undo the act of deleting a name, so if you find that formulas return #NAME? after you delete a name, select Edit → Undo to get the name back.

Deleting Named Cells or Ranges

If you delete the rows or columns that contain named cells or ranges, the names will not be deleted (as you might expect). Rather, each name will contain an invalid reference. For example, if cell A1 on Sheet1 is named *Interest* and you delete row 1 or column A, *Interest* then refers to =Sheet1!#REF! (i.e., an erroneous reference). If you use *Interest* in a formula, the formula displays #REF.

In order to get rid of this erroneous name, you must delete the name manually using the Insert → Name → Define command. Or, you can redefine the name so it refers to a valid cell or range.

Redefining Names

After you define a name, you may want to change the cell or range to which it refers. Select **Insert** → **Name** → **Define** to display the Define Name dialog box. Select the name that you want to change, and then edit the cell or range address in the Refers to field. If you prefer, you can click the Refers to field and select a new cell or range by pointing in the worksheet.

Changing Names

Excel doesn't provide a simple way to change a name once you create one. If you create a name and then realize that you prefer a different name – or, perhaps, that you spelled it incorrectly – you must create the new name and then delete the old name. In the Define Name dialog box, select the old name in the list of names, change the text in the Names in workbook field to the new name, and click the Add button. Then select the old name again and click the Delete button.

When you change a name, Excel does not automatically adjust formulas that use the name. You can, however, use the **Edit** → **Replace** command to find and replace occurrences of the old name with the new name.

Viewing Named Ranges

When you zoom a worksheet to 39 percent or lower, you see a border around the named ranges with the name displayed in blue letters, as shown in Figure 3-12. The border and name do not print; they simply help you visualize the named ranges on your sheet.

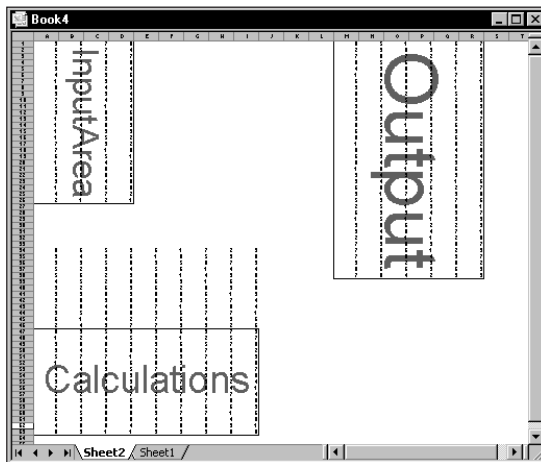


Figure 3-12: Excel displays range names when you zoom a sheet to 39 percent or less.



This feature is available only in Excel 97 or later.

Using Names in Charts

When you create a chart, each data series has an associated SERIES formula. The SERIES formula contains references to the ranges used in the chart. If you have a defined range name, you can edit a SERIES formula and replace the range reference with the name.



Refer to Chapter 17 for additional information about charts.

How Excel Maintains Cell and Range Names

Once you create a name for a cell or range, Excel automatically maintains the name as you edit or modify the worksheet. The following examples assume that Sheet1 contains a workbook-level name (*MyRange*) that refers to =Sheet1!\$C\$3:\$E\$5 (a nine-cell range).

Inserting a Row or Column

When you insert a row above the named range or insert a column to the left of the named range, Excel changes the range reference to reflect its new address. For example, if you insert a new row 1, *MyRange* then refers to =Sheet1!\$C\$4:\$E\$6.

If you insert a new row or column within the named range, the named range expands to include the new row or column. For example, if you insert a new column to the left of column E, *MyRange* then refers to =Sheet1!\$C\$3:\$F\$5.

Deleting a Row or Column

When you delete a row above the named range or delete a column to the left of the named range, Excel adjusts the range reference to reflect its new address. For example, if you delete row 1, *MyRange* refers to =Sheet1!\$B\$3:\$D\$5.

If you delete a row or column within the named range, the name range adjusts accordingly. For example, if you delete column D, *MyRange* then refers to =Sheet1!\$C\$3:\$D\$5.

If you delete all rows or all columns that make up a named range, the named range continues to exist, but it contains an error reference. For example, if you delete columns C, D, and E, *MyRange* then refers to =Sheet1!#REF!. Any formulas that use the name also return errors.

Cutting and Pasting

When you cut and paste an entire named range, Excel changes the reference accordingly. For example, if you move *MyRange* to a new location beginning at cell A1, Excel *MyRange* then refers to =Sheet1!\$A\$1:\$C\$3. Cutting and pasting only a part of a named range does not affect the name's reference.

Potential Problems with Names

Names are great, but they can also cause some problems. This section contains information that you should remember when you use names in a workbook.

Name Problems When Copying Sheets

Excel, as you know, lets you copy a worksheet within the same workbook, or to a different workbook. Let's focus first on copying a sheet within the same workbook. If the copied sheet contains worksheet-level names, those names will also be present on the copy of the sheet, adjusted to use the new sheet name. Usually, this is exactly what you want to happen. But if the workbook contains a workbook-level name that refers to a cell or range on the sheet that's copied, that name will also be present on the copied sheet. However, it will be converted to a worksheet-level name! That is usually *not* what you want to happen.

Consider a workbook that contains one sheet (Sheet1). This workbook has a workbook-level name (called *BookName*) for cell A1, and a worksheet-level name (called *Sheet1!LocalName*) for cell A2. If you make a copy of Sheet1 within the workbook, the new sheet is named Sheet1 (2). You'll find that, after copying the sheet, the workbook contains four names, listed and described in Table 3-1.

TABLE 3-1 NAMES IN A WORKBOOK AFTER COPYING A SHEET

Name	Refers To	Type
BookName	=Sheet1!\$A\$1	Workbook-level
Sheet1!LocalName	=Sheet1!\$A\$2	Worksheet-level
Sheet1 (2)!BookName	='Sheet1 (2)!\$A\$1	Worksheet-level
Sheet1 (2)!LocalName	='Sheet1 (2)!\$A\$2	Worksheet-level

This proliferation of names when copying a sheet is not only confusing, but can result in errors that can be very difficult to identify. In this case, typing the following formula on the copied sheet displays the contents of cell A1 in the copied sheet:

```
=BookName
```

In other words, the newly created worksheet-level name (not the original workbook-level name) is being used.

If you copy the worksheet from a workbook containing a name that refers to a multisheet range, you also copy this name. A #REF! error appears in its Refers to definition.

When you copy a sheet to a new workbook, all of the names in the original workbook that refer to cells on the copied sheet are also copied to the new workbook. This includes both workbook-level and worksheet-level names.



Copying and pasting cells from one sheet to another does not copy names, even if the copied range contains named cells.

Bottom line? You must use caution when copying sheets from a workbook that uses names. After copying the sheet, check the names and delete those that you didn't intend to be copied.

Name Problems when Deleting Sheets

When you delete a worksheet that contains cells used in a workbook-level name, you'll find that the name is not deleted. The name remains with the workbook, but it contains an erroneous reference in its Refers to definition.

Figure 3-13 shows the Define Name dialog box that displays an erroneous name. The workbook originally contained a sheet named Sheet1, which had a named range (a workbook-level name, *MyRange*) for A1:F12. After deleting Sheet1, the name *MyRange* still exists in the workbook, but the Refers to field in the Define Name dialog box displays the following:

```
=#REF!$A$1:$F$12
```

As far as I can tell, keeping erroneous names in a workbook doesn't cause any harm, but it's still a good practice to delete all names that contain an erroneous reference.

Naming Charts and Objects

When you add a chart or any other type of object to a worksheet, the object has a default name. For example, the first chart on a worksheet is named *Chart 1*. When you add a shape (such as a Rectangle or TextBox), the name reflects the type of object (for example, *Rectangle 3*).

To change the name of an object, select it, type the new name in the Name box, and press Enter. Naming charts is an exception. To rename a chart, you must first select the entire chart object (the container for the chart). To do so, press Ctrl while you click the chart.

Excel is a bit inconsistent with regard to the Name box. Although you can use the Name box to rename an object, the Name box does not display a list of objects. To select an object using the Name box, you must type the exact name of the object. Also, you'll find that the Define Name dialog box does not list the names of objects.

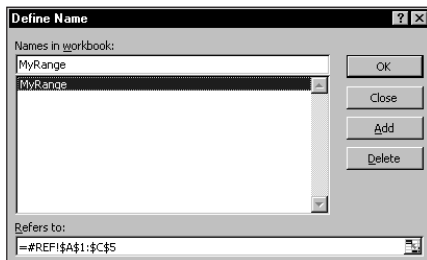


Figure 3-13: Deleting the sheet that contains the cell for MyRange causes an erroneous reference.

The Secret to Understanding Names

Excel users often refer to *named ranges* and *named cells*. In fact, I've used these terms frequently throughout this chapter. Actually, this terminology is not quite accurate.

Here's the secret to understanding names:

When you create a name, you're actually creating a named formula. Unlike a normal formula, a named formula doesn't exist in a cell. Rather, it exists in Excel's memory.

This is not exactly an earth-shaking revelation, but keeping this "secret" in mind will help you understand the advanced naming techniques that follow.

When you work with the Define Name dialog box, the Refers to field contains the formula, and the Names in workbook field contains the formula's name. You'll

find that the contents of the Refers to field always begin with an equal sign, which makes it a formula.

As you can see in Figure 3-14, the workbook contains a name (*InterestRate*) for cell B1 on Sheet1. The Refers to field lists the following formula:

```
=Sheet1!$B$1
```

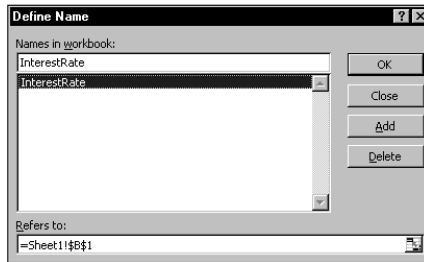


Figure 3-14: Technically, the name *InterestRate* is a named formula, not a named cell.

Whenever you use the name *InterestRate*, Excel actually evaluates the formula with that name and returns the result. For example, you might type this formula into a cell:

```
=InterestRate*1.05
```

When Excel evaluates this formula, it first evaluates the formula named *InterestRate* (which exists only in memory, not in a cell). It then multiplies the result of this named formula by 1.05 and displays the result. This cell formula, of course, is equivalent to the following formula, which uses the actual cell reference instead of the name:

```
=Sheet1!$B$1*1.05
```

At this point, you may be wondering if it's possible to create a named formula that doesn't contain any cell references. The answer comes in the next section.

Naming Constants

Consider a worksheet that generates an invoice and calculates sales tax for a sales amount. The common approach is to insert the sales tax rate value into a cell, and then use this cell reference in your formulas. To make things easier, you probably would name this cell something like *SalesTax*.

You can do this another way. Figure 3-15 demonstrates the following steps:

1. Choose Insert → Name → Define (or press Ctrl+F3) to bring up the Define Name dialog box.
2. Enter the name (in this case, **SalesTax**) into the Names in workbook field.
3. Click the Refers to box, delete its contents, and replace it with a simple formula, such as **=.075**
4. Click OK to close the dialog box.

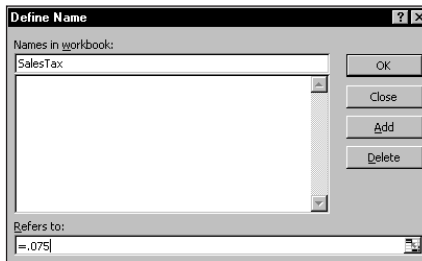


Figure 3-15: Defining a name that refers to a constant

The preceding steps create a named formula that doesn't use any cell references. To try it out, enter the following formula into any cell:

```
=SalesTax
```

This simple formula returns *.075*, the result of the formula named *SalesTax*. Since this named formula always returns the same result, you can think of it as a named constant. And you can use this constant in a more complex formula, such as:

```
=A1*SalesTax
```

SalesTax is a workbook-level name, so you can use it in any worksheet in the workbook.

Naming Text Constants

In the preceding example, the constant consisted of a numeric value. A constant can also consist of text. For example, you can define a constant for a company's name. You can use the Define Name dialog box to create the following formula named *MS*:

```
= "Microsoft Corporation"
```

Then you can use a cell formula such as:

```
= "Annual Report: "&MS
```

This formula returns the text *Annual Report: Microsoft Corporation*.



Names that do not refer to ranges do not appear in the Name box or in the Go To dialog box (which appears when you press F5). This makes sense, because these constants don't reside anywhere tangible. They *do* appear in the Paste Names dialog box, however, which *does* make sense, because you'll use these names in formulas.

As you might expect, you can change the value of the constant at any time by accessing the Define Name dialog box and simply changing the value in the Refers to box. When you close the dialog box, Excel uses the new value to recalculate the formulas that use this name.

Although this technique is useful in many situations, changing the value takes some time. Having a constant located in a cell makes it much easier to modify. If the value is truly a "constant," however, you won't need to change it.

Using Worksheet Functions in Named Formulas

Figure 3-16 shows another example of a named formula. In this case, the formula is named *ThisMonth*, and the actual formula is:

```
=MONTH(TODAY())
```

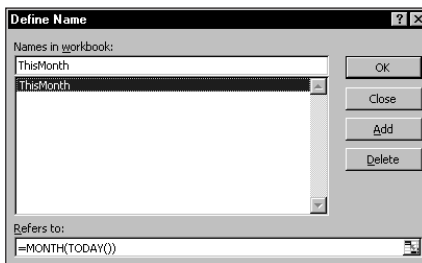


Figure 3-16: Defining a named formula that uses worksheet functions

The formula in Figure 3-16 uses two worksheet functions. The TODAY function returns the current date and the MONTH function returns the month number of its date argument. Therefore, you can enter a formula such as the following into a cell

and it will return the number of the current month. For example, if the current month is April, the formula returns 4.

```
=ThisMonth
```

A more useful named formula would return the actual month name, as text. To do so, create a formula named *MonthName*, defined as:

```
=TEXT(TODAY(),"mmmm")
```

Now enter the following formula into a cell and it returns the current month name, as text. In the month of April, the formula returns the text *April*.

```
=MonthName
```

Using Cell and Range References in Named Formulas

Figure 3-17 shows yet another example of creating a named formula, this time with a cell reference. This formula, named *FirstChar*, returns the first character of the contents of cell A1 on Sheet1. The named formula is:

```
=LEFT(Sheet1!$A$1,1)
```

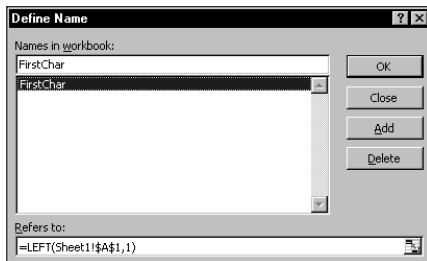


Figure 3-17: Defining a named formula that uses a cell reference

After creating this named formula, you can enter the following formula into a cell. The formula always returns the first character of cell A1 on Sheet1.

```
=FirstChar
```

The next example uses a range reference in a named formula. Figure 3-18 shows the Define Name dialog box when defining the following named formula (named *Total*).

```
=SUM(Sheet1!$A$1:$D$4)
```

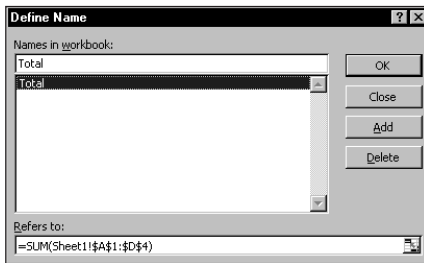


Figure 3-18: Defining a named formula that uses a range reference

After creating this named formula, you can enter the following formula into any cell on any sheet. The formula returns the sum of the values in A1:D4 on Sheet1.

```
=Total
```

Notice that the cell references in the two preceding named formulas are absolute references. By default, all cell and range references in named formulas use an absolute reference, with the worksheet qualifier. But, as you can see in the next section, overriding this default behavior by using a relative cell reference can result in some very interesting named formulas!

Using Named Formulas with Relative References

As I noted previously, when you use the Define Name dialog box to create a named formula that refers to cells or ranges, the Refers to field always uses absolute cell references and the references include the sheet name qualifier. In this section, I describe how to use relative cell and range references in named formulas.

USING A RELATIVE CELL REFERENCE

Let's begin with a simple example. Follow these steps to create a named formula that uses a relative reference:

1. Start with an empty worksheet.
2. Select cell A1 (this step is very important).
3. Select Insert → Name → Define to bring up the Define Name dialog box.
4. Enter `CellToRight` in the Names in workbook field.
5. Delete the contents of the Refers to field, and type the following formula (don't point to the cell in the sheet):

```
=Sheet1!B1
```

6. Click OK to close the Define Name dialog box.
7. Type something (anything) into cell B1.

8. Enter this formula into cell A1:

```
=CellToRight
```

You'll find that the formula in A1 simply returns the contents of cell B1.

Next, copy the formula in cell A1 down a few rows. Then enter some values in column B. You'll find that the formula in column A returns the contents of the cell to the right. In other words, the named formula (*CellToRight*) acts in a relative manner.

You can use the *CellToRight* name in any cell (not just cells in column A). For example, if you enter `=CellToRight` into cell D12, it returns the contents of cell E12.

To demonstrate that the formula named *CellToRight* truly uses a relative cell reference, activate any cell other than cell A1 and display the Define Name dialog box (see Figure 3-19). Select the *CellToRight* item in the list box and examine the Refers to field. You'll see that the formula varies, depending on the active cell. For example, if cell E5 is selected when the Define Name dialog box is displayed, the formula for *CellToRight* appears as:

```
=Sheet1!F5
```

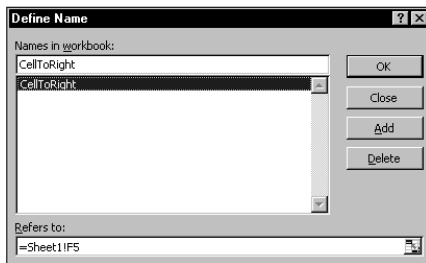


Figure 3-19: The *CellToRight* named formula varies, depending on the active cell.

If you use the *CellToRight* name on a different worksheet, you'll find that it continues to reference the cell to the right – but it's the cell with the same address on Sheet1. This happens because the named formula includes a sheet reference. To modify the named formula so it works on any sheet, follow these steps:

1. Activate cell A1 on Sheet1.
2. Select Insert → Name → Define to bring up the Define Name dialog box.
3. In the Define Name dialog box, click the *CellToRight* item in the list box.
4. Delete the contents of the Refers to field, and type this formula:

```
=!B1
```

5. Click OK to close the Define Name dialog box.

After making this change, you'll find that the *CellToRight* named formula works correctly on any worksheet in the workbook.



The named formula does not work if you use it in a formula in column IV because the formula attempts to reference a nonexistent cell (there is no column to the right of column IV).

USING A RELATIVE RANGE REFERENCE

This example expands upon the previous example, and demonstrates how to create a named formula that sums the values in 10 cells directly to the right of a particular cell. To create this named formula, follow these steps:

1. Activate cell A1.
2. Select Insert → Name → Define to bring up the Define Name dialog box.
3. Enter **Sum10Cells** in the Names in workbook field.
4. Enter this formula in the Refers to field:

```
=SUM(!B1:!K1)
```

After creating this named formula, you can insert the following formula into any cell in any sheet, and it will display the sum of the 10 cells directly to the right:

```
=Sum10Cells
```

For example, if you enter this formula into cell D12, it returns the sum of the values in the 10-cell range E12:N12.

Note that, because cell A1 was the active cell when you defined the named formula, the relative references used in the formula definition are relative to cell A1. Also note that the sheet name was not used in the formula. Omitting the sheet name (but including the exclamation point) causes the named formula to work in any sheet.

If you select cell D12 and then bring up the Define Name dialog box, you'll see that the Refers to field for the *Sum10Cells* name displays the following:

```
=SUM(!E12:!N12)
```



The *Sum10Cells* named formula does not work if you use it in a cell that resides in a column beyond column IL. That's because the formula becomes invalid as it tries to reference a nonexistent cell beyond column IV.

USING A MIXED RANGE REFERENCE

As I discussed in Chapter 2, a cell reference can be absolute, relative, or mixed. A mixed cell reference consists of either of the following:

- ◆ An absolute column reference and a relative row reference (for example, \$A1)
- ◆ A relative column reference and an absolute row reference (for example, A\$1)

As you might expect, a named formula can use mixed cell references. To demonstrate, activate cell B1. Use the Define Name dialog box to create a formula named *FirstInRow*, using this formula definition:

```
=!$A1
```

This formula uses an absolute column reference and a relative row reference. Therefore, it always returns a value in column A. The row depends on the row in which you use the formula. For example, if you enter the following formula into cell F12, it displays the contents of cell A12:

```
=FirstInRow
```



You cannot use the *FirstInRow* formula in column A because it generates a circular reference — a formula that refers to itself.

Advanced Techniques That Use Names

This section presents several examples of advanced techniques that use names. The examples assume that you're familiar with the naming techniques described earlier in this chapter.

Using the INDIRECT Function with a Named Range

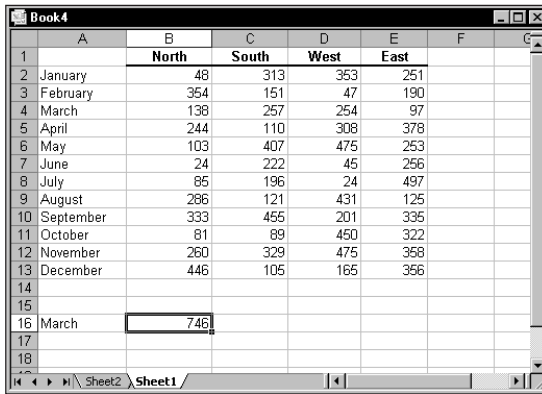
Excel's INDIRECT function lets you specify a cell address indirectly. For example, if cell A1 contains the text *C45*, this formula returns the contents of cell C45:

```
=INDIRECT(A1)
```

The INDIRECT function also works with named ranges. Figure 3-20 shows a worksheet with 12 range names that correspond to the month names. For example, *January* refers to the range B2:E2. Cell B16 contains the following formula:

```
=SUM(INDIRECT(A16))
```

This formula essentially returns the sum of the named range entered as text in cell A16.



	A	B	C	D	E	F
1		North	South	West	East	
2	January	48	313	353	251	
3	February	354	151	47	190	
4	March	138	257	254	97	
5	April	244	110	308	378	
6	May	103	407	475	253	
7	June	24	222	45	256	
8	July	85	196	24	497	
9	August	286	121	431	125	
10	September	333	455	201	335	
11	October	81	89	450	322	
12	November	260	329	475	358	
13	December	446	105	165	356	
14						
15						
16	March	746				
17						
18						

Figure 3-20: Using the INDIRECT function with a named range.



In Excel 97 or later, you can use the Data → Validation command to insert a drop-down box in cell A16 (use the List option in the Data Validation dialog box, and specify A2:A13 as the list source). This allows the user to select a month name from a list; the total for the selected month then displays in B16.

You can also reference worksheet-level names with the INDIRECT function. For example, suppose you have a number of worksheets named *Region1*, *Region2*, and so on. Each sheet contains a worksheet-level name called *TotalSales*. This formula retrieves the value from the appropriate sheet, using the sheet name typed in cell A1:

```
=INDIRECT(A1&"!TotalSales")
```

Using the INDIRECT Function to Create a Named Range with a Fixed Address

It's possible to create a name that always refers to a specific cell or range, even if you insert new rows or columns. For example, suppose you want a range named

UpperLeft to always refer to the range A1. If you create the name using standard procedures, you'll find that inserting a new row 1 causes the *UpperLeft* range to change to A2. Or, inserting a new column causes the *UpperLeft* range to change to B1. To create a named range that uses a fixed address that never changes, create a named formula using the following Refers to definition:

```
=INDIRECT("$A$1")
```

After creating this named formula, *UpperLeft* will always refer to cell A1, even if you insert new rows or columns. The `INDIRECT` function, in the preceding formula, lets you specify a cell address indirectly by using a text argument. Because the argument appears in quotation marks, it never changes.



Because this named formula uses a function, it does not appear in the Go To dialog box or in the Name box.

Using Arrays in Named Formulas

An array is a collection of items. You can visualize an array as a single-column vertical collection, a single-row horizontal collection, or a multirow and multi-column collection.



Part IV of this book discusses arrays and array formulas, but this topic is also relevant when discussing names.

You specify an array by using brackets. A comma or semicolon separates each item in the array. Use a comma to separate items arranged horizontally and use a semicolon to separate items arranged vertically.

Use the Define Name dialog box to create a formula named *MonthNames* that consists of the following formula definition:

```
={"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" }
```

This formula defines a 12-item array of text strings, arranged horizontally.



When you type this formula, make sure that you include the brackets. Entering a formula into the Define Name dialog box is different from entering an array formula into a cell.

After you define the *MonthNames* formula, you can use it in a formula. However, your formula needs to specify which array item to use. The INDEX function is perfect for this. For example, the following formula returns *Aug*:

```
=INDEX(MonthNames,8)
```

You can also display the entire 12-item array, but it requires 12 adjacent cells to do so. For example, to enter the 12 items of the array into A1:L1, follow these steps:

1. Use the Define Name dialog box to create the formula named *MonthNames*.
2. Select the range A1:L1.
3. Type `=MonthNames` in the formula bar.
4. Press Ctrl+Shift+Enter.

Using Ctrl+Shift+Enter tells Excel to insert an array formula into the selected cells. In this case, the single formula is entered into 12 adjacent cells in Figure 3-21. Excel places brackets around an array formula to remind you that it's a special type of formula. If you examine any cell in A1:L1, you'll see its formula listed as:

```
{=MonthNames}
```

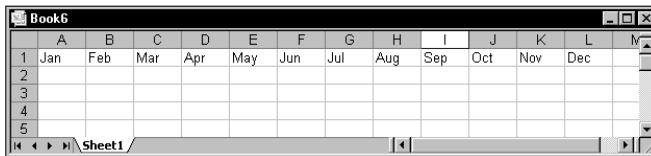


Figure 3-21: You can enter a named formula that contains a 12-item array into 12 adjacent cells.

Creating a Dynamic Named Formula

A dynamic named formula is a named formula that refers to a range not fixed in size. You may find this concept difficult to grasp, so a quick example is in order.

Examine the worksheet shown in Figure 3-22. This sheet contains a listing of sales by month, through the month of May.

	A	B	C	D	E	F
1	January	1,043				
2	February	1,123				
3	March	1,321				
4	April	2,244				
5	May	1,233				
6	June					
7	July					
8	August					
9	September					
10	October					
11	November					
12	December					
13						
14						

Figure 3-22: You can use a dynamic named formula to represent the sales data in column B.

Suppose you want to create a name (*SalesData*) for the data in column B, and you don't want this name to refer to empty cells. In other words, the reference for the *SalesData* range would change each month as you add a new sales figure. You could, of course, use the Define Name dialog box to change the range name definition each month. Or, you could create a dynamic named formula that changes automatically as you enter new data.

To create a dynamic named formula, start by recreating the worksheet shown in Figure 3-22. Then follow these steps:

1. Bring up the Define Name dialog box.
2. Enter *SalesData* in the Names in workbook field.
3. Enter the following formula in the Refers to field:
`=OFFSET(Sheet1!B1,0,0,COUNTA(Sheet1!$B:$B),1)`
4. Click OK to close the Define Name dialog box.

The preceding steps created a named formula that uses Excel's `OFFSET` and `COUNTA` functions. To try out this formula, enter the following formula into any cell not in column B:

```
=SUM(SalesData)
```

This formula returns the sum of the values in column B. Note that *SalesData* does not display in the Name box and does not appear in the Go To dialog box. You can, however, bring up the Go To dialog box and type *SalesData* to select the range.

At this point, you may be wondering about the value of this exercise. After all, a simple formula such as the following does the same job, without the need to define a formula:

```
=SUM(B:B)
```

The value of using dynamic named formulas becomes apparent when creating a chart. You can use this technique to create a chart with a data series that adjusts automatically as you enter new data.



Refer to Chapter 17 for an example that uses this technique to create a dynamic chart.

Summary

This chapter introduced the concept of names. I described how to create and modify names, and compared workbook-level names with worksheet-level names. The chapter provided many examples of using names in your workbooks, and also revealed the secret to understanding names—every name is actually a named formula.

Chapter 4 presents an introduction and overview of Excel's worksheet functions.

Part II

Using Functions in Your Formulas

CHAPTER 4

Introducing Worksheet Functions

CHAPTER 5

Manipulating Text

CHAPTER 6

Working with Dates and Times

CHAPTER 7

Counting and Summing Techniques

CHAPTER 8

Lookups

CHAPTER 9

Databases and Lists

CHAPTER 10

Miscellaneous Calculations

Chapter 4

Introducing Worksheet Functions

IN THIS CHAPTER

- ◆ The advantages of using functions in your formulas
- ◆ The various types of arguments used by functions
- ◆ How to enter a function into a formula
- ◆ Excel's function categories

A THOROUGH KNOWLEDGE OF EXCEL'S worksheet functions is essential for anyone who wants to master the art of formulas. This chapter provides an overview of the functions available for use in formulas.

What Is a Function?

A *worksheet function* is a built-in tool that you use in a formula. A typical function (such as SUM) takes one or more arguments, and then returns a result. The SUM function, for example, accepts a range argument and then returns the sum of the values in that range.

You'll find functions useful because they:

- ◆ Simplify your formulas
- ◆ Permit formulas to perform otherwise impossible calculations
- ◆ Speed up some editing tasks
- ◆ Allow "conditional" execution of formulas – giving them rudimentary decision-making capability

The examples in the sections that follow demonstrate each of these points.

Simplify Formulas

Using a built-in function can simplify a formula significantly. For example, you might need to calculate the average of the values in 10 cells (A1:A10). Without the help of any functions, you would need to construct a formula like this:

```
=(A1+A2+A3+A4+A5+A6+A7+A8+A9+A10)/10
```

Not very pretty, is it? Even worse, you would need to edit this formula if you expanded the range to be summed. You can replace this formula with a much simpler one that uses one of Excel's built-in worksheet functions. For example, the following formula uses Excel's AVERAGE function:

```
=AVERAGE(A1:A10)
```

Perform Otherwise Impossible Calculations

Functions permit formulas to perform impossible calculations. Perhaps you need to determine the largest value in a range. A formula can't tell you the answer without using a function. This simple formula uses Excel's MAX function to return the largest value in the range A1:D100:

```
=MAX(A1:D100)
```

Speed Up Editing Tasks

Functions can sometimes eliminate manual editing. Assume that you have a worksheet that contains 1,000 names in cells A1:A1000 and that all the names appear in all-uppercase letters. Your boss sees the listing and informs you that you need to mail merge the names with a form letter and that the use of all uppercase is not acceptable. For example, JOHN F. CRANE must appear as John F. Crane. You *could* spend the rest of the day reentering the list – or you could use a formula such as the following, which uses Excel's PROPER function to convert the text in cell A1 to proper case:

```
=PROPER(A1)
```

Enter this formula in cell B1 and then copy it down to the next 999 rows. Then select B1:B1000 and use the Edit → Copy command to copy the range to the Clipboard. Next, activate cell A1 and use the Edit → Paste Special command (with the Values option) to convert the formulas to values. Delete column B, and you're finished. With the help of a function, you just accomplished several hours of work in less than a minute.

Provide Decision-Making Capability

Functions can also give your formulas decision-making capability. Suppose you have a worksheet that calculates sales commissions. If a salesperson sells more than \$100,000 of product, the commission rate reaches 7.5 percent; otherwise, the commission rate remains at 5.0 percent. Without using a function, you would need to create two different formulas and make sure that you use the correct formula for each sales amount. Note this formula that uses the IF function to check the value in cell A1 and make the appropriate commission calculation:

```
=IF(A1<100000,A1*5%,A1*7.5%)
```

More about Functions

All told, Excel includes more than 300 functions. And if that's not enough, you can purchase additional specialized functions from third-party suppliers, and even create your own custom functions (using VBA).



If you're ready to create your own custom functions, check out Part IV of this book.

The sheer number of available worksheet functions may overwhelm you, but you'll probably find that you use only a dozen or so of the functions on a regular basis. And as you'll see, Excel's Paste Function dialog box (described later in this chapter) makes it easy to locate and insert a function, even if you use it only rarely.



Appendix B contains a complete listing of Excel's worksheet functions, with a brief description of each.

Function Argument Types

If you examine the preceding examples in this chapter, you'll notice that all of the functions used a set of parentheses. The information within the parentheses is referred to as the function's *arguments*. Functions vary in how they use arguments. A function may use:

- ◆ No arguments
- ◆ One argument

- ◆ A fixed number of arguments
- ◆ An indeterminate number of arguments
- ◆ Optional arguments

For example, the RAND function, which returns a random number between 0 and 1, doesn't use an argument. Even if a function doesn't require an argument, you must provide a set of empty parentheses, like this:

```
=RAND()
```

If a function uses more than one argument, then a comma separates the arguments. For example, the LARGE function, which returns the “nth” largest value in a range, uses two arguments. The first argument represents the range; the second argument represents the value for n. The formula below returns the third largest value in the range A1:A100:

```
=LARGE(A1:A100,3)
```



The character used to separate function arguments can be something other than a comma — for example, a semicolon. This character is determined by the List separator setting for your system, which is specified in the Regional Settings dialog box, accessible via the Control Panel.

The examples at the beginning of the chapter used cell or range references for arguments. Excel proves quite flexible when it comes to function arguments, however. The following sections demonstrate additional argument types for functions.

Names as Arguments

As you've seen, functions can use cell or range references for their arguments. When Excel calculates the formula, it simply uses the current contents of the cell or range to perform its calculations. The SUM function returns the sum of its argument(s). To calculate the sum of the values in A1:A20, you can use:

```
=SUM(A1:A20)
```

And, not surprisingly, if you've defined a name for A1:A20 (such as *Sales*), you can use the name in place of the reference:

```
=SUM(Sales)
```

Accommodating Former Lotus 1-2-3 Users

If you've ever used any of the 1-2-3 spreadsheets (or any version of Corel's Quattro Pro), you might recall that these products require you to type an "at" sign (@) before a function name. Excel is smart enough to distinguish functions without you having to flag them with a symbol.

Because old habits die hard, however, Excel accepts @ symbols when you type functions in your formulas, but it removes them as soon as you enter the formula.

These competing products also use two dots (..) as a range reference operator – for example, A1..A10. Excel also enables you to use this notation when you type formulas, but Excel replaces the notation with its own range reference operator, a colon (:).

This accommodation goes only so far, however. Excel still insists that you use the standard Excel function names, and it doesn't recognize or translate the function names used in other spreadsheets. For example, if you enter the 1-2-3 @AVG function, Excel flags it as an error (Excel's name for this function is AVERAGE). For more information about 1-2-3 compatibility, consult Appendix A.



For more information about defining and using names, refer to Chapter 3.

Full-Column or Full-Row as Arguments

In some cases, you may find it useful to use an entire column or row as an argument. For example, the following formula sums all values in column B:

```
=SUM(B:B)
```

Using full-column and full-row references is particularly useful if the range that you're summing changes (if you continually add new sales figures, for instance). If you do use an entire row or column, just make sure that the row or column doesn't contain extraneous information that you don't want included in the sum.

You might think that using such a large range (a column consists of 65,536 cells) might slow down calculation time. Not true. Excel keeps track of the last-used row and last-used column, and will not use cells beyond them when computing a formula result that references an entire column or row.

Literal Values as Arguments

A *literal argument* refers to a value or text string that you enter directly. For example, the SQRT function, which calculates the square root of a number, takes one argument. In the following example, the formula uses a literal value for the function's argument:

```
=SQRT(225)
```

Using a literal argument with a simple function like this one usually defeats the purpose of using a formula. This formula always returns the same value, so you could just as easily replace it with the value 15. You may want to make an exception to this rule in the interest of clarity. For example, you might want to make it perfectly clear that you are computing the square root of 225.

Using literal arguments makes more sense with formulas that use more than one argument. For example, the LEFT function (which takes two arguments) returns characters from the beginning of its first argument; the second argument specifies the number of characters. If cell A1 contains the text "Budget", the following formula returns the first letter, or "B":

```
=LEFT(A1,1)
```

Expressions as Arguments

Excel also enables you to use *expressions* as arguments. Think of an expression as a formula within a formula. When Excel encounters an expression as a function's argument, it evaluates the expression and then uses the result as the argument's value. Here's an example:

```
=SQRT((A1^2)+(A2^2))
```

This formula uses the SQRT function, and its single argument appears as the following expression:

```
(A1^2)+(A2^2)
```

When Excel evaluates the formula, it first evaluates the expression in the argument and then computes the square root of the result.

Other Functions as Arguments

Because Excel can evaluate expressions as arguments, it shouldn't surprise you that these expressions can include other functions. Writing formulas that have functions within functions is sometimes known as *nesting* functions. Excel starts by evaluating the most deeply nested expression and works its way out. Note this example of a nested function:

```
=SIN(RADIANS(B9))
```

The RADIANS function converts degrees to radians, the unit used by all of Excel's trigonometric functions. If cell B9 contains an angle in degrees, the RADIANS function converts it to radians, and then the SIN function computes the sine of the angle.



A formula can contain up to seven levels of nested functions. If you exceed this level, Excel pops up an error message. In the vast majority of cases, this limit poses no problem. Users often exceed this limitation when attempting to create complex formulas comprised of nested IF functions.

Arrays as Arguments

A function can also use an *array* as an argument. An array is a series of values separated by a comma and enclosed in brackets. The formula below uses the OR function with an array as an argument. The formula returns TRUE if cell A1 contains 1, 3, or 5.

```
=OR(A1={1,3,5})
```



See Part IV of this book for more information about working with arrays.

Often, using arrays can help you simplify your formula. The formula below, for example, returns the same result, but uses nested IF functions instead of an array:

```
=IF(A1=1,TRUE,IF(A1=3,TRUE,IF(A1=5,TRUE,FALSE)))
```

Ways to Enter a Function into a Formula

You can enter a function into a formula by typing it manually, or by using the Paste Function dialog box.

Entering a Function Manually

If you're familiar with a particular function – you know how many arguments it takes and the types of arguments – you may choose simply to type the function and its arguments into your formula. Often, this method is the most efficient.



Excel 2002 provides some additional help by displaying a list of the argument names in a small window (see Figure 4-1). If this window gets in your way, you can drag it to a new position.

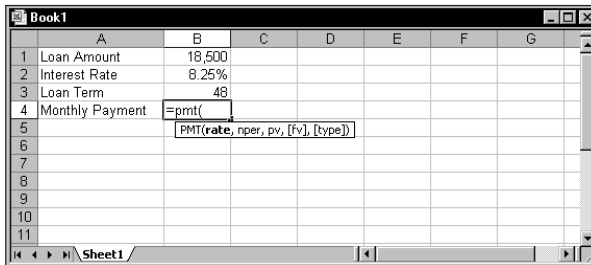


Figure 4-1: When you enter a function, Excel 2002 lists the names of the function arguments.

If you omit the closing parenthesis for a function, Excel adds it for you automatically. For example, if you type =SUM(A1:C12 and press Enter, Excel corrects the formula by adding the right parenthesis.



When you enter a function, Excel always converts the function's name to uppercase. Therefore, it's a good idea to use lowercase when you type functions. If Excel doesn't convert your text to uppercase when you press Enter, then your entry isn't recognized as a function — which means that you spelled it incorrectly or the function isn't available (for example, it may be defined in an add-in not currently installed).

Using the Insert Function Dialog Box to Enter a Function

The Insert Function dialog box assists you by providing a way to enter a function and its arguments in a semi-automated manner. Using the Insert Function dialog box ensures that you spelled the function correctly and that it contains the proper number of arguments in the correct order.



In versions prior to Excel 2002, this dialog box is known as the Paste Function dialog box.

To insert a function, select the function from the Insert Function dialog box, shown in Figure 4-2. You can access this dialog box by using any of the following three methods:

- ◆ Choose the Insert → Function command from the menu.
- ◆ Click the Insert Function button, located next to the formula bar. In versions prior to Excel 2002, this button is located on the Standard toolbar.
- ◆ Press Shift+F3.

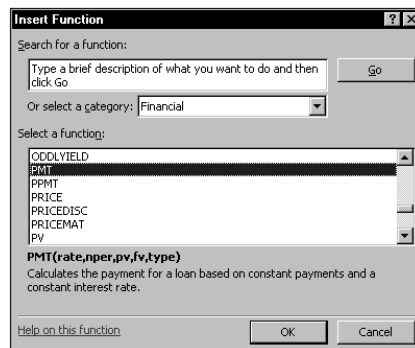


Figure 4-2: The Insert Function dialog box

When you select a category from the drop-down list, the list box displays the functions in the selected category. The Most Recently Used category lists the functions that you've used most recently. The All category lists all the functions available across all categories. Access this category if you know a function's name, but not its category.



A new feature in Excel 2002 enables you to search for a function. Use the field at the top of the Insert Function dialog box. Enter one or more keywords and click Go. Excel will display a list of functions that match your search criteria.

When you select a function in the Select a function list box, notice that Excel displays the function (and its argument names) in the dialog box, along with a brief description of what the function does.

When you locate the function that you want to use, click OK. Excel's Function Arguments dialog box appears, as in Figure 4-3. Use the Function Arguments dialog box to specify the arguments for the function. You can easily specify a range argument by clicking the Collapse Dialog button (the icon at the right edge of each argument field). Excel temporarily collapses the Function Arguments dialog box to a thin box, so that you can select a range in the worksheet.

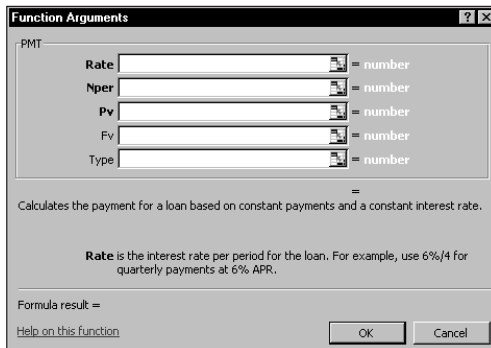


Figure 4-3: The Function Arguments dialog box



The Function Arguments dialog box is new to Excel 2002. Previous versions display the Formula Palette (which is similar in look and functionality).

More Tips for Entering Functions

The following list contains some additional tips to keep in mind when you use the Insert Function dialog box and the Formula Palette to enter functions:

- ◆ Click the Help on this function hyperlink (or press F1) at any time to get help about the function that you selected (see Figure 4-4).

Let Excel Insert Functions for You

Most of the time, you're on your own when it comes to inserting functions. However, at least two situations can arise in which Excel will enter functions for you automatically:

- ◆ When you click the AutoSum button on the Standard toolbar, Excel does a quick check of the selected cells and the surrounding cells. It then proposes a formula that uses the SUM function. If Excel guessed your intentions, just press Enter (or click the AutoSum button a second time) to accept the proposed formula(s). In Excel 2002, the AutoSum button displays an arrow that, when clicked, displays additional functions.
- ◆ When you select the Data → Subtotals command, Excel displays a dialog box that enables you to specify some options. Then it proceeds to insert rows and enter some formulas automatically. These formulas use the SUBTOTAL function.

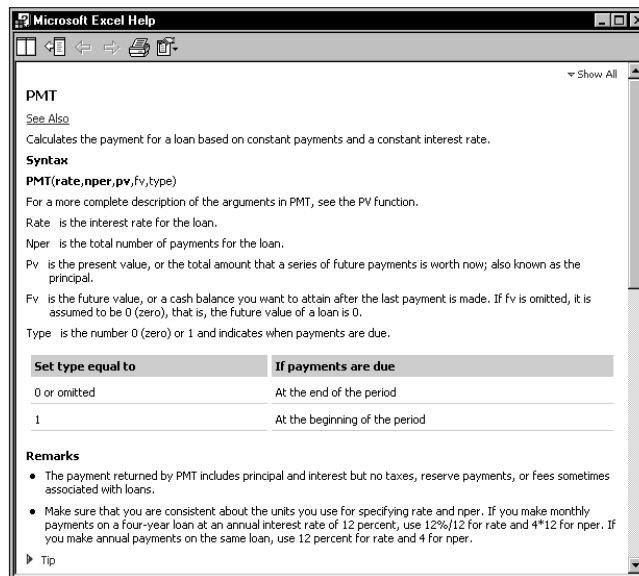


Figure 4-4: Don't forget about Excel's online help. It's the most comprehensive function reference source available.

- ◆ If the active cell already contains a formula that uses a function, clicking the Insert Function button displays the Function Arguments dialog box.

- ◆ You can use the Insert Function dialog box to insert a function into an existing formula. Just edit the formula and move the insertion point to the location where you want to insert the function. Then open the Insert Function dialog box and select the function.
- ◆ If you change your mind about entering a function, click the Cancel button.
- ◆ The number of arguments used by the function that you selected determines the number of boxes you see in the Function Arguments dialog box. If a function uses no arguments, you won't see any boxes. If the function uses a variable number of arguments (as with the AVERAGE function), Excel adds a new box every time you enter an optional argument.
- ◆ On the right side of each box in the Function Arguments dialog box, you'll see the current value for each argument.
- ◆ A few functions, such as INDEX, have more than one form. If you choose such a function, Excel displays another dialog box that enables you to choose which form you want to use.
- ◆ If you only need help remembering a function's arguments, type an equal sign and the function's name, and then press Ctrl+Shift+A. Excel inserts the function with descriptive placeholders for the arguments, as shown in Figure 4-5. You need to replace these placeholders with actual arguments.
- ◆ To locate a function quickly in the Function Name list that appears in the Insert Function dialog box, open the list box, type the first letter of the function name, and then scroll to the desired function. For example, if you select the All category and want to insert the SIN function, click anywhere on the Select a function list box and press S. Excel selects the first function that begins with S. Keep pressing S until you reach the SIN function.
- ◆ If the active cell contains a formula that uses one or more functions, the Function Arguments dialog box enables you to edit each function. In the formula bar, click the function that you want to edit, then click the Insert Function button.

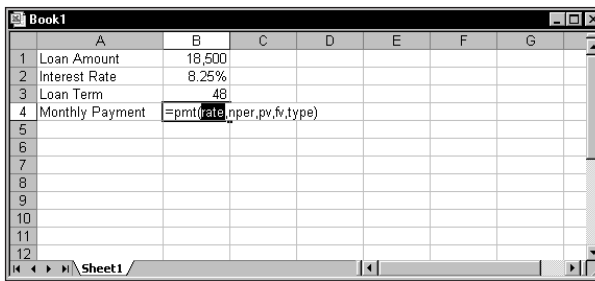


Figure 4-5: Press Ctrl+Shift+A to instruct Excel to display descriptive placeholders for a function.

Function Categories

I list and briefly describe Excel's function categories below.



Refer to subsequent chapters for specific examples of using the functions.

Financial Functions

The financial functions enable you to perform common business calculations that deal with money. For example, you can use the PMT function to calculate the monthly payment for a car loan. (You need to provide the loan amount, interest rate, and loan term as arguments.)

Date & Time Functions

The functions in this category enable you to analyze and work with date and time values in formulas. For example, the TODAY function returns the current date (as stored in the system clock).

Math & Trig Functions

This category contains a wide variety of functions that perform mathematical and trigonometric calculations.



The trigonometric functions all assume radians for angles (not degrees).
Use the RADIANS function to convert degrees to radians.

Statistical Functions

The functions in this category perform statistical analysis on ranges of data. For example, you can calculate statistics such as mean, mode, standard deviation, and variance.



Some of the functions in this category require you to install the Analysis ToolPak add-in.

Lookup and Reference Functions

Functions in this category are used to find (look up) values in lists or tables. A common example is a tax table. You can use the VLOOKUP function to determine a tax rate for a particular income level.

Database Functions

Functions in this category are useful when you need to summarize data in a list (also known as a worksheet database) that meets specific criteria. For example, assume you have a list that contains monthly sales information. You can use the DCOUNT function to count the number of records that describe sales in the Northern region with a value greater than 10,000.

Text Functions

The text functions enable you to manipulate text strings in formulas. For example, you can use the MID function to extract any number of characters beginning at any character position. Other functions enable you to change the case of text (convert to uppercase, for example).

Logical Functions

This category consists of only six functions that enable you to test a condition (for logical TRUE or FALSE). You will find the IF function very useful since it gives your formulas simple decision-making capability.

Information Functions

The functions in this category help you determine the type of data stored within a cell. For example, the ISTEXT function returns TRUE if a cell reference contains text. Or, you can use the ISBLANK function to determine whether a cell is empty. The CELL function returns lots of potentially useful information about a particular cell.

Engineering Functions

The functions in this category can prove useful for engineering applications. They enable you to work with complex numbers, and perform conversions between various numbering and measurement systems.



To use the functions in the Engineering category, you must install the Analysis ToolPak add-in.

User-Defined Functions

Functions that appear in this category are custom worksheet functions created using VBA. These functions can operate just like Excel's built-in functions. One difference, however, is that custom functions do not display a description of each argument in the Paste Function dialog box and Formula Palette.

Other Function Categories

In addition to the function categories described above, Excel includes four other categories that may not appear in the Paste Function dialog box: Commands, Customizing, Macro Control, and DDE/External. These categories appear to be holdovers from older versions of Excel. If you create a custom function, you can assign it to one of these categories. In addition, you may see other function categories created by macros.



Refer to Chapter 23 for information about assigning your custom functions to a function category.

Volatile Functions

Some Excel functions belong to a special class of functions called *volatile*. Excel recalculates a volatile function whenever it recalculates the workbook – even if the formula that contains the function is not involved in the recalculation.

The RAND function represents an example of a volatile function because it generates a new random number every time Excel calculates the worksheet. Other volatile functions include:

AREAS	INDEX	OFFSET
CELL	INDIRECT	ROWS
COLUMNS	NOW	TODAY

As a side effect of using these volatile functions, Excel will always prompt you to save the workbook when you close it – even if you made no changes to it. For example, if you open a workbook that contains any of these volatile functions, scroll around a bit (but don't change anything), and then close the file, Excel will ask whether you want to save the workbook.

You can circumvent this behavior by using the Manual Recalculation mode, with the Recalculate before save option turned off.

Analysis ToolPak Functions

When you feel comfortable with Excel's worksheet functions, you can explore other available functions when you load the Analysis ToolPak. This add-in provides you with dozens of additional worksheet functions.

When you load this add-in, the Paste Function dialog box displays a new category, Engineering. It also adds new functions to the following function categories: Financial, Date & Time, Math & Trig, and Information.

Summary

This chapter provided an introduction to worksheet functions. Excel provides hundreds of functions that you can use in your formulas. In addition, you can use functions defined in add-ins. The remaining chapters in this book provide hundreds of examples of using functions in your formulas. The next chapter demonstrates many of the functions available in the Text category.

Chapter 5

Manipulating Text

IN THIS CHAPTER

- ◆ How Excel handles text entered into cells
- ◆ Excel's worksheet functions that handle text
- ◆ Examples of advanced text formulas
- ◆ Custom VBA text functions

EXCEL, OF COURSE, IS BEST KNOWN for its ability to crunch numbers. However, it is also quite versatile when it comes to handling text. As you know, Excel enables you to enter text for things such as row and column headings, customer names and addresses, part numbers, and just about anything else. And, as you might expect, you can use formulas to manipulate the text contained in cells.

This chapter contains many examples of formulas that use functions to manipulate text. Some of these formulas perform feats you may not have thought possible.

A Few Words about Text

When you enter data into a cell, Excel immediately goes to work and determines whether you're entering a formula, a number (including a date or time), or anything else. Anything else is considered text.



You may hear the term *string* used instead of *text*. You can use these terms interchangeably. Sometimes, they even appear together, as in *text string*.

How Many Characters in a Cell?

In Excel 5 and Excel 95, a single cell can hold up to 255 characters. Beginning with Excel 97, however, Microsoft upped the ante significantly. A single cell in Excel 97 can hold up to 32,000 characters. To put things into perspective, this chapter contains

about 30,000 characters. I certainly don't recommend using a cell in lieu of a word processor, but if you use Excel 97 or later, you really don't have to lose much sleep worrying about filling up a cell with text.



Although a cell can hold up to 32,000 characters, there is a limit on the number of characters that can actually display. And, as I describe later, some functions may not work properly for text strings greater than 255 characters.

Numbers as Text

As I mentioned, Excel distinguishes between numbers and text. If you want to “force” a number to be considered as text, you can do one of the following:

- ◆ Apply the Text number format to the cell. Use Format → Cells, click the Number tab, and select Text from the category list. If you haven't applied other horizontal alignment formatting, the value will appear left aligned in the cell (like normal text).
- ◆ Precede the number with an apostrophe. The apostrophe isn't displayed, but the cell entry will be treated as if it were text.

Even though a cell is formatted as Text (or uses an apostrophe), you can still perform *some* mathematical operations on the cell if the entry *looks* like a number. For example, assume cell A1 contains a value preceded by an apostrophe. The formula that follows will display the value in A1, incremented by 1:

```
=A1+1
```

The formula that follows, however, will treat the contents of cell A1 as 0:

```
=SUM(A1:A10)
```

If you're switching from Lotus 1-2-3, you'll find this to be a significant change. Lotus 1-2-3 never treats text as values. In some cases, treating text as a number can be useful. In other cases, it can cause problems. Bottom line? Just be aware of Excel's inconsistency in how it treats a number formatted as text.



Excel 2002 flags numbers preceded by an apostrophe with a Smart Tag. You can use this Smart Tag to convert the “text” to an actual value.

When a Number Isn't Treated as a Number

If you import data into Excel, you may be aware of a common problem: Sometimes, the imported values are treated as text. Here's a quick way to convert these non-numbers to actual values. Activate any empty cell and enter the value 1. Choose Edit → Copy to copy that value to the Clipboard. Then, select the range that contains the values you need to fix. Choose Edit → Paste Special. In the Paste Special dialog box, select the Multiply option, then click OK. This procedure forces Excel to treat the non-numbers as actual values.

Text Functions

Excel has an excellent assortment of worksheet functions that can handle text. For your convenience, Excel's Insert Function dialog box places most of these functions in the Text category. A few other functions that are relevant to text manipulation appear in other function categories. For example, the ISTEEXT function is in the Information category in the Insert Function dialog box.



Refer to Appendix B for a listing of the functions in the Text category. Or choose Insert → Function to access the Insert Function dialog box, and scroll through the functions in the Text category.

Most of the text functions are not limited for use with text. In other words, these functions can also operate with cells that contain values. Unlike other spreadsheets (such as 1-2-3), Excel is very accommodating when it comes to treating numbers as text and text as numbers.

The examples discussed in this section demonstrate some common (and useful) things you can do with text. You may need to adapt some of these examples for your own use.

Determining Whether a Cell Contains Text

In some situations, you may need a formula that determines the type of data contained in a particular cell. For example, you may use an IF function to return a result only if a cell contains text. Excel provides three functions to help you determine if a particular cell contains text:

- ◆ ISTEEXT
- ◆ CELL
- ◆ TYPE

As you'll see, however, these functions are not always reliable.



The companion CD-ROM includes a workbook that demonstrates these functions (including their problems).

THE ISTEEXT FUNCTION

The ISTEEXT function takes a single argument, and returns TRUE if the argument contains text, and FALSE if it doesn't contain text. The formula that follows returns TRUE if A1 contains a string:

```
=ISTEEXT(A1)
```



The ISTEEXT function, although useful, is certainly not perfect. In fact, it will give you an incorrect result in some cases. Although Excel 97 and later can store a huge amount of text in a cell (up to 32,000 characters), the ISTEEXT function doesn't seem to realize this fact. The ISTEEXT function returns FALSE if its argument refers to a cell that contains more than 255 characters. Excel 2000 corrected this problem, so ISTEEXT works as expected regardless of the amount of text in the cell.

THE TYPE FUNCTION

The TYPE function takes a single argument and returns a value that indicates the type of data in a cell. If cell A1 contains a text string, the formula that follows will return 2 (the code number for text):

```
=TYPE(A1)
```



The TYPE function falls apart when a cell contains more than 255 characters: It returns 16, the code number for an Error value.

THE CELL FUNCTION

Theoretically, the CELL function should help you determine whether a particular cell uses the Text format, or has an apostrophe prefix. The first argument for the CELL function can consist of any of 12 keywords, including *format*, *prefix*, or *type*.

None of these options work as advertised when a number is formatted as Text. For example, if you enter a number into cell A1 and then give it a number format of Text, the following formula returns G, which means Excel considers it formatted using the General format:

```
=CELL("format",A1)
```

Using *prefix* as the first argument for the CELL function returns an apostrophe if a value is preceded by an apostrophe, but it returns nothing if the cell contains a number and is formatted as Text. Using *type* as the first argument in the CELL function also yields inconsistent results. For example, if the cell contains more than 255 characters, the function returns *v* (for value).

Working with Character Codes

Every character that you see on your screen has an associated code number. For Windows systems, Excel uses the standard ANSI character set. The ANSI character set consists of 255 characters, numbered from 1 to 255.

Figure 5-1 shows a portion of an Excel worksheet that displays all of the 255 characters. This example uses the Arial font (other fonts may have different characters).

Row	Col	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character	Code	Character
1	A	1	□	39	'	77	M	115	s	153	™	191	↓	229	å				
2	B	2	□	40	(78	N	116	t	154	§	192	À	230	æ				
3	C	3	□	41)	79	O	117	u	155	>	193	Á	231	ç				
4	D	4	□	42	*	80	P	118	v	156	œ	194	Â	232	è				
5	E	5	□	43	+	81	Q	119	w	157	□	195	Ã	233	é				
6	F	6	□	44	.	82	R	120	x	158	ž	196	Ä	234	ê				
7	G	7	□	45	-	83	S	121	y	159	ÿ	197	Å	235	ë				
8	H	8	□	46	.	84	T	122	z	160		198	Æ	236	ì				
9	I	9	□	47	/	85	U	123	{	161	¡	199	Ç	237	í				
10	J	10	□	48	0	86	V	124		162	¢	200	È	238	î				
11	K	11	□	49	1	87	W	125	}	163	£	201	É	239	ï				
12	L	12	□	50	2	88	X	126	~	164	¤	202	Ê	240	ô				
13	M	13	□	51	3	89	Y	127	□	165	¥	203	Ë	241	ñ				
14	N	14	□	52	4	90	Z	128	€	166	¦	204	Ì	242	ò				
15	O	15	□	53	5	91	[129	□	167	§	205	Í	243	ó				
16	P	16	□	54	6	92	\	130	,	168	¨	206	Î	244	ô				
17	Q	17	□	55	7	93]	131	ƒ	169	©	207	Ï	245	õ				
18	R	18	□	56	8	94	^	132	„	170	®	208	Ð	246	ö				
19	S	19	□	57	9	95	~	133	…	171	«	209	Ñ	247	÷				
20	T	20	□	58	:	96	ˆ	134	†	172	¬	210	Ò	248	ø				
21	U	21	□	59	;	97	a	135	‡	173	-	211	Ó	249	ù				
22	V	22	□	60	<	98	b	136	^	174	®	212	Ô	250	ú				
23	W	23	□	61	=	99	c	137	‰	175	—	213	Õ	251	û				
24	X	24	□	62	>	100	d	138	Š	176	◊	214	Ö	252	ü				
25	Y	25	□	63	?	101	e	139	<	177	±	215	×	253	ý				

Figure 5-1: The ANSI character set (for the Arial font)



The companion CD-ROM includes a copy of this workbook. It has some simple macros that enable you to display the character set for any font installed on your system. This workbook requires Excel 97 or later.

Two functions come into play when dealing with character codes: CODE and CHAR. These functions aren't very useful by themselves. However, they can prove quite useful in conjunction with other functions. I discuss these functions in the following sections.



The CODE and CHAR functions work only with ANSI strings. These functions will not work with double-byte Unicode strings.

THE CODE FUNCTION

Excel's CODE function returns the character code for its argument. The formula that follows returns 65, the character code for uppercase *A*:

```
=CODE("A")
```

If the argument for CODE consists of more than one character, the function uses only the first character. Therefore, this formula also returns 65:

```
=CODE("Abbey Road")
```

THE CHAR FUNCTION

The CHAR function is essentially the opposite of the CODE function. Its argument should be a value between 1 and 255, and the function should return the corresponding character. The following formula, for example, returns the letter *A*:

```
=CHAR(65)
```

To demonstrate the opposing nature of the CODE and CHAR functions, try entering this formula:

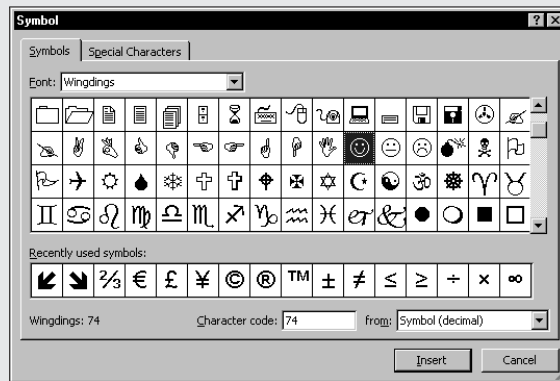
```
=CHAR(CODE("A"))
```

This formula (illustrative rather than useful) returns the letter *A*. First, it converts the character to its code value (65), and then it converts this code back to the corresponding character.

How to Find Special Characters

If you use Excel 2002, don't overlook the handy Symbol dialog box (which appears when you select Insert → Symbol). This dialog box makes it easy to insert special characters (including Unicode characters) into cells. For example, you might (for some strange reason) want to include a smiley face character in your spreadsheet. Access Excel's Symbol dialog box and select the Wingdings font (see the accompanying figure). Examine the characters, locate the smiley face, and click Insert. You'll also find out that this character has a code of 74.

If you use an earlier version of Excel, you can get similar functionality with the Windows Character Map program (charmap.exe).



Assume cell A1 contains the letter *A* (uppercase). The following formula returns the letter *a* (lowercase):

```
=CHAR(CODE(A1)+32)
```

This formula takes advantage of the fact that the alphabetic characters all appear in alphabetical order within the character set, and the lowercase letters follow the uppercase letters (with a few other characters tossed in between). Each lowercase letter lies exactly 32 character positions higher than its corresponding uppercase letter.

Determining Whether Two Strings Are Identical

You can set up a simple logical formula to determine whether two cells contain the same entry. For example, use this formula to determine whether cell A1 has the same contents as cell A2:

```
=A1=A2
```

Excel acts a bit lax in its comparisons when text is involved. Consider the case in which A1 contains the word *January* (initial capitalization), and A2 contains *JANUARY* (all uppercase). You'll find that the previous formula returns TRUE, even though the contents of the two cells are not really the same. In other words, the comparison is not case sensitive.

In many cases, you don't need to worry about the case of the text. But if you need to make an exact, case-sensitive comparison, you can use Excel's EXACT function. The formula that follows returns TRUE only if cells A1 and A2 contain *exactly* the same entry:

```
=EXACT(A1,A2)
```

The following formula returns FALSE because the first string contains a trailing space:

```
=EXACT("zero ", "zero")
```

Joining Two or More Cells

Excel uses an ampersand as its concatenation operator. *Concatenation* is simply a fancy term that describes what happens when you join the contents of two or more cells. For example, if cell A1 contains the text *San Diego*, and cell A2 contains the text *California*, the following formula will return *San DiegoCalifornia*:

```
=A1&A2
```

Notice that the two strings are joined together without an intervening space. To add a space between the two entries (to get *San Diego California*), use a formula like this one:

```
=A1&" "&A2
```

Or, even better, use a comma and a space to produce *San Diego, California*:

```
=A1&", "&A2
```

Another option is to eliminate the quote characters and use the CHAR function, with an appropriate argument. Note this example of using the CHAR function to represent a comma (44) and a space (32):

```
=A1&CHAR(44)&CHAR(32)&A2
```

If you'd like to force a "word wrap," concatenate the strings using CHAR (10), and make sure you apply the wrap text format to the cell. The following example joins the text in cell A1 and the text in cell B1, with a line break in between:

```
=A1&CHAR(10)&B1
```

Here's another example of the CHAR function. The following formula returns the string *Stop* by concatenating four characters returned by the CHAR function:

```
=CHAR(83)&CHAR(116)&CHAR(111)&CHAR(112)
```

Here's a final example of using the & operator. In this case, the formula combines text with the result of an expression that returns the maximum value in column C:

```
= "The largest value in Column C is " & MAX(C:C)
```



Excel also has a CONCATENATE function, which takes up to 30 arguments. This function simply combines the arguments into a single string. You can use this function if you like, but using the & operator results in shorter formulas.

Displaying Formatted Values as Text

Excel's TEXT function enables you to display a value in a specific number format. Although this function may appear to have dubious value, it *does* serve some useful purposes, as the examples in this section demonstrate. Figure 5-2 shows a simple worksheet. The formula in cell D1 is:

```
= "The net profit is " & B3
```

	A	B	C	D	E
1	Gross	\$155,609.84		The net profit is 104616.52	
2	Expenses	\$50,993.32			
3	NET	\$104,616.52			
4					
5					
6					
7					

Figure 5-2: The formula in D1 doesn't display the formatted number.

This formula essentially combines a text string with the contents of cell B3 and displays the result. Note, however, that the contents of B3 are not formatted in any way. You might want to display B3's contents using a currency number format.



Contrary to what you might expect, applying a number format to the cell that contains the formula has no effect. This is because the formula returns a string, not a value.

Note this revised formula that uses the TEXT function to apply formatting to the value in B3:

```
= "The net profit is " & TEXT(B3, "$#,##0.00")
```

This formula displays the text along with a nicely formatted value: *The net profit is \$104,616.52.*

The second argument for the TEXT function consists of a standard Excel number format string. You can enter any valid number format string for this argument.

The preceding example uses a simple cell reference (B3). You can, of course, use an expression instead. Here's an example that combines text with a number resulting from a computation:

```
= "Average Expenditure: " & TEXT(AVERAGE(A:A), "$#,##0.00")
```

This formula might return a string such as *Average Expenditure: \$7,794.57.*

Here's another example that uses the NOW function (which returns the current date and time). The TEXT function displays the date and time, nicely formatted.

```
= "Report printed on " & TEXT(NOW(), "mmm d, yyyy at h:mm AM/PM")
```

The formula might display the following: *Report printed on July 22, 2001 at 3:23 PM.*



Refer to Appendix C for details on Excel number formats.

Displaying Formatted Currency Values as Text

Excel's DOLLAR function converts a number to text using the currency format. It takes two arguments: the number to convert, and the number of decimal places to display. The DOLLAR function uses the regional currency symbol (for example, a \$).

character *n*, which displays as a small square in the Wingdings font. A formula using the REPT function determines the number of characters displayed. Key formulas include:

```
E3: =IF(D3<0,REPT("n",-ROUND(D3*100,0)), "")
F3: =A3
G3: =IF(D3>0,REPT("n",ROUND(D3*100,0)), "")
```

Assign the Wingdings font to cells E3 and G3, and then copy the formulas down the columns to accommodate all the data. Right-align the text in column E and adjust any other formatting. Depending on the numerical range of your data, you may need to change the scaling. Experiment by replacing the 100 value in the formulas. You can substitute any character you like for the *n* in the formulas to produce a different character in the chart.



The workbook shown in Figure 5-3 also appears on the companion CD-ROM.

Padding a Number

You're probably familiar with a common security measure (frequently used on printed checks) in which numbers are padded with asterisks on the right. The following formula displays the value in cell A1, along with enough asterisks to make 24 characters total:

```
=(A1 & REPT(" ",24-LEN(A1)))
```

Or, if you'd prefer to pad the number with asterisks on the left, use this formula:

```
=REPT(" ",24-LEN(A1))&A1
```

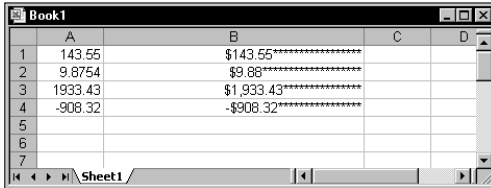
The following formula displays asterisk padding on both sides of the number. It will return 24 characters when the number in cell A1 contains an even number of characters; otherwise, it returns 23 characters.

```
=REPT(" ",12-LEN(A1)/2)&A1&REPT(" ",12-LEN(A1)/2)
```

The preceding formulas are a bit deficient since they don't show any number formatting. Note this revised version that displays the value in A1 (formatted), along with the asterisk padding on the right:

```
=(TEXT(A1,"$#,##0.00")&REPT("*",24-LEN(TEXT(A1,"$#,##0.00"))))
```

Figure 5-4 shows this formula in action.



	A	B	C	D
1	143.55	\$143.55*****		
2	9.8754	\$9.88*****		
3	1933.43	\$1,933.43*****		
4	-908.32	-\$908.32*****		
5				
6				
7				

Figure 5-4: Using a formula to pad a number with asterisks

You can also pad a number by using a custom number format. To repeat the next character in the format to fill the column width, include an asterisk (*) in the custom number format code. For example, use this number format to pad the number with dashes:

```
$#,##0.00*-
```

To pad the number with asterisks, use two asterisks, like this:

```
$#,##0.00**
```



Refer to Appendix C for more information about custom number formats, including additional examples using the asterisk format code.

Removing Excess Spaces and Nonprinting Characters

Often, data imported into an Excel worksheet contains excess spaces or strange (often unprintable) characters. Excel provides you with two functions to help whip your data into shape: TRIM and CLEAN.

- ◆ TRIM: Removes all leading and trailing spaces, and replaces internal strings of multiple spaces by a single space.
- ◆ CLEAN: Removes all nonprinting characters from a string. These “garbage” characters often appear when you import certain types of data.

Of the 255 ANSI character codes, 39 of them comprise nonprinting characters. Specifically, the nonprinting character codes include 1–31, 128–129, 141–144, and 157–158.

This example uses the TRIM function. The formula returns *Fourth Quarter Earnings* (with no excess spaces):

```
=TRIM("  Fourth   Quarter   Earnings  ")
```

Counting Characters in a String

Excel's LEN function takes one argument and returns the number of characters in the cell. For example, assume the string *September Sales* is contained in cell A1. The following formula will return 15:

```
=LEN(A1)
```

Notice that space characters are included in the character count.

The following formula returns the total number of characters in the range A1:A3:

```
=SUM(LEN(A1),LEN(A2),LEN(A3))
```



You will see example formulas that demonstrate how to count the number of specific characters within a string later in this chapter. Also, you may find relevant material in Chapter 7 on counting techniques and Chapter 15 on performing magic with array formulas.

Changing the Case of Text

Excel provides three handy functions to change the case of text:

- ◆ UPPER: Converts the text to ALL UPPERCASE
- ◆ LOWER: Converts the text to all lowercase
- ◆ PROPER: Converts the text to “proper” case (The First Letter In Each Word Is Capitalized)

These functions are quite straightforward. The formula that follows, for example, converts the text in cell A1 to proper case. If cell A1 contained the text *MR. JOHN Q. PUBLIC*, the formula would return *Mr. John Q. Public*.

```
=PROPER(A1)
```

These functions operate only on alphabetic characters; they simply ignore all other characters and return them unchanged.

Extracting Characters from a String

Excel users often need to extract characters from a string. For example, you may have a list of employee names (first and last names) and need to extract the last name from each cell. Excel provides several useful functions for extracting characters:

- ◆ **LEFT**: Returns a specified number of characters from the beginning of a string
- ◆ **RIGHT**: Returns a specified number of characters from the end of a string
- ◆ **MID**: Returns a specified number of characters beginning at any position within a string

The formula that follows returns the last 10 characters from cell A1. If A1 contains fewer than 10 characters, the formula returns all of the text in the cell.

```
=RIGHT(A1,10)
```

This next formula uses the **MID** function to return five characters from cell A1, beginning at character position 2. In other words, it returns characters 2–6.

```
=MID(A1,2,5)
```

The following example returns the text in cell A1, with only the first letter in uppercase. It uses the **LEFT** function to extract the first character and convert it to uppercase. This then concatenates to another string that uses the **RIGHT** function to extract all but the first character (converted to lowercase).

```
=UPPER(LEFT(A1))&RIGHT(LOWER(A1),LEN(A1)-1)
```

If cell A1 contained the text *FIRST QUARTER*, the formula would return *First quarter*.

Replacing Text with Other Text

In some situations, you may need to replace a part of a text string with some other text. For example, you may import data that contains asterisks, and you need to

convert the asterisks to some other character. You could use Excel's Edit → Replace command to make the replacement. If you prefer a formula-based solution, you can take advantage of either of two functions:

- ◆ **SUBSTITUTE:** Replaces specific text in a string. Use this function when you know the character(s) to be replaced, but not the position.
- ◆ **REPLACE:** Replaces text that occurs in a specific location within a string. Use this function when you know the position of the text to be replaced, but not the actual text.

The following formula uses the SUBSTITUTE function to replace 2001 with 2002 in the string *2001 Budget*. The formula returns *2002 Budget*.

```
=SUBSTITUTE("2001 Budget","2001","2002")
```

The following formula uses the SUBSTITUTE function to remove all spaces from a string. In other words, it replaces all space characters with an empty string. The formula returns the title of an excellent Liz Phair CD: *Whitechocolatespaceegg*.

```
=SUBSTITUTE("White chocolate space egg"," ","")
```

The following formula uses the REPLACE function to replace one character beginning at position 5 with nothing. In other words, it removes the fifth character (a hyphen) and returns *Part544*.

```
=REPLACE("Part-544",5,1,"")
```

You can, of course, nest these functions to perform multiple replacements in a single formula. The formula that follows demonstrates the power of nested SUBSTITUTE functions. The formula essentially strips out any of the following seven characters in cell A1: space, hyphen, colon, asterisk, underscore, left parenthesis, and right parenthesis.

```
=SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(A1," ",""),"-",""),":",""),"*",""),"_",""),("(",""),")","")
```

Therefore, if cell A1 contains the string *Part-2A - Z(4M1)_A**, the formula returns *Part2AZ4M1A*.

Finding and Searching within a String

Excel's FIND and SEARCH functions enable you to locate the starting position of a particular substring within a string:

- ◆ **FIND:** Finds a substring within another text string and returns the starting position of the substring. You can specify the character position at which to begin searching. Use this function for case-insensitive text comparisons. Wildcard comparisons are not supported.
- ◆ **SEARCH:** Finds a substring within another text string and returns the starting position of the substring. You can specify the character position at which to begin searching. Use this function for non-case-sensitive text, or when you need to use wildcard characters.

The following formula uses the FIND function and returns 7, the position of the first *m* in the string. Notice that this formula is case sensitive.

```
=FIND("m","Big Mamma Thornton",1)
```

The formula that follows, which uses the SEARCH function, returns 5, the position of the first *m* (either uppercase or lowercase):

```
=SEARCH("m","Big Mamma Thornton",1)
```

You can use the following wildcard characters within the first argument for the SEARCH function:

- ◆ Question mark (?): Matches any single character
- ◆ Asterisk (*): Matches any sequence of characters



If you want to find an actual question mark or asterisk character, type a tilde (~) before the question mark or asterisk.

The next formula examines the text in cell A1 and returns the position of the first three-character sequence that has a hyphen in the middle of it. In other words, it looks for any character followed by a hyphen and any other character. If cell A1 contains the text *Part-A90*, the formula returns 4.

```
=SEARCH("?-?",A1,1)
```

Searching and Replacing within a String

You can use the REPLACE function in conjunction with the SEARCH function to replace part of a text string with another string. In effect, you use the SEARCH function to find the starting location used by the REPLACE function.

For example, assume cell A1 contains the text “Annual Profit Figures.” The following formula searches for the word “Profit,” and replaces it with the word “Loss”:

```
=REPLACE(A1,SEARCH("Profit",A1),6,"Loss")
```

This next formula uses the SUBSTITUTE function to accomplish the same effect in a more efficient manner:

```
=SUBSTITUTE(A1,"Profit","Loss")
```

Advanced Text Formulas

The examples in this section appear more complex than the examples in the previous section. But, as you’ll see, they can perform some very useful text manipulations.



You can access all of the examples in this section on the companion CD-ROM.

Counting Specific Characters in a Cell

This formula counts the number of Bs (uppercase only) in the string in cell A1:

```
=LEN(A1)-LEN(SUBSTITUTE(A1,"B",""))
```

This formula works by using the SUBSTITUTE function to create a new string (in memory) that has all of the Bs removed. Then the length of this string is subtracted from the length of the original string. The result reveals the number of Bs in the original string.

The following formula is a bit more versatile. It counts the number of Bs (both upper- and lowercase) in the string in cell A1.

```
=LEN(A1)-LEN(SUBSTITUTE(SUBSTITUTE(A1,"B",""),"b",""))
```

Counting the Occurrences of a Substring in a Cell

The formulas in the preceding section count the number of occurrences of a particular character in a string. The following formula works with more than one character. It returns the number of occurrences of a particular substring (contained in cell B1) within a string (contained in cell A1). The substring can consist of any number of characters.

```
=(LEN(A1)-LEN(SUBSTITUTE(A1,B1,"")))/LEN(B1)
```

For example, if cell A1 contains the text *Blonde On Blonde* and B1 contains the text *Blonde*, the formula returns 2.

The comparison is case sensitive, so if B1 contains the text *blonde*, the formula returns 0. The following formula is a modified version that performs a case-insensitive comparison:

```
=(LEN(A1)-LEN(SUBSTITUTE(UPPER(A1),UPPER(B1),"")))/LEN(B1)
```

Expressing a Number as an Ordinal

You may need to express a value as an ordinal number. For example, *Today is the 21st day of the month*. In this case, the number 21 converts to an ordinal number by appending the characters *st* to the number.

The characters appended to a number depend on the number. There is no clear pattern, making the construction of a formula more difficult. Most numbers will use the *th* suffix. Exceptions occur for numbers that end with 1, 2, or 3 – except if the preceding number is a 1 (numbers that end with 11, 12, or 13). These may seem like fairly complex rules, but you can translate them into an Excel formula.

The formula that follows converts the number in cell A1 (assumed to be an integer) to an ordinal number:

```
=A1&IF(OR(VALUE(RIGHT(A1,2))={11,12,13}),"th",IF(OR(VALUE(RIGHT(A1))={1,2,3}),CHOOSE(RIGHT(A1),"st","nd","rd"),"th"))
```

This is a rather complicated formula, so it may help to examine its components. Basically, the formula works as follows:

1. If the last two digits of the number consist of 11, 12, or 13, then use *th*.
2. If Rule #1 does not apply, then check the last digit. If the last digit is 1, use *st*. If the last digit is 2, use *nd*. If the last digit is 3, use *rd*.
3. If neither Rule #1 nor Rule #2 apply, use *th*.



The formula uses two arrays, specified by brackets. Refer to Chapter 14 for more information about using arrays in formulas.

Figure 5-5 shows the formula in use.

	A	B	C
1	Number	Ordinal	
2	1	1st	
3	4	4th	
4	7	7th	
5	10	10th	
6	13	13th	
7	16	16th	
8	19	19th	
9	22	22nd	
10	25	25th	
11	28	28th	
12	31	31st	
13	34	34th	
14	37	37th	
15	40	40th	
16	43	43rd	
17	46	46th	
18	49	49th	
19	52	52nd	
20			

Figure 5-5: Using a formula to express a number as an ordinal

Determining a Column Letter for a Column Number

This next formula returns a worksheet column letter (ranging from A to IV) for the value contained in cell A1. For example, if A1 contains 29, the formula returns AC.

```
=IF(A1>26,CHAR(64+INT((A1-1)/26)), "")&CHAR(65+MOD(A1-1,26))
```

Note that the formula doesn't check for a valid column number. In other words, if A1 contains a value less than 1 or greater than 256, the formula will still give an answer—albeit a meaningless one. The following modified version includes an IF function to ensure a valid column:

```
=IF(AND(A1>0,A1<257),IF(A1>26,CHAR(64+INT((A1-1)/26)), "")&CHAR(65+MOD(A1-1,26)), "")
```

Extracting a Filename from a Path Specification

The following formula returns the filename from a full path specification. For example, if cell A1 contains *c:\windows\desktop\myfile.xls*, the formula returns *myfile.xls*.

```
=MID(A1,FIND(" ",SUBSTITUTE(A1,"\\","*",LEN(A1)-LEN(SUBSTITUTE(A1,"\\",""))))+1,LEN(A1))
```

This formula assumes that the system path separator consists of a backslash (\). It essentially returns all of the text following the last backslash character. If cell A1 doesn't contain a backslash character, the formula returns an error.

Extracting the First Word of a String

To extract the first word of a string, a formula must locate the position of the first space character, and then use this information as an argument for the LEFT function. The following formula does just that:

```
=LEFT(A1,FIND(" ",A1)-1)
```

This formula returns all of the text prior to the first space in cell A1. However, the formula has a slight problem: It returns an error if cell A1 consists of a single word. A slightly more complex formula that checks for the error with an IF function solves that problem:

```
=IF(ISERR(FIND(" ",A1)),A1,LEFT(A1,FIND(" ",A1)-1))
```

Extracting the Last Word of a String

Extracting the last word of a string is more complicated, since the FIND function only works from left to right. Therefore, the problem rests with locating the *last* space character. The formula that follows, however, solves this problem. It returns the last word of a string (all of the text following the last space character):

```
=RIGHT(A1,LEN(A1)-FIND("*",SUBSTITUTE(A1," ","*",LEN(A1)-LEN(SUBSTITUTE(A1,"","")))))
```

This formula, however, has the same problem as the first formula in the preceding section: It fails if the string does not contain at least one space character. The following modified formula uses an IF function to count the number of spaces in cell A1. If it contains no spaces, the entire contents of cell A1 are returned. Otherwise, the previous formula kicks in.

```
=IF(LEN(A1)-LEN(SUBSTITUTE(A1,"",""))=0,A1,RIGHT(A1,LEN(A1)-FIND("*",SUBSTITUTE(A1," ","*",LEN(A1)-LEN(SUBSTITUTE(A1,"","")))))
```

Extracting All but the First Word of a String

The following formula returns the contents of cell A1, except for the first word:

```
=RIGHT(A1,LEN(A1)-FIND(" ",A1,1))
```

If cell A1 contains *2002 Operating Budget*, the formula returns *Operating Budget*.

Extracting First Names, Middle Names, and Last Names

Suppose you have a list consisting of people's names in a single column. You have to separate these names into three columns: one for the first name, one for the middle name or initial, and one for the last name. This task is more complicated than you may think, since not every name has a middle initial. However, you can still do it.



The task becomes a *lot* more complicated if the list contains names with titles (such as Mr. or Dr.) or names followed by additional details (such as Jr. or III). In fact, the following formulas will *not* handle these complex cases. However, they still give you a significant head start if you're willing to do a bit of manual editing to handle the special cases.

The formulas that follow all assume that the name appears in cell A1. You can easily construct a formula to return the first name:

```
=LEFT(A1,FIND(" ",A1)-1)
```

Returning the middle name or initial is much more complicated since not all names have a middle initial. This formula returns the middle name (if it exists). Otherwise, it returns nothing.

```
=IF(ISERR(MID(A1,FIND(" ",A1)+1,IF(ISERR(FIND(" ",A1,FIND(" ",A1)+1)),FIND(" ",A1),FIND(" ",A1,FIND(" ",A1)+1))-FIND(" ",A1)-1)),",",MID(A1,FIND(" ",A1)+1,IF(ISERR(FIND(" ",A1,FIND(" ",A1)+1)),FIND(" ",A1),FIND(" ",A1,FIND(" ",A1)+1))-FIND(" ",A1)-1))
```

Finally, this formula returns the last name:

```
=RIGHT(A1,LEN(A1)-FIND("*",SUBSTITUTE(A1," ","*",LEN(A1)-LEN(SUBSTITUTE(A1," ","")))))
```

The formula that follows is a much shorter way to extract the middle name. This formula is useful if you use the other formulas to extract the first name and the last name. It assumes that the first name is in B1 and the last name is in D1.

```
=IF(LEN(B1&D1)+2>=LEN(A1),",",MID(A1,LEN(B1)+2,LEN(A1)-LEN(B1&D1)-2))
```

As you can see in Figure 5-6, the formulas work fairly well. There are a few problems, however – notably names that contain four “words.” But, as I mentioned earlier, you can clean these cases up manually.



If you want to know how I created these complex formulas, refer to Chapter 20 for a discussion of megaformulas.

	A	B	C	D	E
1	Full Name	First	Middle	Last	
2	John Q. Public	John	Q.	Public	
3	Lisa Smith	Lisa		Smith	
4	J. R. Robins	J.	R.	Robins	
5	A.P. Adams	A.P.		Adams	
6	Roger Smith	Roger		Smith	
7	Mr. James Olson	Mr.	James	Olson	
8					
9					

Figure 5-6: This worksheet uses formulas to extract the first name, middle name (or initial), and last name from a list of names in column A.

Removing Titles from Names

You can use the formula that follows to remove three common titles (Mr., Ms., and Mrs.) from a name. For example, if cell A1 contains *Mr. Fred Munster*, the formula would return *Fred Munster*.

```
=IF(OR(LEFT(A1,2)="Mr",LEFT(A1,3)="Mrs",LEFT(A1,2)="Ms"),RIGHT(A1,LEN(A1)-FIND(" ",A1)),A1)
```

Counting the Number of Words in a Cell

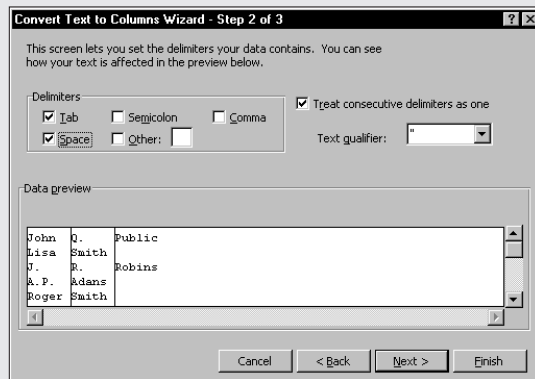
The following formula returns the number of words in cell A1:

```
=LEN(TRIM(A1))-LEN(SUBSTITUTE(TRIM(A1)," ",""))+1
```

The formula uses the TRIM function to remove excess spaces. It then uses the SUBSTITUTE function to create a new string (in memory) that has all the space characters removed. The length of this string is subtracted from the length of the original (trimmed) string to get the number of spaces. This value is then incremented by 1 to get the number of words.

Splitting Text Strings without Using Formulas

In many cases, you can eliminate the use of formulas and use Excel's Data → Text to Columns command to parse strings into their component parts. Selecting this command displays Excel's Convert Text to Columns Wizard, which consists of a series of dialog boxes that walk you through the steps to convert a single column of data into multiple columns. Generally, you'll want to select the Delimited option (in Step 1) and use Space as the delimiter (in Step 2).



Note that this formula will return 1 if the cell is empty. The following modification solves that problem:

```
=IF(LEN(A1)=0,0,LEN(TRIM(A1))-LEN(SUBSTITUTE(TRIM(A1)," ",""))+1)
```

Custom VBA Text Functions

Excel has many functions that work with text, but likely you'll run into a situation in which the appropriate function just doesn't exist. In such a case, you can often create your own worksheet function using VBA.



Chapter 25 contains several additional text functions, written in VBA. I briefly describe these functions here.

- ◆ REVERSETEXT: Returns the text in a cell backwards. For example, using *Evian* as the argument returns *naivE*.
- ◆ ACRONYM: Returns the first letter of each word in its argument. For example, using *Power Utility Pak* as the argument returns *PUP*.
- ◆ SPELLDOLLARS: Returns a number “spelled out” in text – as on a check. For example, using *123.45* as the argument returns *One hundred twenty-three and 45/100 dollars*.
- ◆ SCRAMBLE: Returns the contents of its argument randomized. For example, using *Microsoft* as the argument may return *oficMorts* – or some other random permutation.
- ◆ ISLIKE: Returns TRUE if a string matches a pattern composed of text and wildcard characters.
- ◆ CELLHASTEXT: Returns TRUE if the cell argument contains text, or a value formatted as Text. This function overcomes the problems described at the beginning of this chapter (see “Determining Whether a Cell Contains Text”).
- ◆ EXTRACTELEMENT: Extracts an element from a string based on a specified separator character (such as a hyphen).

Summary

This chapter provided some background on how Excel deals with text entered into cells. It also presented many useful examples that incorporate Excel’s text functions.

The next chapter presents formulas that enable you to calculate dates, times, and other time-period values.

Chapter 6

Working with Dates and Times

IN THIS CHAPTER

- ◆ An overview of using dates and times in Excel
- ◆ Excel's date-related functions
- ◆ Excel's time-related functions

BEGINNERS OFTEN FIND THAT working with dates and times in Excel can be frustrating. To eliminate this frustration, you'll need a good understanding of how Excel handles time-based information. This chapter provides the information you need to create powerful formulas that manipulate dates and times.



The dates in this chapter correspond to the United States English date format: month/day/year. For example, the date 3/1/1952 refers to March 1, 1952, not January 3, 1952. I realize that this is very illogical, but that's the way we Americans have been trained. I trust that the non-American readers of this book can make the adjustment.

How Excel Handles Dates and Times

This section presents a quick overview of how Excel deals with dates and times. It includes coverage of Excel's date and time serial number system, and offers tips for entering and formatting dates and times.



Other chapters in this book contain additional date-related information. For example, refer to Chapter 7 for counting examples that use dates. Chapter 25 contains some VBA functions that work with dates.

Understanding Date Serial Numbers

To Excel, a date is simply a number. More precisely, a date is a “serial number” that represents the number of days since January 0, 1900. A serial number of 1 corresponds to January 1, 1900; a serial number of 2 corresponds to January 2, 1900, and so on. This system makes it possible to deal with dates in formulas. For example, you can create a formula to calculate the number of days between two dates.

You may wonder about January 0, 1900. This “non-date” (which corresponds to date serial number 0) is actually used to represent times that are not associated with a particular day. This will become clear later in this chapter.

To view a date serial number as a date, you must format the cell as a date. Use the Format Cells dialog box (Number tab) to apply a date format.



Excel 97 and later versions support dates from January 1, 1900 through December 31, 9999 (serial number = 2,958,465). Previous versions of Excel support a much smaller range of dates: from January 1, 1900 through December 31, 2078 (serial number = 65,380).

Choose Your Date System: 1900 or 1904

Excel actually supports two date systems: the 1900 date system and the 1904 date system. Which system you use in a workbook determines what date serves as the basis for dates. The 1900 date system uses January 1, 1900 as the day assigned to date serial number 1. The 1904 date system uses January 1, 1904 as the base date. By default, Excel for Windows uses the 1900 date system, and Excel for Macintosh uses the 1904 date system. Excel for Windows supports the 1904 date system for compatibility with Macintosh files. You can choose the date system from the Options dialog box (select Tools → Options and select the Calculation tab). You cannot change the date system if you use Excel for Macintosh.

Generally, you should use the default 1900-date system. And you should exercise caution if you use two different date systems in workbooks that are linked together. For example, assume Book1 uses the 1904 date system and contains the date 1/15/1999 in cell A1. Assume Book2 uses the 1900 date system and contains a link to cell A1 in Book1. Book2 will display the date as 1/14/1995. Both workbooks will use the same date serial number (34713), but they will be interpreted differently.

One advantage to using the 1904 date system is that it enables you to display negative time values. With the 1900 date system, a calculation that results in a negative time (for example, 4:00 PM – 5:30 PM) cannot be displayed. When using the 1904 date system, the negative time displays as -1:30 (that is, a difference of one hour and 30 minutes).

Entering Dates

You can enter a date directly as a serial number (if you know it), but more often you'll enter a date using any of several recognized date formats. Excel automatically converts your entry into the corresponding date serial number (which it uses for calculations), and also applies the default date format to the cell so it displays as an actual date rather than a cryptic serial number.

For example, if you need to enter June 1, 2002, you can simply enter the date by typing **June 1, 2002** (or use any of several different date formats). Excel interprets your entry and stores the value 37408, the date serial number for that date. It also applies the default date format, so the cell contents may not appear exactly as you typed them.



Depending on your regional settings, entering a date in a format such as **June 1, 2002** may be interpreted as a text string. In such a case, you would need to enter the date in a format such as **1 June, 2002**.

When you activate a cell that contains a date, the formula bar shows the cell contents formatted using the default date format – which corresponds to your system's short date style. The formula bar does not display the date's serial number. If you need to find out the serial number for a particular date, format the cell using a non-date number format.



To change the default date format, you need to change a system-wide setting. Access the Windows Control Panel, and select Regional Settings. In the Regional Settings dialog box, select the Date tab. The selected item for the Short date style determines the default date format used by Excel.

Table 6-1 shows a sampling of the date formats that Excel recognizes (using the U.S. settings). Results will vary if you use a different regional setting.

TABLE 6-1 DATE ENTRY FORMATS RECOGNIZED BY EXCEL

Entry	Excel's Interpretation (U.S. Settings)
6-1-01	June 1, 2001
6-1-2001	June 1, 2001

Continued

TABLE 6-1 DATE ENTRY FORMATS RECOGNIZED BY EXCEL (Continued)

Entry	Excel's Interpretation (U.S. Settings)
6/1/01	June 1, 2001
6/1/2001	June 1, 2001
6-1/01	June 1, 2001
June 1, 2001	June 1, 2001
Jun 1	June 1 of the current year
June 1	June 1 of the current year
6/1	June 1 of the current year
6-1	June 1 of the current year
1-Jun-2001	June 1, 2001
2001/6/1	June 1, 2001

As you can see in Table 6-1, Excel is rather intelligent when it comes to recognizing dates entered into a cell. It's not perfect, however. For example, Excel does *not* recognize any of the following entries as dates:

- ◆ June 1 2001
- ◆ Jun-1 2001
- ◆ Jun-1/2001

Rather, it interprets these entries as text. If you plan to use dates in formulas, make sure that Excel can recognize the date you enter as a date; otherwise, the formulas that refer to these dates will produce incorrect results.

If you attempt to enter a date that lies outside of the supported date range, Excel interprets it as text. If you attempt to format a serial number that lies outside of the supported range as a date, the value displays as a series of hash marks (#####).

Understanding Time Serial Numbers

When you need to work with time values, you simply extend Excel's date serial number system to include decimals. In other words, Excel works with times by using fractional days. For example, the date serial number for June 1, 2001, is 37043. Noon (halfway through the day) is represented internally as 37043.5.

Searching for Dates

If your worksheet uses many dates, you may need to search for a particular date by using Excel's Find dialog box (which you can access with the Edit → Find command, or Ctrl+F). You'll find that Excel is rather picky when it comes to finding dates. You must enter a full four-digit date into the Find what field in the Find dialog box. The format must correspond to your system's short date format (this is the format that displays in the formula bar).

The serial number equivalent of one minute is approximately 0.00069444. The formula that follows calculates this number by multiplying 24 hours by 60 minutes, and dividing the result into 1. The denominator consists of the number of minutes in a day (1,440).

```
=1/(24*60)
```

Similarly, the serial number equivalent of one second is approximately 0.00001157, obtained by the following formula (1 divided by 24 hours times 60 minutes times 60 seconds). In this case, the denominator represents the number of seconds in a day (86,400).

```
=1/(24*60*60)
```

In Excel, the smallest unit of time is one one-thousandth of a second. The time serial number shown here represents 23:59:59.999, or one one-thousandth of a second before midnight:

```
0.99999999
```

Table 6-2 shows various times of day, along with each associated time serial number.

TABLE 6-2 TIMES OF DAY AND THEIR CORRESPONDING SERIAL NUMBERS

Time of Day	Time Serial Number
12:00:00 AM (midnight)	0.00000000
1:30:00 AM	0.06250000

Continued

TABLE 6-2 TIMES OF DAY AND THEIR CORRESPONDING SERIAL NUMBERS *(Continued)*

Time of Day	Time Serial Number
3:00:00 AM	0.12500000
4:30:00 AM	0.18750000
6:00:00 AM	0.25000000
7:30:00 AM	0.31250000
9:00:00 AM	0.37500000
10:30:00 AM	0.43750000
12:00:00 PM (noon)	0.50000000
1:30:00 PM	0.56250000
3:00:00 PM	0.62500000
4:30:00 PM	0.68750000
6:00:00 PM	0.75000000
7:30:00 PM	0.81250000
9:00:00 PM	0.87500000
10:30:00 PM	0.93750000

Entering Times

As with entering dates, you normally don't have to worry about the actual time serial numbers. Just enter the time into a cell using a recognized format. Table 6-3 shows some examples of time formats that Excel recognizes:

TABLE 6-3 TIME ENTRY FORMATS RECOGNIZED BY EXCEL

Entry	Excel's Interpretation
11:30:00 am	11:30 AM
11:30:00 AM	11:30 AM
11:30 pm	11:30 PM

Entry	Excel's Interpretation
11:30	11:30 AM
13:30	1:30 PM

Because the preceding samples don't have a specific day associated with them, Excel (by default) uses a date serial number of 0, which corresponds to the non-day January 0, 1900. Often, you'll want to combine a date and time. Do so by using a recognized date entry format, followed by a space, and then a recognized time-entry format. For example, if you enter the text that follows in a cell, Excel interprets it as 11:30 a.m. on June 1, 2001. Its date/time serial number is 37043.4791666667.

6/1/2001 11:30

When you enter a time that exceeds 24 hours, the associated date for the time increments accordingly. For example, if you enter the following time into a cell, it is interpreted as 1:00 AM on January 1, 1900. The day part of the entry increments because the time exceeds 24 hours.

25:00:00

Similarly, if you enter a date *and* a time (and the time exceeds 24 hours), the date that you entered is adjusted. The following entry, for example, is interpreted as 9/2/1999 1:00:00 AM.

9/1/1999 25:00:00

If you enter a time only (without an associated date), you'll find that the maximum time that you can enter into a cell is 9999:59:59 (just under 10,000 hours). Excel adds the appropriate number of days. In this case, 9999:59:59 is interpreted as 3:59:59 PM on 02/19/1901. If you enter a time that exceeds 10,000 hours, the time appears as a text string.

Formatting Dates and Times

You have a great deal of flexibility in formatting cells that contain dates and times. For example, you can format the cell to display the date part only, the time part only, or both the date and time parts.

You format dates and times by selecting the cells, and then using the Number tab of the Format Cells dialog box, shown in Figure 6-1. The Date category shows built-in date formats, and the Time category shows built-in time formats. Some of

the formats include both date and time displays. Just select the desired format from the Type list and click OK.

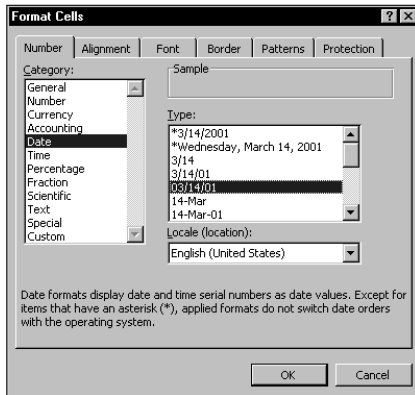


Figure 6-1: Use the Number tab in the Format Cells dialog box to change the appearance of dates and times.



When you create a formula that refers to a cell containing a date or a time, Excel automatically formats the formula cell as a date or a time. Sometimes, this is very helpful; other times, it's completely inappropriate and downright annoying. Unfortunately, you cannot turn off this automatic date formatting. You can, however, use a shortcut key combination to remove all number formatting from the cell and return to the default "General" format. Just select the cell and press **Ctrl+Shift+~**.

If none of the built-in formats meet your needs, you can create a custom number format. Select the Custom category, and then type the custom format codes into the Type box. (See Appendix C for information on creating custom number formats.)



A particularly useful custom number format for displaying times is:

[h]:mm:ss

Using square brackets around the hour part of the format string causes Excel to display hours beyond 24 hours. You will find this useful when adding times that exceed 24 hours. For an example, see "Summing Times That Exceed 24 Hours," later in this chapter.

Problems with Dates

Excel has some problems when it comes to dates. Many of these problems stem from the fact that Excel was designed many years ago, before the acronym Y2K became a household term. And, as I describe, the Excel designers basically emulated Lotus 1-2-3's limited date and time features, which contain a nasty bug duplicated intentionally in Excel. In addition, versions of Excel show inconsistency in how they interpret a cell entry that has a two-digit year. And finally, how Excel interprets a date entry depends on your regional date settings.

If Excel were being designed from scratch today, I'm sure it would be much more versatile in dealing with dates. Unfortunately, we're currently stuck with a product that leaves much to be desired in the area of dates.

EXCEL'S LEAP YEAR BUG

A leap year, which occurs every four years, contains an additional day (February 29). Although the year 1900 was not a leap year, Excel treats it as such. In other words, when you type the following into a cell, Excel does not complain. It interprets this as a valid date and assigns a serial number of 60:

```
2/29/1900
```

If you type the following invalid date, Excel correctly interprets it as a mistake and *doesn't* convert it to a date. Rather, it simply makes the cell entry a text string:

```
2/29/1901
```

How can a product used daily by millions of people contain such an obvious bug? The answer is historical. The original version of Lotus 1-2-3 contained a bug that caused it to consider 1900 as a leap year. When Excel was released some time later, the designers knew of this bug, and chose to reproduce it in Excel to maintain compatibility with Lotus worksheet files.

Why does this bug still exist in later versions of Excel? Microsoft asserts that the disadvantages of correcting this bug outweigh the advantages. If the bug were eliminated, it would mess up hundreds of thousands of existing workbooks. In addition, correcting this problem would affect compatibility between Excel and other programs that use dates. As it stands, this bug really causes very few problems because most users do not use dates before March 1, 1900.

PRE-1900 DATES

The world, of course, didn't begin on January 1, 1900. People who work with historical information using Excel often need to work with dates before January 1, 1900. Unfortunately, the only way to work with pre-1900 dates is to enter the date into a cell as text. For example, you can enter the following into a cell and Excel won't complain:

```
July 4, 1776
```


You can't, however, perform any manipulation on dates recognized as text. For example, you can't change its numeric formatting, you can't determine which day of the week this date occurred on, and you can't calculate the date that occurs seven days later.



The companion CD-ROM contains an add-in that I developed called Extended Date Functions. When you install this add-in, you'll have access to eight new worksheet functions that enable you to work with any date in the years 0100 through 9999. Figure 6-2 shows a worksheet that uses these functions in column D to perform calculations that involve pre-1900 dates.

	A	B	C	D	E
1	President	Born	Died	Age	
2	William McKinley	1/29/1843	9/14/1901	58	
3	Franklin D. Roosevelt	1/30/1882	4/12/1945	63	
4	William Henry Harrison	2/9/1773	4/4/1841	68	
5	Abraham Lincoln	2/12/1809	4/15/1865	56	
6	Zachary Taylor	3/29/1790	7/9/1850	60	
7	Warren G. Harding	11/2/1865	8/2/1923	57	
8	James A. Garfield	11/19/1831	9/19/1881	49	
9					

Figure 6-2: The Extended Date Functions add-in enables you to work with pre-1900 dates.

INCONSISTENT DATE ENTRIES

You need to exercise caution when entering dates by using two digits for the year. When you do so, Excel has some rules that kick in to determine which century to use. And those rules vary depending on the version of Excel that you use.

For Excel 97, two-digit years between 00 and 29 are interpreted as 21st century dates, and two-digit years between 30 and 99 are interpreted as 20th century dates. For example, if you enter 12/5/28, Excel interprets your entry as December 5, 2028. But if you enter 12/5/30, Excel sees it as December 5, 1930. If you use Excel 2000 or later (running on Windows 98 or later), you can use the default boundary year of 2029, or change it using the Windows Control Panel (use the Date tab of the Regional Settings Properties dialog box).

For previous versions of Excel (Excel 3 through Excel 95), two-digit years between 00 and 19 are interpreted as 21st century dates, and two-digit years between 20 and 99 are interpreted as 20th century dates. For example, if you enter 12/5/19, Excel interprets your entry as December 5, 2019. But if you enter 12/5/20, Excel sees it as December 5, 1920.

If, for some unknown reason, you still use Excel 2, when you enter a two-digit date, it is *always* interpreted as a 20th century date. Table 6-4 summarizes these differences for various versions of Excel.

**TABLE 6-4 HOW TWO-DIGIT YEARS ARE INTERPRETED
IN VARIOUS EXCEL VERSIONS**

Excel Version	20th Century Years	21st Century Years
2	00–99	N/A
3, 4, 5, 7 (95)	20–99	00–19
8 (97), 9 (2000), 10 (2002)	30–99	00–29

To avoid any surprises, you should simply enter *all* years using all four digits for the year.

Date-Related Functions

Excel has quite a few functions that work with dates. When you use the Insert Function dialog box, these functions appear in the Date & Time function category.

Table 6-5 summarizes the date-related functions available in Excel. Some of Excel's date functions require that you install the Analysis ToolPak.

TABLE 6-5 DATE-RELATED FUNCTIONS

Function	Description
DATE	Returns the serial number of a particular date
DATEDIF	Calculates the number of days, months, or years between two dates
DATEVALUE	Converts a date in the form of text to a serial number
DAY	Converts a serial number to a day of the month
DAYS360	Calculates the number of days between two dates based on a 360-day year
EDATE*	Returns the serial number of the date that represents the indicated number of months before or after the start date
EOMONTH*	Returns the serial number of the last day of the month before or after a specified number of months

Continued

TABLE 6-5 DATE-RELATED FUNCTIONS *(Continued)*

Function	Description
MONTH	Converts a serial number to a month
NETWORKDAYS*	Returns the number of whole workdays between two dates
NOW	Returns the serial number of the current date and time
TODAY	Returns the serial number of today's date
WEEKDAY	Converts a serial number to a day of the week
WEEKNUM*	Returns the week number in the year
WORKDAY*	Returns the serial number of the date before or after a specified number of workdays
YEAR	Converts a serial number to a year
YEARFRAC*	Returns the year fraction representing the number of whole days between start_date and end_date

**Function is available only when the Analysis ToolPak add-in is installed.*

Displaying the Current Date

The following function displays the current date in a cell:

```
=TODAY()
```

You can also display the date, combined with text. The formula that follows, for example, displays text such as *Today is Monday, April 9, 2001*.

```
="Today is "&TEXT(TODAY(),"dddd, mmmm d, yyyy")
```

It's important to understand that the TODAY function is updated whenever the worksheet is calculated. For example, if you enter either of the preceding formulas into a worksheet, they will display the current date. But when you open the workbook tomorrow, they will display the current date (not the date when you entered the formula).



To enter a “date stamp” into a cell, press Ctrl+; (semicolon). This enters the date directly into the cell and does not use a formula. Therefore, the date will not change.

Displaying Any Date

As explained earlier in this chapter, you can easily enter a date into a cell by simply typing it, using any of the date formats that Excel recognizes. You can also create a date by using the DATE function, which takes three arguments: the year, the month, and the day. The following formula, for example, returns a date comprised of the year in cell A1, the month in cell B1, and the day in cell C1:

```
=DATE(A1,B1,C1)
```



The DATE function accepts invalid arguments, and adjusts the result accordingly. For example, this next formula uses 13 as the month argument, and returns January 1, 2002. The month argument is automatically translated as month 1 of the following year.

```
=DATE(2001,13,1)
```

Often, you’ll use the DATE function with other functions as arguments. For example, the formula that follows uses the YEAR and TODAY functions to return the date for Independence Day (July 4th) of the current year:

```
=DATE(YEAR(TODAY()),7,4)
```

The DATEVALUE function converts a text string that looks like a date into a date serial number. The following formula returns 37490, the date serial number for August 22, 2002:

```
=DATEVALUE("8/22/2002")
```

To view the result of this formula as a date, you need to apply a date number format to the cell.



Be careful when using the DATEVALUE function. A text string that “looks like a date” in your country, may not look like a date in another country. The preceding example works fine if your system is set for U.S. date formats, but it returns an error for other regional date formats because Excel is looking for the eighth day of the 22nd month!

Generating a Series of Dates

Often, you’ll want to insert a series of dates into a worksheet. For example, in tracking weekly sales, you may want to enter a series of dates, each separated by seven days. These dates will serve to identify the sales figures.

The most efficient way to enter a series of dates doesn’t require any formulas. Use Excel’s AutoFill feature to insert a series of dates. Enter the first date, and drag the cell’s fill handle while pressing the right mouse button. Release the mouse button and select an option from the shortcut menu (see Figure 6-3).

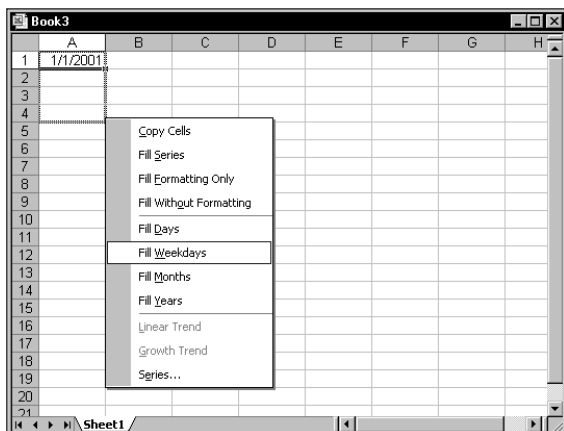


Figure 6-3: Using Excel’s AutoFill feature to create a series of dates

The advantage of using formulas to create a series of dates is that you can change the first date and the others will update automatically. You need to enter the starting date into a cell, and then use formulas (copied down the column) to generate the additional dates.

The following examples assume that you entered the first date of the series into cell A1, and the formula into cell A2. You can then copy this formula down the column as many times as needed.

To generate a series of dates separated by seven days, use this formula:

```
=A1+7
```

To generate a series of dates separated by one month, use this formula:

```
=DATE(YEAR(A1),MONTH(A1)+1,DAY(A1))
```

To generate a series of dates separated by one year, use this formula:

```
=DATE(YEAR(A1)+1,MONTH(A1),DAY(A1))
```

To generate a series of weekdays only (no Saturdays or Sundays), use the formula that follows. This formula assumes that the date in cell A1 is not a weekend.

```
=IF(WEEKDAY(A1)=6,A1+3,A1+1)
```

Converting a Non-Date String to a Date

You may import data that contains dates coded as text strings. For example, the following text represents August 21, 2001 (a four-digit year followed by a two-digit month, followed by a two-digit day):

```
20010821
```

To convert this string to an actual date, you can use a formula such as this one (it assumes the coded data is in cell A1):

```
=DATE(LEFT(A1,4),MID(A1,5,2),RIGHT(A1,2))
```

This formula uses text functions (LEFT, MID, and RIGHT) to extract the digits, and then uses these extracted digits as arguments for the DATE function.



Refer to Chapter 5 for more information about using formulas to manipulate text.

Calculating the Number of Days between Two Dates

A common type of date calculation determines the number of days between two dates. For example, you may have a financial worksheet that calculates interest

earned on a deposit account. The interest earned depends on the number of days the account is open. If your sheet contains the open date and the close date for the account, you can calculate the number of days the account was open.

Because dates store as consecutive serial numbers, you can use simple subtraction to calculate the number of days between two dates. For example, if cells A1 and B1 both contain a date, the following formula returns the number of days between these dates:

```
=A1-B1
```

Excel will automatically format this formula cell as a date, rather than a numeric value. Therefore, you will need to change the number format so the result is displayed as a non-date. If cell B1 contains a more recent date than the date in cell A1, the result will be negative.



If this formula does not display the correct value, make sure that A1 and B1 both contain actual dates — not text that *looks* like a date.

Sometimes, calculating the difference between two days is more difficult. To demonstrate, consider the common “fence-post” analogy. If somebody asks you how many units make up a fence, you can respond with either of two answers: the number of fence posts, or the number of gaps between the fence posts. The number of fence posts is always one more than the number of gaps between the posts.

To bring this analogy into the realm of dates, suppose you start a sales promotion on February 1, and end the promotion on February 9. How many days was the promotion in effect? Subtracting February 1 from February 9 produces an answer of eight days. Actually, the promotion lasted nine days. In this case, the correct answer involves counting the fence posts, not the gaps. The formula to calculate the length of the promotion (assuming you have appropriately named cells) appears like this:

```
=EndDay-StartDay+1
```

Calculating the Number of Work Days between Two Dates

When calculating the difference between two dates, you may want to exclude weekends and holidays. For example, you may need to know how many business days fall in the month of November. This calculation should exclude Saturdays, Sundays, and holidays. The NETWORKDAYS function can help out. (You can access this function only when you install the Analysis ToolPak.)



The NETWORKDAYS function has a very misleading name. This function has nothing to do with networks or networking. Rather, it calculates the net workdays between two dates.

The NETWORKDAYS function calculates the difference between two dates, excluding weekend days (Saturdays and Sundays). As an option, you can specify a range of cells that contain the dates of holidays, which are also excluded. Excel has absolutely no way of determining which days are holidays, so you must provide this information in a range.

Figure 6-4 shows a worksheet that calculates the workdays between two dates. The range A2:A11 contains a list of holiday dates. The formulas in column C calculate the workdays between the dates in column A and column B. For example, the formula in cell C15 is:

```
=NETWORKDAYS(A15,B15,A2:A11)
```

	A	B	C
1	Date	Holiday	
2	1/1/2002	New Year's Day	
3	1/21/2002	Martin Luther King Jr. Day	
4	2/18/2002	Presidents' Day	
5	5/27/2002	Memorial Day	
6	7/4/2002	Independence Day	
7	9/2/2002	Labor Day	
8	11/11/2002	Veterans Day	
9	10/14/2002	Columbus Day	
10	11/28/2002	Thanksgiving Day	
11	12/25/2002	Christmas Day	
12			
13			
14	First Day	Last Day	Working Days
15	Tuesday 1/1/2002	Monday 1/7/2002	4
16	Tuesday 1/1/2002	Tuesday 12/31/2002	251
17			

Figure 6-4: Using the NETWORKDAYS function to calculate the number of working days between two dates

This formula returns 4, which means that the seven-day period beginning with January 1 contains four workdays. In other words, the calculation excludes one holiday, one Saturday, and one Sunday. The formula in cell C16 calculates the total number of work days in the year.



This workbook is available on the companion CD-ROM.

Offsetting a Date Using Only Work Days

The `WORKDAY` function, which is available only when you install the Analysis ToolPak, is the opposite of the `NETWORKDAYS` function. For example, if you start a project on January 4, and the project requires 10 working days to complete, the `WORKDAY` function can calculate the date you will finish the project.

The following formula uses the `WORKDAY` function to determine the date 10 working days from January 4, 2001. A working day consists of a weekday (Monday through Friday).

```
=WORKDAY("1/4/2001",10)
```

The formula returns January 18, 2001 (four weekend dates fall between January 4 and January 18).



The preceding formula may return a different result, depending on your regional date setting (the hard-coded date may be interpreted as April 1, 2001). A better formula is:

```
=WORKDAY(DATE(2001,1,4),10)
```

The second argument for the `WORKDAY` function can be negative. And, as with the `NETWORKDAYS` function, the `WORKDAY` function accepts an optional third argument (a reference to a range that contains a list of holiday dates).

Calculating the Number of Years between Two Dates

The following formula calculates the number of years between two dates. This formula assumes that cells A1 and B1 both contain dates:

```
=YEAR(A1)-YEAR(B1)
```

This formula uses the `YEAR` function to extract the year from each date, and then subtracts one year from the other. If cell B1 contains a more recent date than the date in cell A1, the result will be negative.

Note that this function doesn't calculate *full* years. For example, if cell A1 contains 12/31/2001 and cell B1 contains 01/01/2002, the formula returns a difference of one year, even though the dates differ by only one day.

Calculating a Person's Age

A person's age indicates the number of full years that the person has been alive. The formula in the previous section (for calculating the number of years between

two dates) won't calculate this value correctly. You can use two other formulas, however, to calculate a person's age.

The following formula returns the age of the person whose date of birth you enter into cell A1. This formula uses the YEARFRAC function, which is available only when you install the Analysis ToolPak add-in.

```
=INT(YEARFRAC(TODAY(),A1,1))
```

The following formula, which doesn't rely on an Analysis ToolPak function, uses the DATEDIF function to calculate an age (see the sidebar, "Where's the DATEDIF Function?"):

```
=DATEDIF(A1,TODAY(),"Y")
```

If you're a stickler for detail, use the following formula to calculate the exact age in years, months, and days.

```
=DATEDIF(A1,NOW(),"y") & " years, " & DATEDIF(A1,NOW(),"ym") & " months, " & DATEDIF(A1,NOW(),"md") & " days"
```

This formula will return a text string such as *49 years, 3 months, 29 days*.

Determining the Day of the Year

January 1 is the first day of the year, and December 31 is the last day. But what about all of those days in between? The following formula returns the day of the year for a date stored in cell A1:

```
=A1-DATE(YEAR(A1),1,0)
```

The day of the year is sometimes referred to as a *Julian date*.

The following formula returns the number of days remaining in the year from a particular date (assumed to be in cell A1):

```
=DATE(YEAR(A1),12,31)-A1
```

When you enter either of these formulas, Excel applies date formatting to the cell. You need to apply a non-date number format to view the result as a number.

To convert a particular day of the year (for example, the 90th day of the year) to an actual date in a specified year, use the formula that follows. This formula assumes the year is stored in cell A1, and the day of the year is stored in cell B1.

```
=DATE(A1,1,B1)
```

Where's the DATEDIF Function?

In several places throughout this chapter, I refer to the DATEDIF function. You may notice that this function does not appear in the Paste Function dialog box. Therefore, when you use this function, you must always enter it manually.

The DATEDIF function has its origins in Lotus 1-2-3, and apparently Excel provides it for compatibility purposes. For some reason, Microsoft wants to keep this function a secret. Versions prior to Excel 2000 failed to even mention the DATEDIF function in the online help. Interestingly, references to this function were removed from the online help for Excel 2002 (although the function is still available).

DATEDIF is a handy function that calculates the number of days, months, or years between two dates. The function takes three arguments: `start_date`, `end_date`, and a code that represents the time unit of interest. The following table displays valid codes for the third argument (you must enclose the codes in quotation marks).

Unit Code	Returns
"y"	The number of complete years in the period.
"m"	The number of complete months in the period.
"d"	The number of days in the period.
"md"	The difference between the days in <code>start_date</code> and <code>end_date</code> . The months and years of the dates are ignored.
"ym"	The difference between the months in <code>start_date</code> and <code>end_date</code> . The days and years of the dates are ignored.
"yd"	The difference between the days of <code>start_date</code> and <code>end_date</code> . The years of the dates are ignored.

The `start_date` argument must be earlier than the `end_date` argument, or the function returns an error.

Determining the Day of the Week

The WEEKDAY function accepts a date argument, and returns an integer between 1 and 7 that corresponds to the day of the week. The following formula, for example, returns 3 because the first day of the year 2002 falls on a Tuesday:

```
=WEEKDAY(DATE(2002,1,1))
```


Determining the Date of the Most Recent Sunday

You can use the following formula to return the date for the previous Sunday. If the current day is a Sunday, the formula returns the current date:

```
=TODAY()-MOD(TODAY()-1,7)
```

To modify this formula to find the date of a day other than Sunday, change the 1 to a different number between 2 (for Monday) and 7 (for Saturday).

Determining the First Day of the Week after a Date

This next formula returns the specified day of the week that occurs after a particular date. For example, use this formula to determine the date of the first Monday after June 1, 2001. The formula assumes that cell A1 contains a date, and cell A2 contains a number between 1 and 7 (1 for Sunday, 2 for Monday, and so on).

```
=A1+A2-WEEKDAY(A1)+(A2<WEEKDAY(A1))*7
```

If cell A1 contains June 1, 2001 and cell A2 contains 2 (for Monday), the formula returns June 4, 2001. This is the first Monday after June 1, 2001 (which is a Friday).

Determining the nth Occurrence of a Day of the Week in a Month

You may need a formula to determine the date for a particular occurrence of a weekday. For example, suppose your company payday falls on the second Friday of each month, and you need to determine the paydays for each month of the year. The following formula will make this type of calculation:

```
=DATE(A1,A2,1)+A3-WEEKDAY(DATE(A1,A2,1))+  
(A4-(A3>=WEEKDAY(DATE(A1,A2,1))))*7
```

The formula in this section assumes:

- ◆ Cell A1 contains a year
- ◆ Cell A2 contains a month
- ◆ Cell A3 contains a day number (1 for Sunday, 2 for Monday, etc.)
- ◆ Cell A4 contains the occurrence number (for example, 2 to select the second occurrence of the weekday specified in cell A3)

If you use this formula to determine the date of the first Friday in June 2002, it returns June 7, 2002.



If the value in cell A4 exceeds the number of the specified day in the month, the formula returns a date from a subsequent month. For example, if you attempt to determine the date of the fifth Friday in June 2002 (there is no such date), the formula returns the first Friday in July.

Counting the Occurrences of a Day of the Week

You can use the following formula to count the number of occurrences of a particular day of the week for a specified month. It assumes that cell A1 contains a date, and cell B1 contains a day number (1 for Sunday, 2 for Monday, etc.). The formula is an array formula, so you must enter it using Ctrl+Shift+Enter.

```
{=SUM((WEEKDAY(DATE(YEAR(A1),MONTH(A1),ROW(INDIRECT("1:"&
DAY(DATE(YEAR(A1),MONTH(A1)+1,0))))))=B1)*1)}
```

If cell A1 contains the date January 5, 2002, and cell B1 contains the value 3 (for Tuesday), the formula returns 5, which reveals that January 2002 contains five Tuesdays.

The preceding array formula calculates the year and month by using the YEAR and MONTH functions. You can simplify the formula a bit if you store the year and month in separate cells. The following formula (also an array formula) assumes that the year appears in cell A1, the month in cell A2, and the day number in cell B1:

```
{=SUM((WEEKDAY(DATE(A1,A2,ROW(INDIRECT("1:"&
DAY(DATE(A1,A2+1,0))))))=B1)*1)}
```

Figure 6-5 shows this formula used in a worksheet. In this case, the formula uses mixed cell references so you can copy it. For example, the formula in cell C3 is:

```
{=SUM((WEEKDAY(DATE($B$2,$A3,ROW(INDIRECT("1:"&
DAY(DATE($B$2,$A3+1,0))))))=C$1)*1)}
```

Additional formulas use the SUM function to calculate the number of days per month (column J) and the number of each weekday in the year (row 15).



The workbook shown in Figure 6-5 is available on the companion CD-ROM.

	A	B	C	D	E	F	G	H	I	J
1			1	2	3	4	5	6	7	
2	YEAR ->	2002	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Month
3	1	January	4	4	5	5	5	4	4	31
4	2	February	4	4	4	4	4	4	4	28
5	3	March	5	4	4	4	4	5	5	31
6	4	April	4	5	5	4	4	4	4	30
7	5	May	4	4	4	5	5	5	4	31
8	6	June	5	4	4	4	4	4	5	30
9	7	July	4	5	5	5	4	4	4	31
10	8	August	4	4	4	4	5	5	5	31
11	9	September	5	5	4	4	4	4	4	30
12	10	October	4	4	5	5	5	4	4	31
13	11	November	4	4	4	4	4	5	5	30
14	12	December	5	5	5	4	4	4	4	31
15		Total Days:	52	52	53	52	52	52	52	365

Figure 6-5: Calculating the number of each weekday in each month of a year

Expressing a Date as an Ordinal Number

You may want to express the day portion of a date as an ordinal number. For example, you can display 4/6/2000 as April 6th, 2000. The following formula expresses the date in cell A1 as an ordinal date:

```
=TEXT(A1,"mmm " )&DAY(A1)&IF(INT(MOD(DAY(A1),100)/10)=1,
"th",IF(MOD(DAY(A1),10)=1,
"st",IF(MOD(DAY(A1),10)=2,"nd",IF(MOD(DAY(A1),10)=3,
"rd","th"))))&TEXT(A1," , yyyy")
```



The result of this formula is text, not an actual date.

The following formula shows a variation that expresses the date in cell A1 in day-month-year format. For example, 4/6/2000 would appear as 4th April, 2000. Again, the result of this formula represents text, not an actual date.

```
=DAY(A1)&IF(INT(MOD(DAY(A1),100)/10)=1, "th", IF(MOD(DAY(A1),10)=1,
"st",IF(MOD(DAY(A1),10)=2,"nd", IF(MOD(DAY(A1),10)=3, "rd","th"))))&
" " &TEXT(A1,"mmm , yyyy")
```



The companion CD-ROM contains a workbook that demonstrates the formulas for expressing dates as ordinal numbers.


Calculating Dates of Holidays

Determining the date for a particular holiday can be tricky. Some, such as New Year's Day and U.S. Independence Day, are no-brainers, because they always occur on the same date. For these kinds of holidays, you can simply use the DATE function, which I covered earlier in this chapter. To enter New Year's Day (which always falls on January 1) for a specific year in cell A1, you can enter this function:

```
=DATE(A1,1,1)
```

Other holidays are defined in terms of a particular occurrence of a particular weekday in a particular month. For example, Labor Day falls on the first Monday in September.

Figure 6-6 shows a workbook with formulas to calculate the date for 10 U.S. holidays. The formulas reference the year in cell A1. Notice that because New Year's Day, Independence Day, Veterans Day, and Christmas Day all fall on the same days of the year, the DATE function calculates their dates.



Holiday	Description	Date	Weekday
New Year's Day	1st Day in January	January 1, 2002	Tuesday
Martin Luther King Jr. Day	3rd Monday in January	January 21, 2002	Monday
Presidents' Day	3rd Monday in February	February 18, 2002	Monday
Memorial Day	Last Monday in May	May 27, 2002	Monday
Independence Day	4th Day of July	July 4, 2002	Thursday
Labor Day	1st Monday in September	September 2, 2002	Monday
Veterans Day	11th Day of November	November 11, 2002	Monday
Columbus Day	2nd Monday in October	October 14, 2002	Monday
Thanksgiving Day	4th Thursday in November	November 28, 2002	Thursday
Christmas Day	25th Day of December	December 25, 2002	Wednesday

Figure 6-6: Using formulas to determine the date for various holidays



The workbook shown in Figure 6-6 also appears on the companion CD-ROM.

MARTIN LUTHER KING JR. DAY

This holiday occurs on the third Monday in January. This formula calculates Martin Luther King Jr. Day for the year in cell A1:

```
=DATE(A1,1,1)+IF(2<WEEKDAY(DATE(A1,1,1)),7-WEEKDAY(DATE(A1,1,1))+2,2-WEEKDAY(DATE(A1,1,1)))+((3-1)*7)
```


PRESIDENTS' DAY

Presidents' Day occurs on the third Monday in February. This formula calculates Presidents' Day for the year in cell A1:

```
=DATE(A1,2,1)+IF(2<WEEKDAY(DATE(A1,2,1)),7-WEEKDAY  
(DATE(A1,2,1))+2,2-WEEKDAY(DATE(A1,2,1)))+((3-1)*7)
```

MEMORIAL DAY

The last Monday in May is Memorial Day. This formula calculates Memorial Day for the year in cell A1:

```
=DATE(A1,6,1)+IF(2<WEEKDAY(DATE(A1,6,1)),7-WEEKDAY  
(DATE(A1,6,1))+2,2-WEEKDAY(DATE(A1,6,1)))+((1-1)*7)-7
```

Notice that this formula actually calculates the first Monday in June, and then subtracts 7 from the result to return the last Monday in May.

LABOR DAY

Labor Day occurs on the first Monday in September. This formula calculates Labor Day for the year in cell A1:

```
=DATE(A1,9,1)+IF(2<WEEKDAY(DATE(A1,9,1)),7-WEEKDAY  
(DATE(A1,9,1))+2,2-WEEKDAY(DATE(A1,9,1)))+((1-1)*7)
```

COLUMBUS DAY

This holiday occurs on the second Monday in October. This formula calculates Columbus Day for the year in cell A1:

```
=DATE(A1,10,1)+IF(2<WEEKDAY(DATE(A1,10,1)),7-WEEKDAY  
(DATE(A1,10,1))+2,2-WEEKDAY(DATE(A1,10,1)))+((2-1)*7)
```

THANKSGIVING DAY

Thanksgiving Day is celebrated on the fourth Thursday in November. This formula calculates Thanksgiving Day for the year in cell A1:

```
=DATE(A1,11,1)+IF(5<WEEKDAY(DATE(A1,11,1)),7-WEEKDAY  
(DATE(A1,11,1))+5,5-WEEKDAY(DATE(A1,11,1)))+((4-1)*7)
```

Calculating Easter

You'll notice that I omitted Easter from the previous section. Easter is an unusual holiday because its date is determined based on the phase of the moon and not by the calendar. Because of this, determining when Easter occurs proves a bit of a challenge.

Hans Herber, an Excel master in Germany, once sponsored an Easter formula contest at his Web site. The goal was to create the shortest formula possible that correctly determined the date of Easter for the years 1900 through 2078.

Twenty formulas were submitted, ranging in length from 44 characters up to 154 characters. Some of these formulas, however, work only with European date settings. The following formula, submitted by Thomas Jansen, is the shortest formula that works with any date setting. This formula returns the date for Easter, and assumes the year is stored in cell A1:

```
=DOLLAR(( "4/"&A1)/7+MOD(19*MOD(A1,19)-7,30)*14%,)*7-6
```

Please don't ask me to explain this formula. I haven't a clue!

Determining the Last Day of a Month

To determine the date that corresponds to the last day of a month, you can use the DATE function. However, you need to increment the month by 1, and use a day value of 0. In other words, the "0th" day of the next month is the last day of the current month.

The following formula assumes that a date is stored in cell A1. The formula returns the date that corresponds to the last day of the month.

```
=DATE(YEAR(A1),MONTH(A1)+1,0)
```

You can use a variation of this formula to determine how many days comprise a specified month. The formula that follows returns an integer that corresponds to the number of days in the month for the date in cell A1:

```
=DAY(DATE(YEAR(A1),MONTH(A1)+1,0))
```

Determining Whether a Year Is a Leap Year

To determine whether a particular year is a leap year, you can write a formula that determines whether the 29th day of February occurs in February or March. You can take advantage of the fact that Excel's DATE function adjusts the result when you supply an invalid argument—for example, a day of 29 when February contains only 28 days.

The following formula returns TRUE if the year of the date in cell A1 is a leap year. Otherwise, it returns FALSE.

```
=IF(MONTH(DATE(YEAR(A1),2,29))=2,TRUE,FALSE)
```



This function returns the wrong result (TRUE) if the year is 1900. See “Excel’s Leap Year Bug,” earlier in this chapter.

Determining a Date’s Quarter

For financial reports, you might find it useful to present information in terms of quarters. The following formula returns an integer between 1 and 4 that corresponds to the calendar quarter for the date in cell A1:

```
=ROUNDUP(MONTH(A1)/3,0)
```

This formula divides the month number by 3, and then rounds up the result.

Converting a Year to Roman Numerals

Fans of old movies will like this one. The following formula converts the year 1945 to Roman numerals. It returns MCMXLV.

```
=ROMAN(1945)
```

You can access the ROMAN function once you install the Analysis ToolPak. This function returns a text string, so you can’t perform any calculations using the result! Unfortunately, Excel doesn’t provide a function to convert Roman numerals back to normal numbers.

Creating a Calendar in a Range

The example calendar you see in Figure 6-7 uses a single formula (an array formula) to display a calendar in a range of cells. The scroll bars are linked to cells that contain the month and year. The month is stored in cell B2 (named *m*) and the year is stored in cell D2 (named *y*). Enter the following array formula into the range B6:H11:

```
{=IF(MONTH(DATE(y,m,1))>MONTH(DATE(y,m,1)-(WEEKDAY
(DATE(y,m,1))-1)+{0;1;2;3;4;5}*7+{1,2,3,4,5,6,7}-1),
" ",DATE(y,m,1)-(WEEKDAY(DATE(y,m,1))-1)+{0;1;2;3;4;5}
*7+{1,2,3,4,5,6,7}-1)}
```



Figure 6-7: You can generate this calendar by using a single array formula, entered into 42 cells.



You can access the workbook shown in Figure 6-7 on the companion CD-ROM.

Time-Related Functions

Excel, as you might expect, also includes a number of functions that enable you to work with time values in your formulas. This section contains examples that demonstrate the use of these functions.

Table 6-6 summarizes the time-related functions available in Excel. When you use the Paste Function dialog box, these functions appear in the Date & Time function category.

TABLE 6-6 TIME-RELATED FUNCTIONS

Function	Description
HOUR	Converts a serial number to an hour
MINUTE	Converts a serial number to a minute
MONTH	Converts a serial number to a month
NOW	Returns the serial number of the current date and time
SECOND	Converts a serial number to a second
TIME	Returns the serial number of a particular time
TIMEVALUE	Converts a time in the form of text to a serial number

Displaying the Current Time

This formula displays the current time as a time serial number (or, a serial number without an associated date):

```
=NOW()-TODAY()
```



To enter a time stamp into a cell, press Ctrl+Shift+: (colon).

You need to format the cell with a time format to view the result as a recognizable time. For example, you can apply the following number format:

```
hh:mm AM/PM
```

You can also display the time, combined with text. The formula that follows displays the text, “The current time is 6:28 PM”.

```
= "The current time is "&TEXT(NOW(),"h:mm AM/PM")
```



These formulas are updated only when the worksheet is calculated.

Displaying Any Time

Earlier in this chapter, I described how to enter a time value into a cell: Just type it into a cell, making sure that you include at least one colon (:). You can also create a time by using the TIME function. For example, the following formula returns a time comprised of the hour in cell A1, the minute in cell B1, and the second in cell C1:

```
=TIME(A1,B1,C1)
```

Like the DATE function, the TIME function accepts invalid arguments and adjusts the result accordingly. For example, the following formula uses 80 as the minute argument, and returns 10:20:15 AM. The 80 minutes are simply added to the hour, with 20 minutes remaining.

```
=TIME(9,80,15)
```



If you enter a value greater than 24 as the first argument for the TIME function, the result may not be what you expect. Logically, a formula such as the one that follows should produce a date/time serial number of 1.041667 (that is, one day and one hour).

```
=TIME(25,0,0)
```

In fact, this formula is equivalent to the following:

```
=TIME(1,0,0)
```

You can also use the DATE function along with the TIME function in a single cell. The formula that follows generates 37229.7708333333, the serial number that represents 6:30 PM on December 4, 2001:

```
=DATE(2001,12,4)+TIME(18,30,0)
```

The TIMEVALUE function converts a text string that looks like a time into a time serial number. This formula returns 0.2395833333, the time serial number for 5:45 AM:

```
=TIMEVALUE("5:45 am")
```

To view the result of this formula as a time, you need to apply number formatting to the cell. The TIMEVALUE function doesn't recognize all common time formats. For example, the following formula returns an error because Excel doesn't like the periods in "a.m.":

```
=TIMEVALUE("5:45 a.m.")
```

Summing Times That Exceed 24 Hours

Many people are surprised to discover that, when you sum a series of times that exceed 24 hours, Excel doesn't display the correct total. Figure 6-8 shows an example. The range B2:B8 contains times that represent the hours and minutes worked each day. The formula in cell B9 is:

```
=SUM(B2:B8)
```

As you can see, the formula returns a seemingly incorrect total (18 hours, 30 minutes). The total should read 42 hours, 30 minutes. The problem is that the formula is really displaying a date/time serial number of 1.770833, but the cell formatting is not displaying the "date" part of the date/time.

Day	Hours Worked
Sunday	0
Monday	8:30
Tuesday	8:00
Wednesday	9:00
Thursday	9:30
Friday	4:30
Saturday	3:00
Total for Week:	18:30

Figure 6-8: Using the SUM function to add a series of times. The answer is incorrect because cell B9 has the wrong number format.

To view a time that exceeds 24 hours, you need to change the number format for the cell so square brackets surround the *hour* part of the format string. Applying the number format here to cell B9 displays the sum correctly:

```
[h]:mm
```

Figure 6-9 shows another example of a worksheet that manipulates times. This worksheet keeps track of hours worked during a week (regular hours and overtime hours).

Weekday	Date	Start Work	Time Out (Lunch)	Time In (Lunch)	End Work	Total Hours	Weekly Hours
Monday	3/4/2002	8:00 AM	12:00 PM	1:00 PM	6:00 PM	9:00	9:00
Tuesday	3/5/2002	10:00 AM	2:00 PM	2:30 PM	7:00 PM	8:30	17:30
Wednesday	3/6/2002	9:00 AM	12:00 PM	1:00 PM	6:30 PM	8:30	26:00
Thursday	3/7/2002	7:30 AM	11:30 AM	12:00 PM	6:00 PM	10:00	36:00
Friday	3/8/2002	9:30 AM	2:00 PM	3:30 PM	5:00 PM	6:00	42:00
Saturday	3/9/2002					0:00	42:00
Sunday	3/10/2002					0:00	42:00

WEEKLY TOTAL	
Total hours:	42:00
Regular hours:	40:00
Overtime hours:	2:00

Figure 6-9: An employee timesheet workbook

The week's starting date appears in cell D5, and the formulas in column B fill in the dates for the days of the week. Times appear in the range D8:G14, and formulas in column H calculate the number of hours worked each day. For example, the formula in cell H8 is:

```
=IF(E8<D8,E8+1-D8,E8-D8)+IF(G8<F8,G8+1-G8,G8-F8)
```

The first part of this formula subtracts the time in column D from the time in column E to get the total hours worked before lunch. The second part subtracts the time in column F from the time in column G to get the total hours worked after lunch. I use IF functions to accommodate graveyard shift cases that span midnight – for example, an employee may start work at 10:00 PM and begin lunch at 2:00 AM. Without the IF function, the formula returns a negative result.

The following formula in cell H17 calculates the weekly total by summing the daily totals in column H:

```
=SUM(H8:H14)
```


This worksheet assumes that hours that exceed 40 hours in a week are considered overtime hours. The worksheet contains a cell named *Overtime* (not shown in Figure 6-9). This cell contains the following formula:

```
=1+TIME(16,0,0)
```

This formula returns 40:00 (that is, 24 hours plus 16 hours). If your standard workweek consists of something other than 40 hours, you can change this formula.

The following formula (in cell H18) calculates regular (non-overtime) hours. This formula returns the smaller of two values: the total hours, or the overtime hours.

```
=MIN(E17,Overtime)
```

The final formula, in cell H19, simply subtracts the regular hours from the total hours to yield the overtime hours.

```
=E17-E18
```

The times in H17:H19 may display time values that exceed 24 hours, so these cells use a custom number format:

```
[h]:mm
```



The workbook shown in Figure 6-9 also appears on the companion CD-ROM.

Calculating the Difference between Two Times

Because times are represented as serial numbers, you can subtract the earlier time from the later time to get the difference. For example, if cell A2 contains 5:30:00 and cell B2 contains 14:00:00, the following formula returns 08:30:00 (a difference of eight hours and 30 minutes):

```
=B2-A2
```

If the subtraction results in a negative value, however, it becomes an invalid time; Excel displays a series of hash marks (#####) because a time without a

date has a date serial number of 0. A negative time results in a negative serial number, which is not permitted.

If the direction of the time difference doesn't matter, you can use the ABS function to return the absolute value of the difference:

```
=ABS(B2-A2)
```

This “negative time” problem often occurs when calculating an elapsed time—for example, calculating the number of hours worked given a start time and an end time. This presents no problem if the two times fall in the same day. But if the work shift spans midnight, the result is an invalid negative time. For example, you may start work at 10:00 PM and end work at 6:00 AM the next day. Figure 6-10 shows a worksheet that calculates the hours worked. As you can see, the shift that spans midnight presents a problem.

The screenshot shows a spreadsheet window titled 'Book3' with a single sheet named 'Sheet1'. The spreadsheet has four columns: A, B, C, and D. Row 1 contains the headers: 'Start Shift' in A1, 'End Shift' in B1, and 'Hours Worked' in C1. Row 2 shows a shift from 8:00 AM to 5:30 PM, resulting in 9:30 hours worked. Row 3 shows a shift from 10:00 PM to 6:00 AM, resulting in a series of hash marks (#####) in the 'Hours Worked' column, indicating an error.

	A	B	C	D
1	Start Shift	End Shift	Hours Worked	
2	8:00 AM	5:30 PM	9:30	
3	10:00 PM	6:00 AM	#####	
4				
5				
6				
7				

Figure 6-10: Calculating the number of hours worked returns an error if the shift spans midnight.

Using the ABS function (to calculate the absolute value) isn't an option in this case because it returns the wrong result (16 hours). The following formula, however, *does* work:

```
=(B2+(B2<A2))-A2)
```

Another, simpler, formula can do the job:

```
=MOD(B2-A2,1)
```



Negative times *are* permitted if the workbook uses the 1904 date system. To switch to the 1904 date system, select Tools → Options, and click the Calculation tab. Place a check mark next to the 1904 date system option. But beware! When changing the workbook's date system, if the workbook uses dates, the dates will be off by four years.

Converting from Military Time

Military time is expressed as a four-digit number from 0000 to 2359. For example, 1:00 AM is expressed as 0100 hours, and 3:30 PM is expressed as 1530 hours. The following formula converts such a number (assumed to appear in cell A1) to a standard time:

```
=TIMEVALUE(LEFT(A1,2)&"."&RIGHT(A1,2))
```

The formula returns an incorrect result if the contents of cell A1 do not contain four digits. The following formula corrects the problem, and returns a valid time for any military time value from 0 to 2359:

```
=TIMEVALUE(LEFT(TEXT(A1,"0000"),2)&"."&RIGHT(A1,2))
```

Following is a simpler formula that uses the TEXT function to return a formatted string, and then uses the TIMEVALUE function to express the result in terms of a time:

```
=TIMEVALUE(TEXT(A1,"00\ :00"))
```

Converting Decimal Hours, Minutes, or Seconds to a Time

To convert decimal hours to a time, divide the decimal hours by 24. For example, if cell A1 contains 9.25 (representing hours), this formula returns 09:15:00 (nine hours, 15 minutes):

```
=A1/24
```

To convert decimal minutes to a time, divide the decimal hours by 1,440 (the number of minutes in a day). For example, if cell A1 contains 500 (representing minutes), the following formula returns 08:20:00 (eight hours, 20 minutes):

```
=A1/1440
```

To convert decimal seconds to a time, divide the decimal hours by 86,400 (the number of seconds in a day). For example, if cell A1 contains 65,000 (representing seconds), the following formula returns 18:03:20 (18 hours, three minutes, and 20 seconds):

```
=A1/86400
```

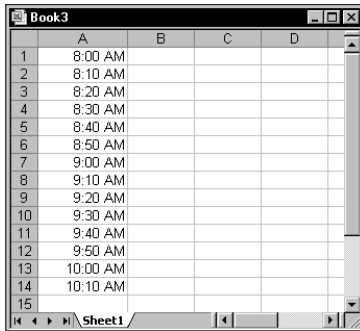
Adding Hours, Minutes, or Seconds to a Time

You can use the TIME function to add any number of hours, minutes, or seconds to a time. For example, assume cell A1 contains a time. The following formula adds two hours and 30 minutes to that time and displays the result:

```
=A1+TIME(2,30,0)
```

You can use the TIME function to fill a range of cells with incremental times. Figure 6-11 shows a worksheet with a series of times in 10-minute increments. Cell A1 contains a time that was entered directly. Cell A2 contains the following formula, which copied down the column:

```
=A1+TIME(0,10,0)
```



	A	B	C	D
1	8:00 AM			
2	8:10 AM			
3	8:20 AM			
4	8:30 AM			
5	8:40 AM			
6	8:50 AM			
7	9:00 AM			
8	9:10 AM			
9	9:20 AM			
10	9:30 AM			
11	9:40 AM			
12	9:50 AM			
13	10:00 AM			
14	10:10 AM			
15				

Figure 6-11: Using a formula to create a series of incremental times

Converting between Time Zones

You may receive a worksheet that contains dates and times in Greenwich Mean Time (GMT, sometimes referred to as Zulu time), and you need to convert these values to local time. To convert dates and times into local times, you need to determine the difference in hours between the two time zones. For example, to convert GMT times to U.S. Central Standard Time, the hour conversion factor is -6.

You can't use the TIME function with a negative argument, so you need to take a different approach. One hour equals 1/24 of a day, so you can divide the time conversion factor by 24, and then add it to the time.

Figure 6-12 shows a worksheet set up to convert dates and times (expressed in GMT) to local times. Cell B1 contains the hour conversion factor (-5 hours for U.S. Eastern Standard Time). The formula in B4, which copies down the column, is:

```
=A4+($B$1/24)
```

	A	B	C
1	Conversion Factor:	.5 hours	
2			
3	GMT	Local Time	
4	01/05/2002 01:00 AM	01/04/2002 08:00 PM	
5	01/05/2002 03:30 AM	01/04/2002 10:30 PM	
6	01/05/2002 06:00 AM	01/05/2002 01:00 AM	
7	01/05/2002 08:30 AM	01/05/2002 03:30 AM	
8	01/05/2002 11:00 AM	01/05/2002 06:00 AM	
9	01/05/2002 01:30 PM	01/05/2002 08:30 AM	
10	01/05/2002 04:00 PM	01/05/2002 11:00 AM	
11	01/05/2002 06:30 PM	01/05/2002 01:30 PM	
12	01/05/2002 09:00 PM	01/05/2002 04:00 PM	
13	01/05/2002 11:30 PM	01/05/2002 06:30 PM	
14	01/06/2002 02:00 AM	01/05/2002 09:00 PM	
15	01/06/2002 04:30 AM	01/05/2002 11:30 PM	
16	01/06/2002 07:00 AM	01/06/2002 02:00 AM	
17	01/06/2002 09:30 AM	01/06/2002 04:30 AM	
18	01/06/2002 12:00 PM	01/06/2002 07:00 AM	
19			
20			
21			

Figure 6-12: This worksheet converts dates and times between time zones.

This formula effectively adds x hours to the date and time in column A. If cell B1 contains a negative hour value, the value subtracts from the date and time in column A. Note that, in some cases, this also affects the date.

Rounding Time Values

You may need to create a formula that rounds a time to a particular value. For example, you may need to enter your company's time records rounded to the nearest 15 minutes. This section presents examples of various ways to round a time value.

The following formula rounds the time in cell A1 to the nearest minute:

```
=ROUND(A1*1440,0)/1440
```

The formula works by multiplying the time by 1440 (to get total minutes). This value is passed to the ROUND function, and the result is divided by 1440. For example, if cell A1 contains 11:52:34, the formula returns 11:53:00.

The following formula resembles this example, except that it rounds the time in cell A1 to the nearest hour:

```
=ROUND(A1*24,0)/24
```

If cell A1 contains 5:21:31, the formula returns 5:00:00.

The following formula rounds the time in cell A1 to the nearest 15 minutes:

```
=ROUND(A1*24/0.25,0)*(0.25/24)
```

In this formula, 0.25 represents the fractional hour. To round a time to the nearest 30 minutes, change 0.25 to 0.5, as in the following formula:

```
=ROUND(A1*24/0.5,0)*(0.5/24)
```

Working with Non-Time-of-Day Values

Sometimes, you may want to work with time values that don't represent an actual time of day. For example, you might want to create a list of the finish times for a race, or record the time you spend jogging each day. Such times don't represent a time of day. Rather, a value represents the time for an event (in hours, minutes, and seconds). The time to complete a test, for instance, might take 35 minutes and 40 seconds. You can enter that value into a cell as:

```
00:35:45
```

Excel interprets such an entry as 12:35:45 AM, which works fine (just make sure that you format the cell so it appears as you like). When you enter such times that do not have an hour component, you must include at least one zero for the hour. If you omit a leading zero for a missing hour, Excel interprets your entry as 35 hours and 45 minutes.

Figure 6-13 shows an example of a worksheet set up to keep track of someone's jogging activity. Column A contains simple dates. Column B contains the distance, in miles. Column C contains the time it took to run the distance. Column D contains formulas to calculate the speed, in miles per hour. For example, the formula in cell D2 is:

```
=B2/(C2*24)
```

	A	B	C	D	E	F	G
1	Date	Distance	Time	Speed (mph)	Pace (min/mile)	YTD Distance	Cumulative Time
2	1/1/2002	1.50	00:18:45	4.80	12.50	1.50	00:18:45
3	1/2/2002	1.50	00:17:40	5.09	11.78	3.00	00:36:25
4	1/3/2002	2.00	00:21:30	5.58	10.75	5.00	00:57:55
5	1/4/2002	1.50	00:15:20	5.87	10.22	6.50	01:13:15
6	1/5/2002	2.40	00:25:05	5.74	10.45	8.90	01:38:20
7	1/6/2002	3.00	00:31:06	5.79	10.37	11.90	02:09:26
8	1/7/2002	3.80	00:41:06	5.55	10.82	15.70	02:50:32
9	1/8/2002	5.00	01:09:00	4.35	13.80	20.70	03:59:32
10	1/9/2002	4.00	00:45:10	5.31	11.29	24.70	04:44:42
11	1/10/2002	3.00	00:29:06	6.19	9.70	27.70	05:13:48
12	1/11/2002	5.50	01:08:30	4.82	12.45	33.20	06:22:18
13							

Figure 6-13: This worksheet uses times not associated with a time of day.

Column E contains formulas to calculate the pace, in minutes per mile. For example, the formula in cell E2 is:

```
=(C2*60*24)/B2
```

Columns F and G contain formulas that calculate the year-to-date distance (using column B), and the cumulative time (using column C). The cells in column G are formatted using the following number format (which permits time displays that exceed 24 hours):

```
[hh]:mm:ss
```



You can also access the workbook shown in Figure 6-13 on the companion CD-ROM.

Summary

This chapter explored the date- and time-related features of Excel. I provided an overview of Excel's serial number date and time system, and I described how to enter dates and times into cells. The chapter also listed many examples of formulas that use dates and times.

The next chapter presents various techniques to count data in a spreadsheet.

Chapter 7

Counting and Summing Techniques

IN THIS CHAPTER

- ◆ Information on counting and summing cells
- ◆ Information on counting and summing records in databases and pivot tables
- ◆ Basic counting formulas
- ◆ Advanced counting formulas
- ◆ Formulas for performing common summing tasks
- ◆ Conditional summing formulas using a single criterion
- ◆ Conditional summing formulas using multiple criteria
- ◆ The use of VBA to perform counting and summing tasks

MANY OF THE MOST FREQUENTLY ASKED spreadsheet questions involve counting and summing values and other worksheet elements. It seems that people are always looking for formulas to count or sum various items in a worksheet. If I've done my job, this chapter will answer the vast majority of such questions.

Counting and Summing Worksheet Cells

Generally, a counting formula returns the number of cells in a specified range that meet certain criteria. A summing formula returns the sum of the values of the cells in a range that meet certain criteria. The range you want counted or summed may or may not consist of a worksheet database.

Table 7-1 lists Excel's worksheet functions that come into play when creating counting and summing formulas. If none of the functions in Table 7-1 can solve your problem, it's likely that an array formula can come to the rescue.



See Part V for detailed information and examples of array formulas used for counting and summing.

TABLE 7-1 EXCEL'S COUNTING AND SUMMING FUNCTIONS

Function	Description
COUNT	Returns the number of cells in a range that contain a numeric value
COUNTA	Returns the number of nonblank cells in a range
COUNTBLANK	Returns the number of blank cells in a range
COUNTIF	Returns the number of cells in a range that meet a specified criterion
DCOUNT	Counts the number of records in a worksheet database that meet specified criteria
DCOUNTA	Counts the number of nonblank records in a worksheet database that meet specified criteria
DEVSQ	Returns the sum of squares of deviations of data points from the sample mean; used primarily in statistical formulas
DSUM	Returns the sum of a column of values in a worksheet database that meet specified criteria
FREQUENCY	Calculates how often values occur within a range of values, and returns a vertical array of numbers; used only in a multicell array formula
SUBTOTAL	When used with a first argument of 2 or 3, returns a count of cells that comprise a subtotal; when used with a first argument of 9, returns the sum of cells that comprise a subtotal
SUM	Returns the sum of its arguments
SUMIF	Returns the sum of cells in a range that meet a specified criterion
SUMPRODUCT	Multiplies corresponding cells in two or more ranges, and returns the sum of those products
SUMSQ	Returns the sum of the squares of its arguments; used primarily in statistical formulas

Function	Description
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two ranges; used primarily in statistical formulas
SUMXMY2	Returns the sum of squares of the differences of corresponding values in two ranges; used primarily in statistical formulas
SUMX2MY2	Returns the sum of the differences of squares of corresponding values in two ranges; used primarily in statistical formulas

Counting or Summing Records in Databases and Pivot Tables

Special database functions and pivot tables provide additional ways to achieve counting and summing. Excel's DCOUNT and DSUM functions are database functions. They work in conjunction with a worksheet database, and require a special criterion range that holds the counting or summing criteria.



Chapter 9 covers the database functions and provides information about counting and summing using a worksheet database.

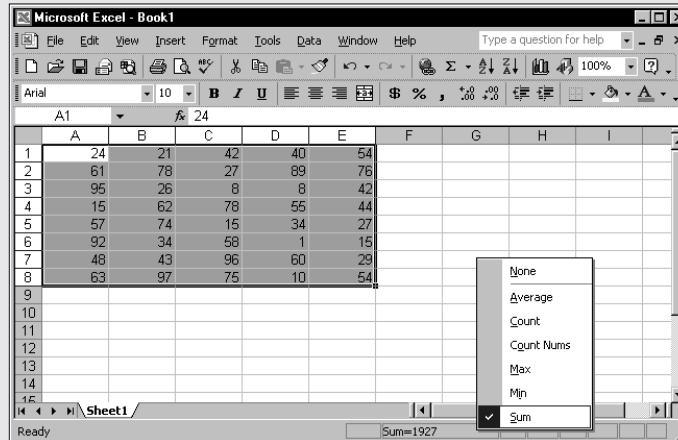
Creating a pivot table is a great way to get a count or sum of items without using formulas. Like the database function, using a pivot table is appropriate when your data appears in the form of a database.



Refer to Chapter 18 for information about pivot tables.

Getting a Quick Count or Sum

In Excel 97, Microsoft introduced a feature known as AutoCalculate. This feature displays, in the status bar, information about the selected range. Normally, the status bar displays the sum of the values in the selected range. You can, however, right-click the AutoCalculate display to bring up a menu with some other options.



If you select Count, the status bar displays the number of nonempty cells in the selected range. If you select Count Nums, the status bar displays the number of numeric cells in the selected range.

Basic Counting Formulas

The basic counting formulas presented here are all straightforward and relatively simple. They demonstrate the ability of Excel's counting functions to count the number of cells in a range that meet specific criteria. Figure 7-1 shows a worksheet that uses formulas (in column E) to summarize the contents of range A1:B10 – a 20-cell range named *Data*.



You can access the workbook shown in Figure 7-1 on the companion CD-ROM.

	A	B	C	D	E	F
1	Jan	Feb		Total cells:	20	
2	525	718		Blank cells:	6	
3				Nonblank cells:	14	
4	3			Numeric values:	7	
5	552	911		Non-text cells:	17	
6	250	98		Text cells:	3	
7				Logical values:	2	
8	TRUE	FALSE		Error values:	2	
9		#DIV/0!		#N/A errors:	0	
10	Total	#NAME?		#NULL errors:	0	
11				#DIV/0! errors:	1	
12				#VALUE! errors:	0	
13				#REF! errors:	0	
14				#NAME? errors:	1	
15				#NUM! errors:	0	
16						

Figure 7-1: Formulas provide various counts of the data in A1:B10.

Counting the Total Number of Cells

To get a count of the total number of cells in a range, use the following formula. This formula returns the number of cells in a range named *Data*. It simply multiplies the number of rows (returned by the ROWS function) by the number of columns (returned by the COLUMNS function).

```
=ROWS(Data)*COLUMNS(Data)
```

Counting Blank Cells

The following formula returns the number of blank (empty) cells in a range named *Data*:

```
=COUNTBLANK(Data)
```

About This Chapter's Examples

Many of the examples in this chapter consist of array formulas. An array formula, as explained in Chapter 14, is a special type of formula. You can spot an array formula because it is enclosed in brackets when it is displayed in the formula bar. For example:

```
{=Data*2}
```

When you enter an array formula, press Ctrl+Shift+Enter (not just Enter). And don't type the brackets (Excel inserts the brackets for you).

The COUNTBLANK function also counts cells containing a formula that returns an empty string. For example, the formula that follows returns an empty string if the value in cell A1 is greater than 5. If the cell meets this condition, then the COUNTBLANK function counts that cell.

```
=IF(A1>5, "", A1)
```



The COUNTBLANK function does not count cells that contain a zero value, even if you uncheck the Zero values option in the Options dialog box (select Tools → Options, then click the View tab).

You can use the COUNTBLANK function with an argument that consists of entire rows or columns. For example, this next formula returns the number of blank cells in column A:

```
=COUNTBLANK(A:A)
```

The following formula returns the number of empty cells on the entire worksheet named Sheet1. You must enter this formula on a sheet other than Sheet1, or it will create a circular reference.

```
=COUNTBLANK(Sheet1!1:65536)
```

Counting Nonblank Cells

The following formula uses the COUNTA function to return the number of non-blank cells in a range named *Data*:

```
=COUNTA(Data)
```

The COUNTA function counts cells that contain values, text, or logical values (TRUE or FALSE).



If a cell contains a formula that returns an empty string, that cell is included in the count returned by COUNTA, even though the cell appears to be blank.

Counting Numeric Cells

To count only the numeric cells in a range, use the following formula (which assumes the range is named *Data*):

```
=COUNT(Data)
```

Cells that contain a date or a time are considered to be numeric cells. Cells that contain a logical value (TRUE or FALSE) are not considered to be numeric cells.

Counting Nontext Cells

The following array formula uses Excel's ISNONTEXT function, which returns TRUE if its argument refers to any nontext cell (including a blank cell). This formula returns the count of the number of cells not containing text (including blank cells):

```
{=SUM(IF(ISNONTEXT(Data),1))}
```

Counting Text Cells

To count the number of text cells in a range, you need to use an array formula. The array formula that follows returns the number of text cells in a range named *Data*:

```
{=SUM(IF(ISTEXT(Data),1))}
```

Counting Logical Values

The following array formula returns the number of logical values (TRUE or FALSE) in a range named *Data*:

```
{=SUM(IF(ISLOGICAL(Data),1))}
```

Error Values in a Range

Excel has three functions that help you determine whether a cell contains an error value:

- ◆ ISERROR: Returns TRUE if the cell contains any error value (#N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!)
- ◆ ISERR: Returns TRUE if the cell contains any error value except #N/A
- ◆ ISNA: Returns TRUE if the cell contains the #N/A error value

You can use these functions in an array formula to count the number of error values in a range. The following array formula, for example, returns the total number of error values in a range named *Data*:

```
{=SUM(IF(ISERROR(data),1))}
```

Depending on your needs, you can use the ISERR or ISNA function in place of ISERROR.

If you would like to count specific types of errors, you can use the COUNTIF function. The following formula, for example, returns the number of #DIV/0! error values in the range named *Data*:

```
=COUNTIF(Data,"#DIV/0!")
```

Advanced Counting Formulas

Most of the basic examples I presented previously use functions or formulas that perform conditional counting. The advanced counting formulas that I present here represent more complex examples for counting worksheet cells, based on various types of criteria.

Counting Cells Using the COUNTIF Function

Excel's COUNTIF function is useful for single-criterion counting formulas. The COUNTIF function takes two arguments:

- ◆ *range*: The range that contains the values that determine whether to include a particular cell in the count.
- ◆ *criteria*: The logical criteria that determine whether to include a particular cell in the count.

Listed here are several examples of formulas that use the COUNTIF function. These formulas all work with a range named *Data*. As you can see, the *criteria* argument proves quite flexible. You can use constants, expressions, functions, cell references, and even wildcard characters (* and ?).

The following formula returns the number of cells containing the value 12:

```
=COUNTIF(Data,12)
```

The following formula returns the number of cells containing a negative value:

```
=COUNTIF(Data,"<0")
```

The following formula returns the number of cells not equal to 0:

```
=COUNTIF(Data,"<>0")
```

The following formula returns the number of cells greater than 5:

```
=COUNTIF(Data,">5")
```

The following formula returns the number of cells equal to the contents of cell A1:

```
=COUNTIF(Data,A1)
```

The following formula returns the number of cells greater than the value in cell A1:

```
=COUNTIF(Data,">"&A1)
```

The following formula returns the number of cells containing text:

```
=COUNTIF(Data,"*")
```

The following formula returns the number of text cells containing exactly three characters:

```
=COUNTIF(Data,"???")
```

The following formula returns the number of cells containing the single word *budget* (not case sensitive):

```
=COUNTIF(Data,"budget")
```

The following formula returns the number cells containing the text *budget* anywhere within the text:

```
=COUNTIF(Data,"*budget*")
```

The following formula returns the number of cells containing text that begins with the letter A (not case sensitive):

```
=COUNTIF(Data,"A*")
```

The following formula returns the number of cells containing the current date:

```
=COUNTIF(Data,TODAY())
```


The following formula returns the number of cells with a value greater than the average:

```
=COUNTIF(Data, ">"&AVERAGE(Data))
```

The following formula returns the number of values exceeding three standard deviations above the mean:

```
=COUNTIF(Data, ">"&AVERAGE(Data)+STDEV(Data)*3)
```

The following formula returns the number of cells containing the value 3 or -3:

```
=COUNTIF(Data, 3)+COUNTIF(Data, -3)
```

The following formula returns the number of cells containing logical TRUE:

```
=COUNTIF(Data, TRUE)
```

The following formula returns the number of cells containing a logical value (TRUE or FALSE):

```
=COUNTIF(Data, TRUE)+COUNTIF(Data, FALSE)
```

The following formula returns the number of cells containing the #N/A error value:

```
=COUNTIF(Data, "#N/A")
```

Counting Cells Using Multiple Criteria

In many cases, your counting formula will need to count cells only if two or more criteria are met. These criteria can be based on the cells that are being counted, or based on a range of corresponding cells.

Figure 7-2 shows a simple worksheet that I use for the examples in this section. This sheet shows sales data categorized by Month, SalesRep, and Type. The worksheet contains named ranges that correspond to the labels in row 1.



This workbook is available on the companion CD-ROM.

	A	B	C	D
1	Month	SalesRep	Type	Amount
2	January	Albert	New	85
3	January	Albert	New	675
4	January	Brooks	New	130
5	January	Cook	New	1350
6	January	Cook	Existing	685
7	January	Brooks	New	1350
8	January	Cook	New	475
9	January	Brooks	New	1205
10	February	Brooks	Existing	450
11	February	Albert	New	495
12	February	Cook	New	210
13	February	Cook	Existing	1050
14	February	Albert	New	140
15	February	Brooks	New	900
16	February	Brooks	New	900
17	February	Cook	New	95
18	February	Cook	New	780
19	March	Brooks	New	900
20	March	Albert	Existing	875
21	March	Brooks	New	50
22	March	Brooks	New	875
23	March	Cook	Existing	225
24	March	Cook	New	175
25	March	Brooks	Existing	400
26	March	Albert	New	840
27	March	Cook	New	132

Figure 7-2: This worksheet demonstrates various counting techniques that use multiple criteria.

USING AND CRITERIA

An And criterion counts cells if all specified conditions are met. A common example is a formula that counts the number of values that fall within a numerical range. For example, you may want to count cells that contain a value greater than 0 *and* less than or equal to 12. Any cell that has a positive value less than or equal to 12 will be included in the count. For this example, the COUNTIF function will do the job:

```
=COUNTIF(Data, ">0")-COUNTIF(Data, ">12")
```

This formula counts the number of values that are greater than 0 and then subtracts the number of values that are greater than 12. The result is the number of cells that contain a value greater than 0 and less than or equal to 12.

Creating this type of formula can be confusing, because the formula refers to a condition ">12" even though the goal is to count values that are less than or equal to 12. An alternate technique is to use an array formula, such as the one that follows. You may find creating this type of formula easier.

```
{=SUM((Data>0)*(Data<=12))}
```

Sometimes, the counting criteria will be based on cells other than the cells being counted. You may, for example, want to count the number of sales that meet the following criteria:

- ◆ Month is January, *and*
- ◆ SalesRep is Brooks, *and*
- ◆ Amount is greater than 1000

The following array formula returns the number of items that meet all three criteria:

```
{=SUM((Month="January")*(SalesRep="Brooks")*(Amount>1000))}
```

USING OR CRITERIA

To count cells using an Or criterion, you can sometimes use multiple COUNTIF functions. The following formula, for example, counts the number of 1s, 3s, and 5s in the range named *Data*:

```
=COUNTIF(Data,1)+COUNTIF(Data,3)+COUNTIF(Data,5)
```

You can also use the COUNTIF function in an array formula. The following array formula, for example, returns the same result as the previous formula:

```
{=SUM(COUNTIF(Data,{1,3,5}))}
```

But if you base your Or criteria on cells other than the cells being counted, the COUNTIF function won't work. Refer back to Figure 7-2. Suppose you want to count the number of sales that meet the following criteria:

- ◆ Month is January, *or*
- ◆ SalesRep is Brooks, *or*
- ◆ Amount is greater than 1000

The following array formula returns the correct count:

```
{=SUM(IF((Month="January")+(SalesRep="Brooks")+(Amount>1000),1))}
```

COMBINING AND AND OR CRITERIA

You can combine And and Or criteria when counting. For example, perhaps you want to count sales that meet the following criteria:

- ◆ Month is January, *and*
- ◆ SalesRep is Brooks, *or* SalesRep is Cook

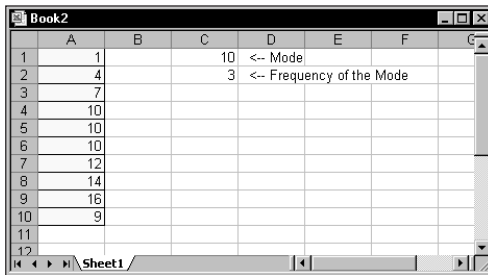
This array formula returns the number of sales that meet the criteria:

```
{=SUM((Month="January")*IF((SalesRep="Brooks")+
(SalesRep="Cook"),1))}
```

Counting the Most Frequently Occurring Entry

Excel's `MODE` function returns the most frequently occurring value in a range. Figure 7-3 shows a worksheet with values in range A1:A10 (named *Data*). The formula that follows returns 10 because that value appears most frequently in the *Data* range:

```
=MODE(Data)
```



	A	B	C	D	E	F	G
1	1			10	<- Mode		
2	4			3	<- Frequency of the Mode		
3	7						
4	10						
5	10						
6	10						
7	12						
8	14						
9	16						
10	9						
11							
12							

Figure 7-3: The `MODE` function returns the most frequently occurring value in a range.

To count the number of times the most frequently occurring value appears in the range (in other words, the frequency of the mode), use the following formula:

```
=COUNTIF(Data,MODE(Data))
```

This formula returns 3, because the modal value (10) appears three times in the *Data* range.

The `MODE` function works only for numeric values. It simply ignores cells that contain text. To find the most frequently occurring text entry in a range, you need to use an array formula.

To count the number of times the most frequently occurring item (text or values) appears in a range named *Data*, use the following array formula:

```
{=MAX(COUNTIF(Data,Data))}
```

This next array formula operates like the `MODE` function, except that it works with both text and values:

```
{=INDEX(Data,MATCH(MAX(COUNTIF(Data,Data)),COUNTIF(Data,Data),0))}
```

Counting the Occurrences of Specific Text

The examples in this section demonstrate various ways to count the occurrences of a character or text string in a range of cells. Figure 7-4 shows a worksheet used for these examples. Various text appears in the range A1:A10 (named *Data*); cell B1 is named *Text*.

	A	B	C	D	E	F	G
1	aa	Alpha					
2	Alpha						
3	AAA						
4	aaa						
5	Beta						
6	B						
7	BBB						
8	Alpha Beta						
9	AB						
10	alpha						
11							
12		2 Entire cell (not case-sensitive)					
13		1 Entire cell (case-sensitive)					
14							
15		3 Part of cell (not case-sensitive)					
16		2 Part of cell (case-sensitive)					
17							
18		3 Total occurrences in range (not case-sensitive)					
19		2 Total occurrences in range (case-sensitive)					
20							

Figure 7-4: This worksheet demonstrates various ways to count characters in a range.



The companion CD-ROM contains a workbook that demonstrates the formulas in this section.

ENTIRE CELL CONTENTS

To count the number of cells containing the contents of the *Text* cell (and nothing else), you can use the COUNTIF function. The following formula demonstrates:

```
=COUNTIF(Data,Text)
```

For example, if the *Text* cell contains the string “Alpha”, the formula returns 2 because two cells in the *Data* range contain this text. This formula is not case sensitive, so it counts both “Alpha” (cell A2) and “alpha” (cell A10). Note, however, that it does not count the cell that contains “Alpha Beta” (cell A8).

The following array formula is similar to the preceding formula, but this one is case sensitive:

```
{=SUM(IF(EXACT(Data,Text),1))}
```

PARTIAL CELL CONTENTS

To count the number of cells that contain a string that includes the contents of the *Text* cell, use this formula:

```
=COUNTIF(Data,"*"&Text&"*")
```

For example, if the *Text* cell contains the text “Alpha”, the formula returns 3, because three cells in the *Data* range contain the text “alpha” (cells A2, A8, and A10). Note that the comparison is not case sensitive.

If you need a case-sensitive count, you can use the following array formula:

```
{=SUM(IF(LEN(Data)-LEN(SUBSTITUTE(Data,Text,""))>0,1))}
```

If the *Text* cells contain the text “Alpha”, the preceding formula returns 2 because the string appears in two cells (A2 and A8).

TOTAL OCCURRENCES IN A RANGE

To count the total number of occurrences of a string within a range of cells, use the following array formula:

```
{=(SUM(LEN(Data))-SUM(LEN(SUBSTITUTE(Data,Text,""))))/LEN(Text)}
```

If the *Text* cell contains the character “B,” the formula returns 7 because the range contains seven instances of the string. This formula is case sensitive.

The following array formula is a modified version that is not case sensitive:

```
{=(SUM(LEN(Data))-SUM(LEN(SUBSTITUTE(UPPER(Data),UPPER(Text),""))))/LEN(Text)}
```

Counting the Number of Unique Values

The following array formula returns the number of unique values in a range named *Data*:

```
{=SUM(1/COUNTIF(Data,Data))}
```

To understand how this formula works, you need a basic understanding of array formulas. (See Chapter 14 for an introduction to this topic.) In Figure 7-5, range A1:A12 is named *Data*. Range C1:C12 contains the following array formula (entered into all 12 cells in the range):

```
{=COUNTIF(Data,Data)}
```

	A	B	C	D	E	F	G
1	100		3	0.333333			
2	100		3	0.333333			
3	100		3	0.333333			
4	200		2	0.5			
5	200		2	0.5			
6	300		1	1			
7	400		2	0.5			
8	400		2	0.5			
9	500		4	0.25			
10	500		4	0.25			
11	500		4	0.25			
12	500		4	0.25			
13				5	<- Unique items in Column A		

Figure 7-5: Using an array formula to count the number of unique values in a range



You can access the workbook shown in Figure 7-5 on the companion CD-ROM.

The array in range C1:C12 consists of the count of each value in *Data*. For example, the number 100 appears three times, so each array element that corresponds to a value of 100 in the *Data* range has a value of 3.

Range D1:D12 contains the following array formula:

```
{=1/C1:C12}
```

This array consists of each value in the array in range C1:C12, divided into 1. For example, each cell in the original *Data* range that contains a 200 has a value of 0.5 in the corresponding cell in D1:D12.

Summing the range D1:D12 gives the number of unique items in *Data*. The array formula presented at the beginning of this section essentially creates the array that occupies D1:D12, and sums the values.

This formula has a serious limitation: If the range contains any blank cells, it returns an error. The following array formula solves this problem:

```
{=SUM(IF(COUNTIF(Data,Data)=0,"",1/COUNTIF(Data,Data)))}
```



To create an array formula that returns a list of unique items in a range, refer to Chapter 15.

Creating a Frequency Distribution

A frequency distribution basically comprises a summary table that shows the frequency of each value in a range. For example, an instructor may create a frequency distribution of test scores. The table would show the count of As, Bs, Cs, and so on. Excel provides a number of ways to create frequency distributions. You can:

- ◆ Use the FREQUENCY function
- ◆ Create your own formulas
- ◆ Use the Analysis ToolPak add-in



A workbook that demonstrates these three techniques appears on the companion CD-ROM.



If your data is in the form of a database, you can also use a pivot table to create a frequency distribution.

THE FREQUENCY FUNCTION

Using Excel's FREQUENCY function presents the easiest way to create a frequency distribution. This function always returns an array, so you must use it in an array formula entered into a multicell range.

Figure 7-6 shows some data in range A1:E20 (named *Data*). These values range from 1 to 500. The range G2:G11 contains the bins used for the frequency distribution. Each cell in this bin range contains the upper limit for the bin. In this case, the bins consist of 1–50, 51–100, 101–150, and so on. See the sidebar, “Creating Bins for a Frequency Distribution” to discover an easy way to create a bin range.

	A	B	C	D	E	F	G	H
1	55	316	223	185	124		Bins	
2	124	93	163	213	314		50	
3	211	41	231	241	212		100	
4	118	113	400	205	254		150	
5	262	1	201	172	101		200	
6	167	479	205	337	118		250	
7	489	15	89	362	148		300	
8	179	248	125	197	177		350	
9	456	153	269	49	127		400	
10	289	500	198	317	300		450	
11	126	114	303	314	270		500	
12	151	279	347	314	170			
13	250	175	93	209	61			
14	166	113	356	124	242			
15	152	384	157	233	99			
16	277	195	436	6	240			
17	147	80	173	211	244			
18	386	93	330	400	141			
19	332	173	129	323	188			
20	338	263	444	84	220			
21								

Figure 7-6: Creating a frequency distribution for the data in A1:E20

To create the frequency distribution, select a range of cells that correspond to the number of cells in the bin range. Then enter the following array formula:

```
{=FREQUENCY(Data,G2:G11)}
```

The array formula enters the count of values in the *Data* range that fall into each bin. To create a frequency distribution that consists of percentages, use the following array formula:

```
{=FREQUENCY(Data,G2:G10)/COUNT(Data)}
```

Figure 7-7 shows two frequency distributions – one in terms of counts, and one in terms of percentages. The figure also shows a chart (histogram) created from the frequency distribution.

USING FORMULAS TO CREATE A FREQUENCY DISTRIBUTION

Figure 7-8 shows a worksheet that contains test scores for 50 students in column B (the range is named *Grades*). Formulas in columns G and H calculate a frequency distribution for letter grades. The minimum and maximum values for each letter grade appear in columns D and E. For example, a test score between 80 and 89 (inclusive) qualifies for a B.

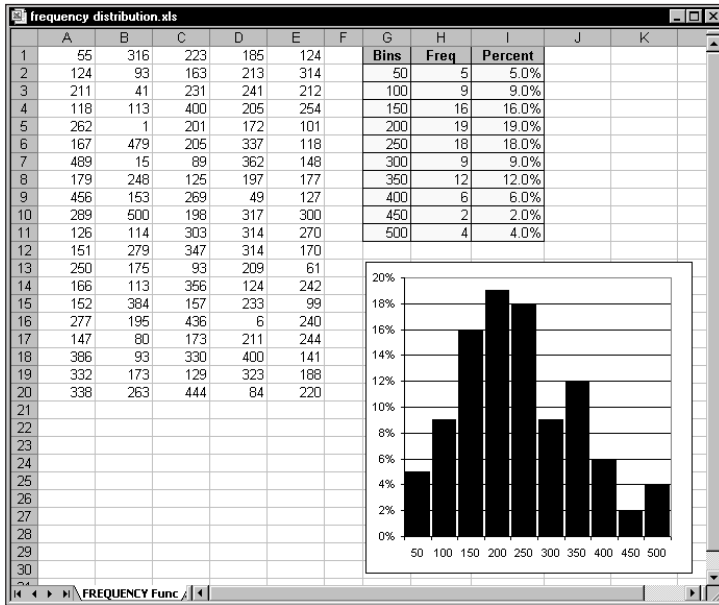


Figure 7-7: Frequency distributions created using the FREQUENCY function

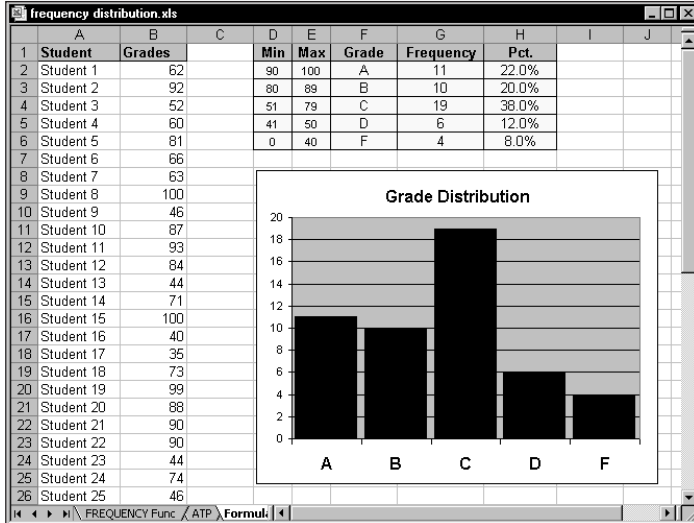


Figure 7-8: Creating a frequency distribution of test scores

Creating Bins for a Frequency Distribution

When creating a frequency distribution, you must first enter the values into the bin range. The number of bins determines the number of categories in the distribution. Most of the time, each of these bins will represent an equal range of values.

To create 10 evenly spaced bins for values in a range named *Data*, enter the following array formula into a range of 10 cells in a column:

```
{=MIN(Data)+(ROW(INDIRECT("1:10"))*(MAX(Data)-MIN(Data)+1)/10)-1}
```

This formula creates 10 bins, based on the values in the *Data* range. The upper bin will always equal the maximum value in the range.

To create more or fewer bins, use a value other than 10 and enter the array formula into a range that contains the same number of cells. For example, to create five bins, enter the following array formula into a five-cell vertical range:

```
{=MIN(Data)+(ROW(INDIRECT("1:5"))*(MAX(Data)-MIN(Data)+1)/5)-1}
```

The formula in cell G2 that follows is an array formula that counts the number of scores that qualify for an A:

```
{=SUM((Grades>=D2)*(Grades<=E2))}
```

You may recognize this formula from a previous section in this chapter (see “Counting Cells Using Multiple Criteria”). This formula was copied to the four cells below G2.

The formulas in column H calculate the percentage of scores for each letter grade. The formula in H2, which was copied to the four cells below H2, is:

```
=G2/SUM($G$2:$G$6)
```

USING THE ANALYSIS TOOLPAK TO CREATE A FREQUENCY DISTRIBUTION

Once you install the Analysis ToolPak add-in, you can use the Histogram option to create a frequency distribution. Start by entering your bin values in a range. Then select Tools → Data Analysis to display the Data Analysis dialog box. Next, select Histogram and click OK. You should see the Histogram dialog box shown in Figure 7-9.

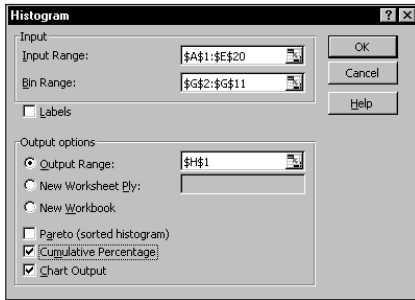


Figure 7-9: The Analysis ToolPak's Histogram dialog box

Specify the ranges for your data (Input Range), bins (Bin Range), and results (Output Range), and then select any options. Figure 7-10 shows a frequency distribution (and chart) created with the Histogram option.

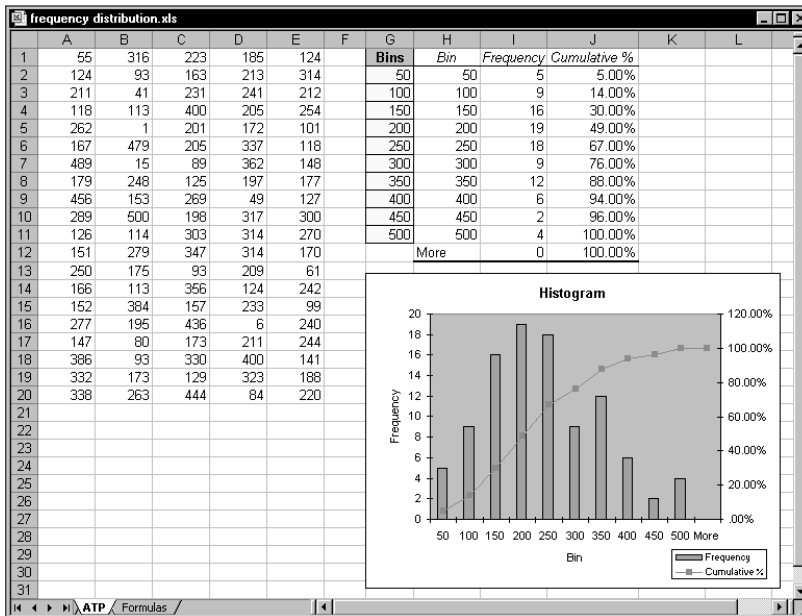


Figure 7-10: A frequency distribution and chart generated by the Analysis ToolPak's Histogram option



Note that the frequency distribution consists of values, not formulas. Therefore, if you make any changes to your input data, you need to rerun the Histogram procedure to update the results.

USING ADJUSTABLE BINS TO CREATE A HISTOGRAM

Figure 7-11 shows a worksheet with student grades listed in column B (67 students total). Columns D and E contain formulas that calculate the upper and lower limits for bins, based on the entry in cell E1 (named *BinSize*). For example, if *BinSize* is 10 (as in the figure), then each bin contains 10 scores (1–10, 11–20, and so on).

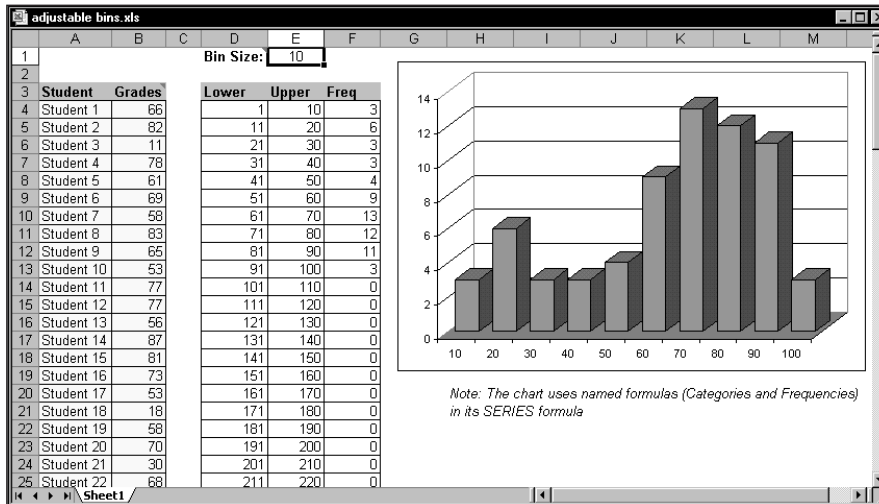


Figure 7-11: The chart displays a histogram; the contents of cell E1 determine the number of categories.



The workbook shown in Figure 7-11 also appears on the companion CD-ROM.

The chart uses two dynamic names in its *SERIES* formula. You can define the name *Categories* with the following formula:

```
=OFFSET(Sheet1!$E$4,0,0,ROUNDUP(100/BinSize,0))
```

You can define the name *Frequencies* with this formula:

```
=OFFSET(Sheet1!$F$4,0,0,ROUNDUP(100/BinSize,0))
```

The net effect is that the chart adjusts automatically when you change the *BinSize* cell.



See Chapter 17 for more about creating a chart that uses dynamic names in its SERIES formula.

Summing Formulas

The examples in this section demonstrate how to perform common summing tasks using formulas. The formulas range from very simple to relatively complex array formulas that compute sums using multiple criteria.

Summing All Cells in a Range

It doesn't get much simpler than this. The following formula returns the sum of all values in a range named *Data*:

```
=SUM(Data)
```

The SUM function can take up to 32 arguments. The following formula, for example, returns the sum of the values in five noncontiguous ranges:

```
=SUM(A1:A9,C1:C9,E1:E9,G1:G9,I1:I9)
```

You can use complete rows or columns as an argument for the SUM function. The formula that follows, for example, returns the sum of all values in column A. If this formula appears in a cell in column A, it generates a circular reference error.

```
=SUM(A:A)
```

The following formula returns the sum of all values on Sheet1. To avoid a circular reference error, this formula must appear on a sheet other than Sheet1.

```
=SUM(Sheet1!1:65536)
```

The SUM function is very versatile. The arguments can be numerical values, cells, ranges, text representations of numbers (which are interpreted as values), logical values, and even embedded functions. For example, consider the following formula:

```
=SUM(B1,5,"6",,SQRT(4),A1:A5,TRUE)
```

This formula, which is a perfectly valid formula, contains all of the following types of arguments, listed here in the order of their presentation:

- ◆ A single cell reference
- ◆ A literal value
- ◆ A string that looks like a value
- ◆ A missing argument
- ◆ An expression that uses another function
- ◆ A range reference
- ◆ A logical TRUE value



The SUM function is versatile, but it's also inconsistent when you use logical values (TRUE or FALSE). Logical values stored in cells are always treated as 0. But logical TRUE, when used as an argument in the SUM function, is treated as 1.

Computing a Cumulative Sum

You may want to display a cumulative sum of values in a range—sometimes known as a “running total.” Figure 7-12 illustrates a cumulative sum. Column B shows the monthly amounts, and column C displays the cumulative (year-to-date) totals.

	A	B	C
1	Month	Amount	Year-to-Date
2	January	850	850
3	February	900	1,750
4	March	750	2,500
5	April	1,100	3,600
6	May	600	4,200
7	June	500	4,700
8	July	1,200	5,900
9	August		5,900
10	September		5,900
11	October		5,900
12	November		5,900
13	December		5,900
14	TOTAL	5,900	5,900
15			

Figure 7-12: Simple formulas in column C display a cumulative sum of the values in column B.

The formula in cell C2 is:

```
=SUM(B$2:B2)
```

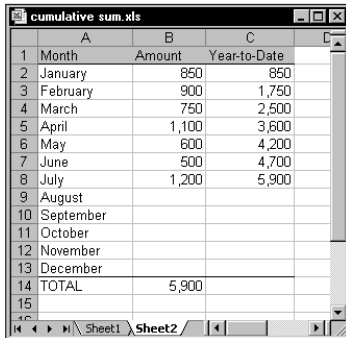
Notice that this formula uses a *mixed reference*. The first cell in the range reference always refers to row 2. When this formula is copied down the column, the range argument adjusts such that the sum always starts with row 2 and ends with the current row. For example, after copying this formula down column C, the formula in cell C8 is:

```
=SUM(B$2:B8)
```

You can use an IF function to hide the cumulative sums for rows in which data hasn't been entered. The following formula, entered in cell C2 and copied down the column, is:

```
=IF(B2<>"",SUM(B$2:B2),"")
```

Figure 7-13 shows this formula at work.



The screenshot shows a spreadsheet window titled 'cumulative sum.xls'. The data is as follows:

1	Month	Amount	Year-to-Date
2	January	850	850
3	February	900	1,750
4	March	750	2,500
5	April	1,100	3,600
6	May	600	4,200
7	June	500	4,700
8	July	1,200	5,900
9	August		
10	September		
11	October		
12	November		
13	December		
14	TOTAL	5,900	
15			

Figure 7-13: Using an IF function to hide cumulative sums for missing data



This workbook is available on the companion CD-ROM.

Summing the “Top n” Values

In some situations, you may need to sum the n largest values in a range—for example, the top 10 values. One approach is to sort the range in descending order, and then use the SUM function with an argument consisting of the first n values in the sorted range. An array formula such as this one accomplishes the task without sorting:

```
{=SUM(LARGE(Data,{1,2,3,4,5,6,7,8,9,10}))}
```

This formula sums the 10 largest values in a range named *Data*. To sum the 10 smallest values, use the SMALL function instead of the LARGE function:

```
{=SUM(SMALL(Data,{1,2,3,4,5,6,7,8,9,10}))}
```

These formulas use an array constant comprised of the arguments for the LARGE or SMALL function. If the value of n for your top- n calculation is large, you may prefer to use the following variation. This formula returns the sum of the top 30 values in the *Data* range. You can, of course, substitute a different value for 30.

```
{=SUM(LARGE(Data,ROW(INDIRECT("1:30"))))}
```

Conditional Sums Using a Single Criterion

Often, you need to calculate a *conditional sum*. With a conditional sum, values in a range that meet one or more conditions are included in the sum. This section presents examples of conditional summing using a single criterion.

The SUMIF function is very useful for single-criterion sum formulas. The SUMIF function takes three arguments:

- ◆ *range*: The range containing the values that determine whether to include a particular cell in the sum.
- ◆ *criteria*: An expression that determines whether to include a particular cell in the sum.
- ◆ *sum_range*: Optional. The range that contains the cells you want to sum. If you omit this argument, the function uses the range specified in the first argument.

The examples that follow demonstrate the use of the SUMIF function. These formulas are based on the worksheet shown in Figure 7-14, set up to track invoices. Column F contains a formula that subtracts the date in column E from the date in column D. A negative number in column F indicates a past-due payment. The worksheet uses named ranges that correspond to the labels in row 1.

	A	B	C	D	E	F
1	InvoiceNum	Office	Amount	DateDue	Today	Difference
2	AG-0145	Oregon	\$5,000.00	3-Apr	6-May	-33
3	AG-0189	California	\$450.00	21-Apr	6-May	-15
4	AG-0220	Washington	\$3,211.56	30-Apr	6-May	-6
5	AG-0310	Oregon	\$250.00	2-May	6-May	-4
6	AG-0355	Washington	\$125.50	6-May	6-May	0
7	AG-0409	Washington	\$3,000.00	12-May	6-May	6
8	AG-0581	Oregon	\$2,100.00	25-May	6-May	19
9	AG-0600	Oregon	\$335.39	25-May	6-May	19
10	AG-0602	Washington	\$65.00	30-May	6-May	24
11	AG-0633	California	\$250.00	1-Jun	6-May	26
12	TOTAL		\$14,787.45			36
13						

Figure 7-14: A negative value in column F indicates a past-due payment.

Let a Wizard Create Your Formula

Beginning with Excel 97, Excel ships with an add-in called Conditional Sum Wizard. Once you install this add-in, you can invoke the Wizard by selecting Tools → Conditional Sum.

The Conditional Sum Wizard helps you write formulas that sum specific values in a column based on other values in the list.

Region	Code	Sales Amount
North	Retail	\$413
East	Wholesale	\$166
North	Retail	\$538
North	Wholesale	\$230
		+
		\$951

Where is the list that contains the values to sum, including the column labels?

Sheet1!\$A\$1:\$F\$11

Buttons: Cancel, < Back, Next >, Finish

You can specify various conditions for your summing, and the add-in creates the formula for you (always an array formula). The Conditional Sum Wizard add-in, although a handy tool, is not all that versatile. For example, you can combine multiple criteria using an And condition, but not an Or condition.

By the way, the data table shown in the Conditional Sum Wizard dialog box does not use your actual data.



All of the examples in this section also appear on the companion CD-ROM.

Summing Only Negative Values

The following formula returns the sum of the negative values in column F. In other words, it returns the total number of past-due days for all invoices. For this worksheet, the formula returns -58.

```
=SUMIF(Difference,"<0")
```

Because you omit the third argument, the second argument (" <0 ") applies to the values in the *Difference* range.



You can also use the following array formula to sum the negative values in the *Difference* range:

```
{=SUM(IF(Difference<0,Difference))}
```

You do not need to hard-code the arguments for the SUMIF function into your formula. For example, you can create a formula such as the following, which gets the criteria argument from the contents of cell G2:

```
=SUMIF(Difference,G2)
```

This formula returns a new result if you change the criteria in cell G2.

Summing Values Based on a Different Range

The following formula returns the sum of the past-due invoice amounts (in column C):

```
=SUMIF(Difference,"<0",Amount)
```

This formula uses the values in the *Difference* range to determine whether the corresponding values in the *Amount* range contribute to the sum.



You can also use the following array formula to return the sum of the values in the *Amount* range, where the corresponding value in the *Difference* range is negative:

```
{=SUM(IF(Difference<0,Amount))}
```

Summing Values Based on a Text Comparison

The following formula returns the total invoice amounts for the Oregon office:

```
=SUMIF(Office,"=Oregon",Amount)
```

Using the equal sign is optional. The following formula has the same result:

```
=SUMIF(Office,"Oregon",Amount)
```

To sum the invoice amounts for all offices *except* Oregon, use this formula:

```
=SUMIF(Office,"<>Oregon",Amount)
```

Summing Values Based on a Date Comparison

The following formula returns the total invoice amounts that have a due date after June 1, 1999:

```
=SUMIF(DateDue,">="&DATE(1999,6,1),Amount)
```

Notice that the second argument for the SUMIF function is an expression. The expression uses the DATE function, which returns a date. Also, the comparison operator, enclosed in quotation marks, is concatenated (using the & operator) with the result of the DATE function.

The formula that follows returns the total invoice amounts that have a future due date (including today):

```
=SUMIF(DateDue,">="&TODAY(),Amount)
```

Conditional Sums Using Multiple Criteria

The examples in the preceding section all used a single comparison criterion. The examples in this section involve summing cells based on multiple criteria. Because the SUMIF function does not work with multiple criteria, you need to resort to using an array formula. Figure 7-15 shows the sample worksheet again, for your reference.

	A	B	C	D	E	F
1	InvoiceNum	Office	Amount	DateDue	Today	Difference
2	AG-0145	Oregon	\$5,000.00	3-Apr	6-May	-33
3	AG-0189	California	\$450.00	21-Apr	6-May	-15
4	AG-0220	Washington	\$3,211.56	30-Apr	6-May	-6
5	AG-0310	Oregon	\$250.00	2-May	6-May	-4
6	AG-0355	Washington	\$125.50	6-May	6-May	0
7	AG-0409	Washington	\$3,000.00	12-May	6-May	6
8	AG-0581	Oregon	\$2,100.00	25-May	6-May	19
9	AG-0600	Oregon	\$335.39	25-May	6-May	19
10	AG-0602	Washington	\$65.00	30-May	6-May	24
11	AG-0633	California	\$250.00	1-Jun	6-May	26
12	TOTAL		\$14,787.45			36
13						

Figure 7-15: This worksheet demonstrates summing based on multiple criteria.

Using And Criteria

Suppose you want to get a sum of the invoice amounts that are past due, *and* associated with the Oregon office. In other words, the value in the *Amount* range will be summed only if both of the following criteria are met:

- ◆ The corresponding value in the *Difference* range is negative.
- ◆ The corresponding text in the *Office* range is “Oregon.”

The following array formula does the job:

```
{=SUM((Difference<0)*(Office="Oregon")*Amount)}
```

This formula creates two new arrays (in memory):

- ◆ A Boolean array that consists of TRUE if the corresponding *Difference* value is less than zero; FALSE otherwise
- ◆ A Boolean array that consists of TRUE if the corresponding *Office* value equals “Oregon”; FALSE otherwise

Multiplying Boolean values results in the following:

```
TRUE * TRUE = 1
TRUE * FALSE = 0
FALSE * FALSE = 0
```

Therefore, the corresponding *Amount* value returns non-zero only if the corresponding values in the memory arrays are both TRUE. The result produces a sum of the *Amount* values that meet the specified criteria.



You may think that you can rewrite the previous array function as follows, using the SUMPRODUCT function to perform the multiplication and addition:

```
=SUMPRODUCT((Difference<0),(Office="Oregon"),Amount)
```

For some reason, the SUMPRODUCT function does not handle Boolean values properly, so the formula does not work. The following formula, which multiplies the Boolean values by 1, *does* work:

```
=SUMPRODUCT(1*(Difference<0),1*(Office="Oregon"),Amount)
```

Using Or Criteria

Suppose you want to get a sum of past-due invoice amounts, *or* ones associated with the Oregon office. In other words, the value in the *Amount* range will be summed if either of the following criteria is met:

- ◆ The corresponding value in the *Difference* range is negative.
- ◆ The corresponding text in the *Office* range is “Oregon.”

The following array formula does the job:

```
{=SUM(IF((Office="Oregon")+(Difference<0),1,0)*Amount)}
```

A plus sign (+) joins the conditions; you can include more than two conditions.

Using And and Or Criteria

As you might expect, things get a bit tricky when your criteria consists of both And and Or operations. For example, you might want to sum the values in the *Amount* range when the following conditions are met:

- ◆ The corresponding value in the *Difference* range is negative.

- ◆ The corresponding text in the *Office* range is “Oregon” or “California.”

Notice that the second condition actually consists of two conditions, joined with Or. The following array formula does the trick:

```
{=SUM((Difference<0)*IF((Office="Oregon")+  
(Office="California"),1)*Amount)}
```

Using VBA Functions to Count and Sum

Some types of counting and summing tasks are simply impossible using Excel’s built-in functions, or even array formulas. Fortunately, Excel has a powerful tool that enables you to create custom functions. Excel’s Visual Basic for Applications (VBA) language can usually come to the rescue when all else fails.

I devote Part IV of this book to VBA. Chapter 25 contains several custom functions relevant to counting and summing. I briefly describe these functions here:

- ◆ **COUNTBETWEEN**: Returns the number of cells that contain a value between two specified values.
- ◆ **COUNTVISIBLE**: Returns the number of visible cells in a range.
- ◆ **DATATYPE**: Returns a string that describes the type of data in a cell. This function enables you to count cells that contain dates (something not normally possible).
- ◆ **ISBOLD**, **ISITALIC**, **FILLCOLOR**: These functions return TRUE if a specified cell has a particular type of formatting (bold, italic, or a specific color). You can use these functions to sum or count cells based on their formatting.
- ◆ **NUMBERFORMAT**: Returns the number format string for a cell. This function enables you to count or sum cells based on their number format.
- ◆ **SUMVISIBLE**: Returns the sum of the visible cells in a range.

Summary

This chapter provided many examples of functions and formulas that count or sum cells meeting certain criteria. Many of these formulas are array formulas.

The next chapter covers using formulas to look up specific information in tables or ranges of data.

Chapter 8

Lookups

IN THIS CHAPTER

- ◆ An introduction to formulas that look up values in a table
- ◆ An overview of the worksheet functions used to perform lookups
- ◆ Basic lookup formulas
- ◆ More sophisticated lookup formulas

THIS CHAPTER DISCUSSES VARIOUS techniques that you can use to look up a value in a table. Excel has three functions (LOOKUP, VLOOKUP, and HLOOKUP) designed for this task, but you may find that these functions don't quite cut it. This chapter provides many lookup examples, including alternative techniques that go well beyond Excel's normal lookup capabilities.

What Is a Lookup Formula?

A lookup formula essentially returns a value from a table (in a range) by looking up another value. A common telephone directory provides a good analogy. If you want to find a person's telephone number, you first locate the name (look it up), and then retrieve the corresponding number.

Figure 8-1 shows a simple worksheet that uses several lookup formulas. This worksheet contains a table of employee data (named *EmpData*), beginning in row 9. When you enter a name into cell C2, lookup formulas in D2:G2 retrieve the matching information from the table. The following lookup formulas use the VLOOKUP function:

```
D2      =VLOOKUP(C2,EmpData,2,FALSE)
```

```
E2      =VLOOKUP(C2,EmpData,3,FALSE)
```

```
F2      =VLOOKUP(C2,EmpData,4,FALSE)
```

```
G2      =VLOOKUP(C2,EmpData,5,FALSE)
```

This particular example uses four formulas to return information from the *EmpData* range. In many cases, you'll only want a single value from the table, so use only one formula.

	A	B	C	D	E	F	G
1			Last Name	First Name	Department	Extension	Date Hired
2	Enter a Name -->		Davis	Rita	Administration	4441	09/19/98
3							
4							
5							
6							
7							
8							
9			Last Name	First Name	Department	Extension	Date Hired
10			Allen	Yolanda	Sales	4465	03/05/98
11			Baker	Nancy	Operations	4498	11/24/98
12			Bunnel	Ken	Marketing	4422	05/12/98
13			Charles	Larry	Administration	3988	06/30/97
14			Cramden	Oliver	Administration	4421	01/04/01
15			Davis	Rita	Administration	4441	09/19/98
16			Dunwell	James	Operations	3321	04/16/99
17			Ellis	Pamela	Data Processing	3398	02/01/00
18			Endow	Ed	Data Processing	4448	02/01/99

Figure 8-1: Lookup formulas in row 2 look up the information for the employee name in cell C2.

Functions Relevant to Lookups

Several Excel functions are useful when writing formulas to look up information in a table. Table 8-1 lists and describes these functions.

TABLE 8-1 FUNCTIONS USED IN LOOKUP FORMULAS

Function	Description
CHOOSE	Returns a specific value from a list of values (up to 29) supplied as arguments.
HLOOKUP	Horizontal lookup. Searches for a value in the top row of a table and returns a value in the same column from a row you specify in the table.
INDEX	Returns a value (or the reference to a value) from within a table or range.
LOOKUP	Returns a value either from a one-row or one-column range.
MATCH	Returns the relative position of an item in a range that matches a specified value.
OFFSET	Returns a reference to a range that is a specified number of rows and columns from a cell or range of cells.
VLOOKUP	Vertical lookup. Searches for a value in the first column of a table and returns a value in the same row from a column you specify in the table.

The examples in this chapter use the functions listed in Table 8-1.

Basic Lookup Formulas

You can use Excel's basic lookup functions to search a column or row for a lookup value to return another value as a result. Excel provides three basic lookup functions: HLOOKUP, VLOOKUP, and LOOKUP. The MATCH and INDEX functions are often used together to return a cell or relative cell reference for a lookup value.

The VLOOKUP Function

The VLOOKUP function looks up the value in the first column of the lookup table and returns the corresponding value in a specified table column. The lookup table is arranged vertically. The syntax for the VLOOKUP function is:

```
VLOOKUP(lookup_value,table_array,col_index_num,range_lookup)
```

The VLOOKUP function's arguments are as follows:

- ◆ *lookup_value*: The value to be looked up in the first column of the lookup table.
- ◆ *table_array*: The range that contains the lookup table.
- ◆ *col_index_num*: The column number within the table from which the matching value is returned.
- ◆ *range_lookup*: Optional. If TRUE or omitted, an approximate match is returned (if an exact match is not found, the next largest value that is less than *lookup_value* is returned). If FALSE, VLOOKUP will search for an exact match. If VLOOKUP cannot find an exact match, the function returns #N/A.



If the *range_lookup* argument is TRUE or omitted, the first column of the lookup table must be in ascending order. If *lookup_value* is smaller than the smallest value in the first column of *table_array*, VLOOKUP returns #N/A. If the *range_lookup* argument is FALSE, the first column of the lookup table need not be in ascending order. If an exact match is not found, the function returns #N/A.



Although not indicated in the online help, if the *lookup_value* argument is text, it can include wildcard characters * and ?.

The classic example of a lookup formula involves an income tax rate schedule (see Figure 8-2). The tax rate schedule shows the income tax rates for various income levels. The following formula (in cell B3) returns the tax rate for the income in cell B2:

```
=VLOOKUP(B2,D2:F7,3)
```

Figure 8-2: Using VLOOKUP to look up a tax rate



You can access the workbook shown in Figure 8-2 on the companion CD-ROM.

The lookup table resides in a range that consists of three columns (D2:F7). Because the last argument for the VLOOKUP function is 3, the formula returns the corresponding value in the third column of the lookup table.

Note that an exact match is not required. If an exact match is not found in the first column of the lookup table, the VLOOKUP function uses the next largest value that is less than the lookup value. In other words, the function uses the row in which the value you want to look up is greater than or equal to the row value, but less than the value in the next row. In the case of a tax table, this is exactly what you want to happen.

The HLOOKUP Function

The HLOOKUP function works just like the VLOOKUP function, except that the lookup table is arranged horizontally instead of vertically. The HLOOKUP function looks up the value in the first row of the lookup table and returns the corresponding value in a specified table row.

The syntax for the HLOOKUP function is:

```
HLOOKUP(lookup_value,table_array,row_index_num,range_lookup)
```

The HLOOKUP function's arguments are as follows:

- ◆ *lookup_value*: The value to be looked up in the first row of the lookup table.
- ◆ *table_array*: The range that contains the lookup table.
- ◆ *row_index_num*: The row number within the table from which the matching value is returned.
- ◆ *range_lookup*: Optional. If TRUE or omitted, an approximate match is returned (if an exact match is not found, the next largest value less than *lookup_value* is returned). If FALSE, VLOOKUP will search for an exact match. If VLOOKUP cannot find an exact match, the function returns #N/A.



Although not indicated in the online help, if the *lookup_value* argument is text, it can include wildcard characters * and ?.

Figure 8-3 shows the tax rate example with a horizontal lookup table (in the range E1:J3). The formula in cell B3 is:

```
=HLOOKUP(B2,E1:J3,3)
```

	A	B	C	D	E	F	G	H	I	J
1				Income is Greater Than or Equal To...	\$0	\$2,651	\$27,301	\$58,501	\$131,801	\$284,701
2	Enter Income:	\$21,566		But Less Than...	\$2,650	\$27,300	\$58,500	\$131,800	\$284,700	
3	The Tax Rate is:	28.00%		Tax Rate	15.00%	28.00%	31.00%	36.00%	39.60%	45.25%
4										
5										
6										

Figure 8-3: Using HLOOKUP to look up a tax rate

The LOOKUP Function

The LOOKUP function has the following syntax:

```
LOOKUP(lookup_value,lookup_vector,result_vector)
```

The function's arguments are as follows:

- ◆ *lookup_value*: The value to be looked up in the *lookup_vector*.
- ◆ *lookup_vector*: A single-column or single-row range that contains the values to be looked up. These values must be in ascending order.
- ◆ *result_vector*: The single-column or single-row range that contains the values to be returned. It must be the same size as the *lookup_vector*.

The LOOKUP function looks in a one-row or one-column range (*lookup_vector*) for a value (*lookup_value*) and returns a value from the same position in a second one-row or one-column range (*result_vector*).



Values in the *lookup_vector* must be in ascending order. If *lookup_value* is smaller than the smallest value in *lookup_vector*, LOOKUP returns #N/A.



The online help also lists an “array” syntax for the LOOKUP function. This alternative syntax is included for compatibility with other spreadsheet products. In general, you can use the VLOOKUP or HLOOKUP functions rather than the array syntax.

Figure 8-4 shows the tax table again. This time, the formula in cell B3 uses the LOOKUP function to return the corresponding tax rate. The formula in B3 is:

```
=LOOKUP(B2,D2:D7,F2:F7)
```



If the values in the first column are not arranged in ascending order, the LOOKUP function may return an incorrect value.

	A	B	C	D	E	F
1				Income is Greater Than or Equal To...	But Less Than...	Tax Rate
2	Enter Income:	\$123,409		\$0	\$2,650	15.00%
3	The Tax Rate is:	36.00%		\$2,651	\$27,300	28.00%
4				\$27,301	\$58,500	31.00%
5				\$58,501	\$131,800	36.00%
6				\$131,801	\$284,700	39.60%
7				\$284,701		45.25%
8						
9						

Figure 8-4: Using LOOKUP to look up a tax rate

Note that LOOKUP (as opposed to VLOOKUP) requires two range references (a range to be looked in, and a range that contains result values). VLOOKUP, on the other hand, uses a single range for the lookup table and the third argument determines which column to use for the result. This argument, of course, can consist of a cell reference.

Combining the MATCH and INDEX Functions

The MATCH and INDEX functions are often used together to perform lookups. The MATCH function returns the relative position of a cell in a range that matches a specified value. The syntax for MATCH is:

MATCH(lookup_value,lookup_array,match_type)

The MATCH function's arguments are as follows:

- ◆ *lookup_value*: The value you want to match in *lookup_array*. If *match_type* is 0 and the *lookup_value* is text, this argument can include wildcard characters "*" and "?"
- ◆ *lookup_array*: The range being searched
- ◆ *match_type*: An integer (-1, 0, or 1) that specifies how the match is determined



If *match_type* is 1, MATCH finds the largest value less than or equal to *lookup_value* (*lookup_array* must be in ascending order). If *match_type* is 0, MATCH finds the first value exactly equal to *lookup_value*. If *match_type* is -1, MATCH finds the smallest value greater than or equal to *lookup_value* (*lookup_array* must be in descending order). If you omit *match_type*, it is assumed to be 1.

The INDEX function returns a cell from a range. The syntax for the INDEX function is:

```
INDEX(array, row_num, column_num)
```

The INDEX function's arguments are as follows:

- ◆ *array*: A range
- ◆ *row_num*: A row number within *array*
- ◆ *col_num*: A column number within *array*



If *array* contains only one row or column, the corresponding *row_num* or *column_num* argument is optional.

Figure 8-5 shows a worksheet with dates, day names, and amounts in columns D, E, and F. When you enter a date in cell B1, the following formula (in cell B2) searches the dates in column D and returns the corresponding amount from column F. The formula in B2 is:

```
=INDEX(F2:F21, MATCH(B1, D2:D21, 0))
```

	A	B	C	D	E	F
1	Date:	1/19/2001		Date	Weekday	Amount
2	Amount:	100		1/1/2001	Monday	23
3				1/2/2001	Tuesday	179
4				1/3/2001	Wednesday	149
5				1/4/2001	Thursday	196
6				1/5/2001	Friday	131
7				1/6/2001	Monday	179
8				1/8/2001	Tuesday	134
9				1/10/2001	Wednesday	179
10				1/11/2001	Thursday	193
11				1/12/2001	Friday	191
12				1/15/2001	Monday	176
13				1/16/2001	Tuesday	189
14				1/17/2001	Wednesday	163
15				1/18/2001	Thursday	121
16				1/19/2001	Friday	100
17				1/22/2001	Monday	109
18				1/23/2001	Tuesday	151
19				1/24/2001	Wednesday	138
20				1/25/2001	Thursday	114
21				1/26/2001	Friday	156
22						

Figure 8-5: Using the INDEX and MATCH functions to perform a lookup

When a Blank Is Not a Zero

Excel's lookup functions treat empty cells in the result range as zeros. The worksheet in the accompanying figure contains a two-column lookup table and this formula looks up the name in cell B1 and returns the corresponding amount:

```
=VLOOKUP(B1,D2:E8,2)
```

Note that the Amount cell for Charlie is blank, but the formula returns a 0.

	A	B	C	D	E	F
1	Name:	Charlie		Name	Amount	
2	Amount:		0	Bob	45	
3				Charlie		
4				David	0	
5				Frank	32	
6				George	9	
7				Harry	0	
8				Mike	1	
9						
10						

If you need to distinguish zeros from blank cells, you must modify the lookup formula by adding an IF function to check if the length of the returned value is 0. When the looked up value is blank, the length of the return value is 0. In all other cases, the length of the returned value is non-zero. The following formula displays an empty string (a blank) whenever the length of the looked-up value is zero, and the actual value whenever the length is anything but zero:

```
=IF(LEN(VLOOKUP(B1,D2:E8,2))=0,"",(VLOOKUP(B1,D2:E8,2)))
```

To understand how this works, start with the MATCH function. This function searches the range D2:D21 for the date in cell B1. It returns the relative row number where the date is found. This value is then used as the second argument for the INDEX function. The result is the corresponding value in F2:F21.

Specialized Lookup Formulas

You can use some additional types of lookup formulas to perform more specialized lookups. For instance, you can look up an exact value, search in another column besides the first in a lookup table, perform a case-sensitive lookup, return a value from among multiple lookup tables, and perform other specialized and complex lookups.

Looking Up an Exact Value

As demonstrated in the previous examples, VLOOKUP and HLOOKUP don't necessarily require an exact match between the value to be looked up and the values in the lookup table. An example is looking up a tax rate in a tax table. In some cases, you may require a perfect match. For example, when looking up an employee number, you would probably require a perfect match for the number.

To look up an exact value only, use the VLOOKUP (or HLOOKUP) function with the optional fourth argument set to FALSE.

Figure 8-6 shows a worksheet with a lookup table that contains employee numbers (column C) and employee names (column D). The lookup table is named *EmpList*. The formula in cell B2, which follows, looks up the employee number entered in cell B1 and returns the corresponding employee name:

```
=VLOOKUP(B1,EmpList,2,FALSE)
```

	A	B	C	D
1	Employee No.:	999	Employee Number	Employee Name
2	Employee Name:	#N/A	873	Charles K. Barkley
3			1109	Francis Jenkins
4			1549	James Brackman
5			1334	Linda Harper
6			1643	Louise Victor
7			1101	Melinda Hindquest
8			1873	Michael Orenthal
9			983	Peter Yates
10			972	Sally Rice
11			1398	Walter Franklin
12				
13				
14				

Figure 8-6: This lookup table requires an exact match.

Because the last argument for the VLOOKUP function is FALSE, the function returns a value only if an exact match is found. If the value is not found, the formula returns #N/A. This, of course, is exactly what you want to happen because returning an approximate match for an employee number makes no sense. Also, notice that the employee numbers in column C are not in ascending order. If the last argument for VLOOKUP is FALSE, the values need not be in ascending order.



If you prefer to see something other than #N/A when the employee number is not found, you can use an IF function to test for the #N/A result (using the ISNA function) and substitute a different string. The following formula displays the text "Not Found" rather than #N/A:

```
=IF(ISNA(VLOOKUP(B1,EmpList,2,FALSE)),"Not Found",  
VLOOKUP(B1,EmpList,2,FALSE))
```

Looking Up a Value to the Left

The VLOOKUP function always looks up a value in the first column of the lookup range. But what if you want to look up a value in a column other than the first column? It would be helpful if you could supply a negative value for the third argument for VLOOKUP – but you can't.

Figure 8-7 illustrates the problem. Suppose you want to look up the batting average (column B, in a range named *Averages*) of a player in column C (in a range named *Players*). The player you want data for appears in a cell named *LookupValue*. The VLOOKUP function won't work because the data is not arranged correctly. One option is to rearrange your data, but sometimes that's not possible.

	A	B	C	D	E	F	G	H	I
1	At Bats	Average	Player		Player to lookup:	Darr			
2	12	0.333	Arias						
3	41	0.390	Darr		Average	0.390	<-- LOOKUP		
4	24	0.333	Davis		At Bats:	41	<-- LOOKUP		
5	25	0.160	Gomez						
6	23	0.217	Gonzalez		Average	0.390	<-- INDEX and MATCH		
7	30	0.300	Gwynn		At Bats:	41	<-- INDEX and MATCH		
8	0	0.000	Henderson						
9	51	0.333	Jackson						
10	43	0.186	Klesko						
11	36	0.139	Kotsay						
12	9	0.333	Magadan						
13	16	0.313	Mendez						
14	44	0.341	Nevin						
15	14	0.266	Perez						
16	28	0.321	Trammell						
17									

Figure 8-7: The VLOOKUP function can't look up a value in column B, based on a value in column C.

One solution is to use the LOOKUP function, which requires two range arguments. The following formula (in cell F3) returns the batting average from column B of the player name contained in the cell named *LookupValue*:

```
=LOOKUP(LookupValue,Players,Averages)
```

Using the VLOOKUP function requires that the lookup range (in this case, the *Players* range) is in ascending order. In addition to this limitation, the formula suffers from a slight problem: If you enter a nonexistent player (in other words, the *LookupValue* cell contains a value not found in the *Players* range), the formula returns an erroneous result.

A better solution uses the INDEX and MATCH functions. The formula that follows works just like the previous one, except that it returns #N/A if the player is not found. Another advantage is that the player names need not be sorted.

```
=INDEX(Averages,MATCH(LookupValue,Players,0))
```



You can access a workbook that demonstrates both of the formulas in this section on the companion CD-ROM.

Performing a Case-Sensitive Lookup

Excel's lookup functions (LOOKUP, VLOOKUP, and HLOOKUP) are not case sensitive. For example, if you write a lookup formula to look up the text *budget*, the formula considers any of the following a match: *BUDGET*, *Budget*, or *BuDgEt*.

Figure 8-8 shows a simple example. Range D2:D7 is named *Range1*, and range E2:E7 is named *Range2*. The word to be looked up appears in cell B1 (named *Value*).

	A	B	C	D	E	F
1	Word	DOG		Range1	Range2	
2	Result:	300		APPLE	100	
3				apple	200	
4				DOG	300	
5				dog	400	
6				CANDY	500	
7				candy	600	
8						
9						

Figure 8-8: Using an array formula to perform a case-sensitive lookup

The array formula that follows is in cell B2. This formula does a case-sensitive lookup in *Range1* and returns the corresponding value in *Range2*.

```
{=INDEX(Range2,MATCH(TRUE,EXACT(Value,Range1),0))}
```

The formula looks up the word *DOG* (uppercase) and returns 300. The following standard LOOKUP formula returns 400:

```
=LOOKUP(Value,Range1,Range2)
```



When entering an array formula, remember to use Ctrl+Alt+Enter.

Choosing among Multiple Lookup Tables

You can, of course, have any number of lookup tables in a worksheet. In some cases, your formula may need to decide which lookup table to use. Figure 8-9 shows an example.

	A	B	C	D	E	F	G	H	I	J	K
1	Sales Rep	Years	Sales	Comm. Rate	Commission		<3 Years Tenure			3+ Years Tenure	
2	Benson	2	120,000	7.00%	8,400		Amt Sold	Rate		Amt Sold	Rate
3	Davidson	1	210,921	7.00%	14,764		0	1.50%		0	2.00%
4	Ellison	1	100,000	7.00%	7,000		5,000	3.25%		50,000	6.25%
5	Gomez	2	87,401	6.00%	5,244		10,000	3.50%		100,000	7.25%
6	Hernandez	6	310,983	9.25%	28,766		20,000	5.00%		200,000	6.25%
7	Kelly	3	43,902	2.00%	878		50,000	6.00%		300,000	9.25%
8	Martin	2	121,021	7.00%	8,471		100,000	7.00%		500,000	10.00%
9	Oswald	3	908	2.00%	18		250,000	8.00%			
10	Reginald	1	0	1.50%	0						
11	Veras	4	359,832	9.25%	33,284						
12	Wilmington	4	502,983	10.00%	50,298						
13											

Figure 8-9: This worksheet demonstrates the use of multiple lookup tables.

This workbook calculates sales commission and contains two lookup tables: G3:H9 (named *Table1*) and J3:K8 (named *Table2*). The commission rate for a particular sales representative depends on two factors: the sales rep's years of service (column B) and the amount sold (column C). Column D contains formulas that look up the commission rate from the appropriate table. For example, the formula in cell D2 is:

```
=VLOOKUP(C2,IF(B2<3,Table1,Table2),2)
```

The second argument for the VLOOKUP function consists of an IF formula that uses the value in column B to determine which lookup table to use.

The formula in column E simply multiplies the sales amount in column C by the commission rate in column D. The formula in cell E2, for example, is:

```
=C2*D2
```



You can access the workbook shown in Figure 8-9 on the companion CD-ROM.

Determining Letter Grades for Test Scores

A common use of a lookup table is to assign letter grades for test scores. Figure 8-10 shows a worksheet with student test scores. The range E2:F6 (named *GradeList*) displays a lookup table used to assign a letter grade to a test score.

Student	Score	Grade	Score	Grade
Adams	36	F	0	F
Baker	68	D	40	D
Camden	50	D	70	C
Dailey	77	C	80	B
Gomez	92	A	90	A
Hernandez	100	A		
Jackson	74	C		
Maplethorpe	45	D		
Paulson	60	D		
Ramirez	89	B		
Sosa	99	A		
Thompson	91	A		
Wilson	59	D		

Figure 8-10: Looking up letter grades for test scores



The companion CD-ROM contains a workbook that demonstrates both formulas in this section.

Column C contains formulas that use the VLOOKUP function and the lookup table to assign a grade based on the score in column B. The formula in C2, for example, is:

```
=VLOOKUP(B2,GradeList,2)
```

When the lookup table is small (as in the example shown in Figure 8-10), you can use a literal array in place of the lookup table. The formula that follows, for example, returns a letter grade without using a lookup table. Rather, the information in the lookup table is hard-coded into a literal array. See Chapter 14 for more information about literal arrays.

```
=VLOOKUP(B2,{0,"F";40,"D";70,"C";80,"B";90,"A"},2)
```

Another approach, which uses a more legible formula, is to use the LOOKUP function with two array arguments:

```
=LOOKUP(B2,{0,40,70,80,90},{ "F", "D", "C", "B", "A" })
```

Calculating a Grade Point Average

A student's grade point average (GPA) is a numerical measure of the average grade received for classes taken. This discussion assumes a letter grade system, in which each letter grade is assigned a numeric value (A=4, B=3, C=2, D=1, and F=0). The GPA comprises an average of the numeric grade values, weighted by the credit hours of the course. A one-hour course, for example, receives less weight than a three-hour course. The GPA ranges from 0 (all Fs) to 4.00 (all As).

Figure 8-11 shows a worksheet with information for a student. This student took five courses, for a total of 13 credit hours. Range B2:B6 is named *CreditHours*. The grades for each course appear in column C (Range C2:C6 is named *Grades*). Column D uses a lookup formula to calculate the grade value for each course. The lookup formula in cell D2, for example, follows. This formula uses the lookup table in G2:H6 (named *GradeTable*).

```
=VLOOKUP(C2,GradeTable,2,FALSE)
```

	A	B	C	D	E	F	G	H
1	Course	Credit Hrs	Grade	Grade Val	Weighted Val		GradeTable	
2	Psych 101	3	A	4	12		A	4
3	PhysEd	2	C	2	4		B	3
4	PoliSci 101	4	B	3	12		C	2
5	IndepStudy	1	A	4	4		D	1
6	IntroMath	3	A	4	12		F	0
7								
8		GPA: 3.38	<- Requires multiple formulas and lookup table					
9								
10								

Figure 8-11: Using multiple formulas to calculate a GPA

Formulas in column E calculate the weighted values. The formula in E2 is:

```
=D2*B2
```

Cell B8 computes the GPA using the following formula:

```
=SUM(E2:E6)/SUM(B2:B6)
```

The preceding formulas work fine, but you can streamline the GPA calculation quite a bit. In fact, you can use a single array formula to make this calculation and avoid using the lookup table and the formulas in columns D and E. This array formula does the job:

```
{=SUM((MATCH(Grades,{"F","D","C","B","A"},0)-1)*CreditHours)
/SUM(CreditHours)}
```



You can access a workbook that demonstrates both the multiformula and the array formula techniques on the companion CD-ROM.

Performing a Two-Way Lookup

Figure 8-12 shows a worksheet with a table that displays product sales by month. To retrieve sales for a particular month and product, the user enters a month in cell B1 and a product name in cell B2.

	A	B	C	D	E	F	G	H
1	Month:	July		Widgets	Sprockets	Snapaholytes	Combined	
2	Product:	Sprockets		January	2,892	1,771	4,718	9,381
3				February	3,380	4,711	2,615	10,706
4	Month Offset:	8		March	3,744	3,223	5,312	12,279
5	Product Offset:	3		April	3,221	2,438	1,108	6,767
6	Sales:	3,337		May	4,839	1,999	1,994	8,832
7				June	3,767	5,140	3,830	12,737
8				July	5,467	3,337	3,232	12,036
9	Single-formula -->	3,337		August	3,154	4,895	1,607	9,656
10				September	1,718	2,040	1,563	5,321
11				October	1,548	1,061	2,590	5,199
12				November	5,083	3,558	3,960	12,601
13				December	5,753	2,839	3,013	11,605
14				Total	44,566	37,012	35,542	117,120
15								

Figure 8-12: This table demonstrates a two-way lookup.



The companion CD-ROM contains the workbook shown in Figure 8-12.

To simplify things, the worksheet uses the following named ranges:

Name	Refers To
Month	B1
Product	B2
Table	D1:H14
MonthList	D1:D14
ProductList	D1:H1

The following formula (in cell B4) uses the MATCH function to return the position of the *Month* within the *MonthList* range. For example, if the month is January, the formula returns 2 because January is the second item in the *MonthList* range (the first item is a blank cell, D1).

```
=MATCH(Month,MonthList,0)
```

The formula in cell B5 works similarly, but uses the *ProductList* range.

```
=MATCH(Product,ProductList,0)
```

The final formula, in cell B6, returns the corresponding sales amount. It uses the INDEX function with the results from cells B4 and B5.

```
=INDEX(Table,B4,B5)
```

You can, of course, combine these formulas into a single formula, as shown here:

```
=INDEX(Table,MATCH(Month,MonthList,0),MATCH(Product,ProductList,0))
```



If you use Excel 97 or later, you can use the Lookup Wizard add-in to create this type of formula (see Figure 8-13). The Lookup Wizard add-in is distributed with Excel.

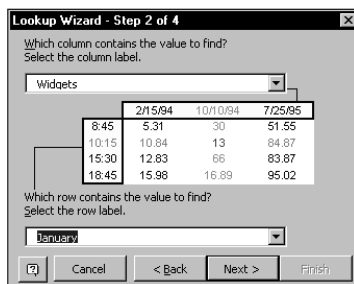


Figure 8-13: The Lookup Wizard add-in can create a formula that performs a two-way lookup.



Another way to accomplish a two-way lookup is to provide a name for each row and column of the table. A quick way to do this is to select the table and use Insert → Name → Create. After creating the names, you can use a simple formula such as:

```
= Sprockets July
```

This formula, which uses the range intersection operator (a space), returns July sales for Sprockets. See Chapter 3 for details.

Performing a Two-Column Lookup

Some situations may require a lookup based on the values in two columns. Figure 8-14 shows an example.

	A	B	C	D	E	F
1	Make:	Jeep		Make	Model	Code
2	Model:	Grand Cherokee		Chevy	Blazer	C-094
3	Code:	J-701		Chevy	Tahoe	C-823
4				Ford	Explorer	F-772
5				Ford	Expedition	F-229
6				Isuzu	Rodeo	I-897
7				Isuzu	Trooper	I-900
8				Jeep	Cherokee	J-983
9				Jeep	Grand Cherokee	J-701
10				Nissan	Pathfinder	N-231
11				Toyota	4Runner	T-871
12				Toyota	Land Cruiser	T-981
13						

Figure 8-14: This workbook performs a lookup using information in two columns (D and E).



The workbook shown in Figure 8-14 also appears on the companion CD-ROM.

The lookup table contains automobile makes and models, and a corresponding code for each. The worksheet uses named ranges, as shown here:

F2:F12 *Code*

B1 *Make*

B2 *Model*

D2:D12 *Range1*

E2:E12 *Range2*

The following array formula displays the corresponding code for an automobile make and model:

```
{=INDEX(Code,MATCH(Make&Model,Range1&Range2,0))}
```

This formula works by concatenating the contents of *Make* and *Model*, and then searching for this text in an array consisting of the concatenated corresponding text in *Range1* and *Range2*.

Determining the Address of a Value within a Range

Most of the time, you want your lookup formula to return a value. You may, however, need to determine the cell address of a particular value within a range. For example, Figure 8-15 shows a worksheet with a range of numbers that occupy a single column (named *Data*). Cell B1, which contains the value to look up, is named *Target*.

	A	B	C	D
1	Target:	24	Data	
2	Address:	\$C\$9	74	
3			62	
4			60	
5			44	
6			50	
7			41	
8			77	
9			24	
10			55	
11			30	
12			12	
13			21	
14			7	
15			1	
16			22	
17			53	
18			36	
19			18	
20			68	
21				

Figure 8-15: The formula in cell B2 returns the address in the *Data* range for the value in cell B1.

The formula in cell B2, which follows, returns the address of the cell in the *Data* range that contains the *Target* value:

```
=ADDRESS(ROW(Data)+MATCH(Target,Data,0)-1,COLUMN(Data))
```

If the *Data* range occupies a single row, use this formula to return the address of the *Target* value:

```
=ADDRESS(ROW(Data),COLUMN(Data)+MATCH(Target,Data,0)-1)
```



The companion CD-ROM contains the workbook shown in Figure 8-15.

If the *Data* range contains more than one instance of the *Target* value, the address of the first occurrence is returned. If the *Target* value is not found in the *Data* range, the formula returns #N/A.

Looking Up a Value Using the Closest Match

The VLOOKUP and HLOOKUP functions are useful in the following situations:

- ◆ You need to identify an exact match for a target value. Use FALSE as the function's fourth argument.
- ◆ You need to locate an approximate match. If the function's fourth argument is TRUE or omitted and an exact match is not found, the next largest value less than the lookup value is returned.

But what if you need to look up a value based on the *closest* match? Neither VLOOKUP nor HLOOKUP can do the job.

Figure 8-16 shows a worksheet with student names in column A and values in column B. Range B2:B20 is named *Data*. Cell E2, named *Target*, contains a value to search for in the *Data* range. Cell E3, named *ColOffset*, contains a value that represents the column offset from the *Data* range.



You can access the workbook shown in Figure 8-16 on the companion CD-ROM.

The array formula that follows identifies the closest match to the *Target* value in the *Data* range, and returns the names of the corresponding student in column A (i.e., the column with an offset of -1). The formula returns Leslie (with a matching value of 8,000, which is the one closest to the *Target* value of 8,025).

	A	B	C	D	E	F
1	Student	Data				
2	Ann	9,101		Target Value -->	8025	
3	Betsy	8,873		Column Offset -->	-1	
4	Chuck	6,000				
5	David	9,820		Student:	Leslie	
6	George	10,500				
7	Hilda	3,500				
8	James	12,873				
9	John	5,867				
10	Keith	8,989				
11	Leslie	8,000				
12	Michelle	1,124				
13	Nora	9,099				
14	Paul	6,800				
15	Peter	5,509				
16	Rasmusen	5,460				
17	Sally	8,400				
18	Theresa	7,777				
19	Violet	3,600				
20	Wendy	5,400				
21						

Figure 8-16: This workbook demonstrates how to perform a lookup using the closest match.

```
{=INDIRECT(ADDRESS(ROW(Data)+MATCH(MIN(ABS(Target-Data)),
ABS(Target-Data),0)-1,COLUMN(Data)+ColOffset))}
```

If two values in the *Data* range are equidistant from the *Target* value, the formula uses the first one in the list.

The value in *ColOffset* can be negative (for a column to the left of *Data*), positive (for a column to the right of *Data*), or 0 (for the actual closest match value in the *Data* range).

To understand how this formula works, you need to understand the INDIRECT function. This function's first argument is a text string in the form of a cell reference (or a reference to a cell that contains a text string). In this example, the text string is created by the ADDRESS function, which accepts a row and column reference and returns a cell address.

Looking Up a Value Using Linear Interpolation

Interpolation refers to the process of estimating a missing value by using existing values. To illustrate, refer to Figure 8-17. Column D contains a list of values (named *x*) and column E contains corresponding values (named *y*).

The worksheet also contains a chart that depicts the relationship between the *x* range and the *y* range graphically. As you can see, there is an approximate linear relationship between the corresponding values in the *x* and *y* ranges: as *x* increases, so does *y*. Notice that the values in the *x* range are not strictly consecutive. For example, the *x* range doesn't contain the following values: 3, 6, 7, 14, 17, 18, and 19.

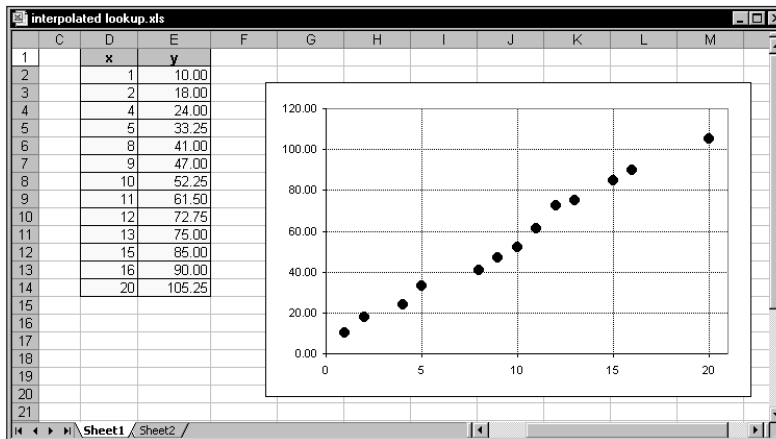


Figure 8-17: This workbook demonstrates a table lookup using linear interpolation.

You can create a lookup formula that looks up a value in the x range and returns the corresponding value from the y range. But what if you want to estimate the y value for a missing x value? A normal lookup formula does not return a very good result because it simply returns an existing y value (not an estimated y value). For example, the following formula looks up the value 3, and returns 18.00 (the value that corresponds to 2 in the x range):

```
=LOOKUP(3,x,y)
```

In such a case, you probably want to interpolate. In other words, because the lookup value (3) is halfway between existing x values (2 and 4), you want the formula to return a y value of 21.000 – a value halfway between the corresponding y values 18.00 and 24.00.

FORMULAS TO PERFORM A LINEAR INTERPOLATION

Figure 8-18 shows a worksheet with formulas in column B. The value to be looked up is entered into cell B1. The final formula, in cell B16, returns the result. If the value in B3 is found in the x range, the corresponding y value is returned. If the value in B3 is not found, the formula in B16 returns an estimated y value, obtained using linear interpolation.



The companion CD-ROM contains the workbook shown in Figure 8-18.

	A	B	C	D	E
1	X-value to look up:	3		x	y
2				1	10.00
3	Matching row:	2		2	18.00
4	Exact match?	FALSE		4	24.00
5				5	33.25
6	1st row:	2		8	41.00
7	2nd row:	3		9	47.00
8				10	52.25
9	1st x value:	2		11	61.50
10	2nd x value:	4		12	72.75
11				13	75.00
12	1st looked up y value:	18		15	85.00
13	2nd looked up y value:	24		16	90.00
14				20	105.25
15	Adjustment factor:	0.5			
16	Interpolated lookup:	21.00			
17					
18					

Figure 8-18: Column B contains formulas that perform a lookup using linear interpolation.

It's critical that the values in the x range appear in ascending order. If B1 contains a value less than the lowest value in x or greater than the largest value in x , the formula returns an error value. Table 8-2 lists and describes these formulas.

TABLE 8-2 FORMULAS FOR A LOOKUP USING LINEAR INTERPOLATION

Cell	Formula	Description
B3	=LOOKUP(B1, x, y)	Performs a standard lookup, and returns looked-up value in the x range.
B4	=B1=B3	Returns TRUE if the looked-up value equals the value to be looked up.
B6	=MATCH(B3, x, 0)	Returns the row number of the x range that contains the matching value.
B7	=IF(B4, B6, B6+1)	Returns the same row as the formula in B6 if an exact match is found. Otherwise, it adds 1 to the result in B6.
B9	=INDEX(x, B6)	Returns the x value that corresponds to the row in B6.
B10	=INDEX(x, B7)	Returns the x value that corresponds to the row in B7.
B12	=LOOKUP(B9, x, y)	Returns the y value that corresponds to the x value in B9.

Continued

TABLE 8-2 FORMULAS FOR A LOOKUP USING LINEAR INTERPOLATION (Continued)

Cell	Formula	Description
B13	=LOOKUP(B10,x,y)	Returns the y value that corresponds to the x value in B10.
B15	=IF(B4,0,(B1-B3)/(B10-B9))	Calculates an adjustment factor based on the difference between the x values.
B16	=B12+((B13-B12)*B15)	Calculates the estimated y value using the adjustment factor in B15.

COMBINING THE LOOKUP AND TREND FUNCTIONS

Another slightly different approach, which you may find preferable to performing lookup using linear interpolation, uses the LOOKUP and TREND functions. One advantage is that it requires only one formula (see Figure 8-19).

	A	B	C	D	E
1	X-value to look up:	3		x	y
2				1	10.00
3	y value:	20.27913834		2	18.00
4				4	24.00
5				5	33.25
6				8	41.00
7				9	47.00
8				10	52.25
9				11	61.50
10				12	72.75
11				13	75.00
12				15	85.00
13				16	90.00
14				20	105.25
15					

Figure 8-19: This worksheet uses a formula that utilizes the LOOKUP function and the TREND function.

The formula in cell B3 follows. This formula uses an IF function to make a decision. If an exact match is found in the x range, the formula returns the corresponding y value (using the LOOKUP function). If an exact match is not found, the formula uses the TREND function to return the calculated “best-fit” y value (it does not perform a linear interpolation).

```
=IF(B1=LOOKUP(B1,x,x),LOOKUP(INDEX(x,MATCH(LOOKUP(B1,x,x),x,0)),x,y),TREND(y,x,B1))
```

Summary

This chapter presented an overview of the functions available to perform table lookups. It included many formula examples demonstrating basic lookups, as well as not-so-basic lookups.

The next chapter discusses useful formulas for summarizing information contained in a database.

Chapter 9

Databases and Lists

IN THIS CHAPTER

- ◆ Basic information about using lists or worksheet databases
- ◆ Using AutoFiltering to filter a list using simple criteria
- ◆ Using advanced filtering to filter a list using more complex criteria
- ◆ Understanding how to create a criteria range for use with advanced filtering or database functions
- ◆ Using the SUBTOTAL function to summarize data in a list

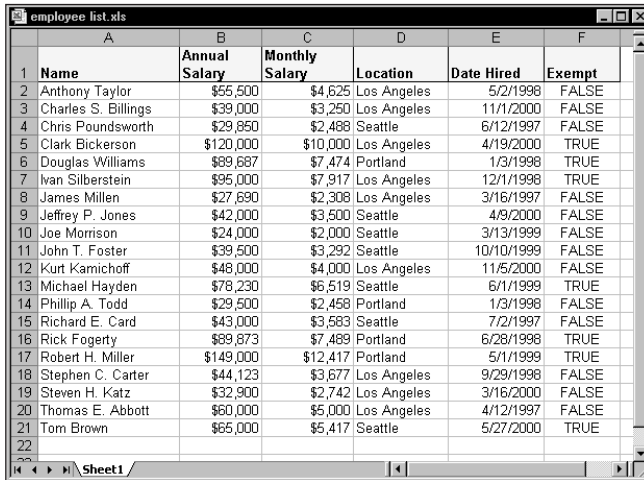
A **WORKSHEET DATABASE** (also known as a *list*) is an organized collection of information. More specifically, it consists of a row of headers (descriptive text), followed by additional rows of data comprised of values or text. This chapter provides an overview of Excel's worksheet database features, and presents some powerful formulas to help you get a handle on even the most unwieldy database.



Be aware that the term *database* is used loosely. An Excel worksheet database is more like a single table in a standard database. Unlike a conventional database, Excel does not allow you to set up a relationship between tables.

Worksheet Lists or Databases

Figure 9-1 shows an example of a worksheet list (or database). This particular list has its headers in row 1 and has 20 rows of data. Notice that the data consists of several different types: text, numerical values, dates, and logical values. Column C contains a formula that calculates the monthly salary from the value in column B.



	A	B	C	D	E	F
1	Name	Annual Salary	Monthly Salary	Location	Date Hired	Exempt
2	Anthony Taylor	\$55,500	\$4,625	Los Angeles	5/2/1998	FALSE
3	Charles S. Billings	\$39,000	\$3,250	Los Angeles	11/1/2000	FALSE
4	Chris Poundsworth	\$29,850	\$2,488	Seattle	6/12/1997	FALSE
5	Clark Bickerson	\$120,000	\$10,000	Los Angeles	4/19/2000	TRUE
6	Douglas Williams	\$89,687	\$7,474	Portland	1/3/1998	TRUE
7	Ivan Silberstein	\$95,000	\$7,917	Los Angeles	12/1/1998	TRUE
8	James Millen	\$27,690	\$2,308	Los Angeles	3/16/1997	FALSE
9	Jeffrey P. Jones	\$42,000	\$3,500	Seattle	4/9/2000	FALSE
10	Joe Morrison	\$24,000	\$2,000	Seattle	3/13/1999	FALSE
11	John T. Foster	\$39,500	\$3,292	Seattle	10/10/1999	FALSE
12	Kurt Kamichoff	\$46,000	\$4,000	Los Angeles	11/5/2000	FALSE
13	Michael Hayden	\$78,230	\$6,519	Seattle	6/1/1999	TRUE
14	Phillip A. Todd	\$29,500	\$2,458	Portland	1/3/1998	FALSE
15	Richard E. Card	\$43,000	\$3,583	Seattle	7/2/1997	FALSE
16	Rick Fogerty	\$89,873	\$7,489	Portland	6/28/1998	TRUE
17	Robert H. Miller	\$149,000	\$12,417	Portland	5/1/1999	TRUE
18	Stephen C. Carter	\$44,123	\$3,677	Los Angeles	9/29/1998	FALSE
19	Steven H. Katz	\$32,900	\$2,742	Los Angeles	3/16/2000	FALSE
20	Thomas E. Abbott	\$60,000	\$5,000	Los Angeles	4/12/1997	FALSE
21	Tom Brown	\$65,000	\$5,417	Seattle	5/27/2000	TRUE
22						

Figure 9-1: A simple worksheet list

People often refer to the columns in a list as *fields* and to the rows as *records*. Using this terminology, the list shown in the figure has six fields (Name, Annual Salary, Monthly Salary, Location, Date Hired, and Exempt) and 20 records.

The size of a list that you develop in Excel is limited by the size of a single worksheet. In other words, a list can have no more than 256 fields and can consist of no more than 65,535 records (one row contains the field names). A list of this size requires a great deal of memory and, even then, may prove impossible. At the other extreme, a list can consist of a single cell – not very useful, but still considered a list.



In versions prior to Excel 97, a worksheet contains only 16,384 rows.

Why are lists used? People use worksheet lists for a wide variety of purposes. For some users, a list simply keeps track of information (for example, customer information); others use lists to store data that ultimately appears in a report. Common list operations include:

- ◆ Entering data into the list
- ◆ Filtering the list to display only the rows that meet certain criteria
- ◆ Sorting the list
- ◆ Inserting formulas to calculate subtotals

- ◆ Creating formulas to calculate results on the list, filtered by certain criteria
- ◆ Creating a summary table of the data in the list (often done by using a pivot table)

When creating lists, it helps to plan the organization of your list information. This sidebar on designing lists has guidelines to help you create lists.

Designing a List

Although Excel is quite accommodating with regard to the information that is stored in a list, planning the organization of your list information is important, and makes the list easier to work with. Remember the following guidelines when you create lists:

- ◆ **Insert descriptive labels (one for each column) in the first row (the header row) of the list.** If you use lengthy labels, consider using the Wrap Text format so that you don't have to widen the columns.
- ◆ **Make sure that each column contains only one type of information.** For example, don't mix dates and text in a single column.
- ◆ **Consider using formulas that perform calculations on other fields in the same record.** If you use formulas that refer to cells outside the list, make these absolute references; otherwise, you get unexpected results when you sort the list.
- ◆ **Don't leave any empty rows within the list.** For list operations, Excel determines the list boundaries automatically, and an empty row signals the end of the list.
- ◆ **Keep the list on a worksheet by itself, to obtain the best results.** If you must place other information on the same worksheet as the list, place the information above or below the list. In other words, don't use the cells to the left or right of a list.
- ◆ **Freeze the first row.** Select the cell in the first column and first row of your table, then choose Window → Freeze Panes to make sure that you can see the headings when you scroll the list.
- ◆ **Preformat the entire column to ensure that the data has the same format.** For example, if a column contains dates, format the entire column with the same date format.

Using AutoFiltering

Filtering a list involves the process of hiding all rows in the list except those rows that meet some criteria that you specify. For example, if you have a list of customers, you can filter the list to show only those who live in Oregon. Filtering is a common (and very useful) technique.



Excel provides two ways to filter a list. AutoFiltering is useful for simple filtering criteria. Advanced filtering (discussed later in this chapter) is for more complex filtering.

AutoFiltering Basics

To use Excel's AutoFilter feature to filter a list, place the cell pointer anywhere within the list and then choose **Data** → **Filter** → **AutoFilter**. Excel determines the range occupied by the list, and adds drop-down arrows to the field names in the header row (as shown in Figure 9-2).

	A	B	C	D	E	F
1	Name	Annual Salary	Monthly Salary	Location	Date Hired	Exempt
2	Anthony Taylor	\$55,500	\$4,625	Los Angeles	5/2/1998	FALSE
3	Charles S. Billings	\$39,000	\$3,250	Los Angeles	11/1/2000	FALSE
4	Chris Poundsworth	\$29,850	\$2,488	Seattle	6/12/1997	FALSE
5	Clark Bickerson	\$120,000	\$10,000	Los Angeles	4/19/2000	TRUE
6	Douglas Williams	\$89,687	\$7,474	Portland	1/3/1996	TRUE
7	Ivan Silberstein	\$95,000	\$7,917	Los Angeles	12/1/1998	TRUE
8	James Millen	\$27,690	\$2,308	Los Angeles	3/16/1997	FALSE
9	Jeffrey P. Jones	\$42,000	\$3,500	Seattle	4/9/2000	FALSE
10	Joe Morrison	\$24,000	\$2,000	Seattle	3/13/1999	FALSE
11	John T. Foster	\$39,500	\$3,292	Seattle	10/10/1999	FALSE
12	Kurt Kamichoff	\$48,000	\$4,000	Los Angeles	11/5/2000	FALSE
13	Michael Hayden	\$76,230	\$6,519	Seattle	6/1/1999	TRUE
14	Phillip A. Todd	\$29,500	\$2,458	Portland	1/3/1998	FALSE
15	Richard E. Card	\$43,000	\$3,583	Seattle	7/2/1997	FALSE
16	Rick Fogerty	\$89,873	\$7,489	Portland	6/28/1998	TRUE
17	Robert H. Miller	\$149,000	\$12,417	Portland	5/1/1999	TRUE
18	Stephen C. Carter	\$44,123	\$3,677	Los Angeles	9/29/1998	FALSE
19	Steven H. Katz	\$32,900	\$2,742	Los Angeles	3/16/2000	FALSE
20	Thomas E. Abbott	\$60,000	\$5,000	Los Angeles	4/12/1997	FALSE
21	Tom Brown	\$65,000	\$5,417	Seattle	5/27/2000	TRUE
22						

Figure 9-2: When you choose the **Data** → **Filter** → **AutoFilter** command, Excel adds drop-down arrows to the field names in the header row.

When you click the arrow in one of these drop-down lists, the list expands to show the unique items in that column. Select an item, and Excel hides all rows except those that include the selected item. You can filter the list using a single field or multiple fields. The drop-down arrow changes color to remind you that you filtered the list by a value in that column.



AutoFiltering has a limit. Only the first 1,000 unique items in the column appear in the drop-down list. If your list exceeds this limit, you can use advanced filtering, which I describe later.

Besides showing every item in the column, the drop-down list offers five other choices:

- ◆ All: Displays all items in the column. Use this to remove filtering for a column.
- ◆ Top 10: Filters to display the “top 10” items in the list. Actually, this option is a misnomer; you can display the “top n” items (you choose the number).
- ◆ Custom: Enables you to filter the list by multiple items (see Figure 9-3).
- ◆ Blanks: Filters the list by showing rows that contain blanks in this column. This option is available only if the column contains one or more blank cells.
- ◆ NonBlanks: Filters the list by showing rows that contain nonblanks in this column. This option is available only if the column contains one or more blank cells.

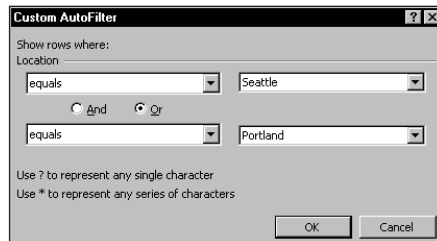


Figure 9-3: The Custom AutoFilter dialog box gives you more filtering options.



Excel automatically creates a hidden name (*_FilterDatabase*) for the range occupied by the filtered list. Note that the name begins with an underscore character. You can use this name in a VBA macro or in a formula. To select the filtered data range, press Ctrl+G to bring up the Go To dialog box. The hidden name does not appear in the list of names, so you need to enter it manually. Type **_FilterDatabase** in the Reference field and click OK.

Custom AutoFiltering is useful, but it definitely has limitations. For example, if you want to filter a list to show only three values in a field (such as New York or New Jersey or Connecticut), you can't do it through AutoFiltering. Such filtering tasks require the advanced filtering feature, which I discuss later in this chapter.

To display the entire unfiltered list again, click the arrow and choose All – the first item on the drop-down list. Or, you can select Data → Filter → Show All. To exit AutoFilter mode and remove the drop-down arrows from the field names, choose Data → Filter → AutoFilter again.

Counting and Summing Filtered Data

You can create a formula to display the number of filtered records. The formula that follows, for example, displays the number of filtered records by using the SUBTOTAL function, with 3 as the first argument:

```
=SUBTOTAL(3,A5:A400)
```

The first argument for the SUBTOTAL function determines the type of “totalling” that is performed. An argument of 3 specifies that the totalling will be equivalent to using Excel's COUNTA function.

Make sure that the range argument for the SUBTOTAL function begins with the first row of the list, and extends (at least) to the last row of the list.



You should put this formula in a row above or below the list. Otherwise, filtering the list may hide the row that contains the formula. Also, be aware that the count returned by the SUBTOTAL function does not include blank cells.

To display the sum of filtered records, use 9 as the first argument for the SUBTOTAL function. The following formula, for example, returns the sum of the filtered values in column C:

```
=SUBTOTAL(9,C5:C400)
```

Figure 9-4 shows the result of these formulas when applied to a filtered list.

Month	Region	Sales
Feb	North	5,684
Mar	North	3,531
Apr	North	5,955
May	North	3,286
Jun	North	4,705
Jul	North	3,436
Aug	North	3,292
Sep	North	3,779
Oct	North	3,193

Figure 9-4: The formulas in cells C1 and C2 use the SUBTOTAL function.



The SUBTOTAL function is the only function that recognizes data hidden by AutoFiltering. If you have other formulas that refer to data in a filtered list, these formulas don't adjust to use only the visible cells. For example, if a cell contains a formula that sums values in column C, the formula continues to show the sum for *all* the values in column C, not just those in the visible rows.



You can use the SUBTOTAL function to generate consecutive numbers for nonhidden rows in a filtered list. The numbering will adjust as you apply filtering to hide or display rows. If your list has the field names in row 1, enter this formula in cell A2, and then copy it down for each row in your list:

```
=SUBTOTAL(3, B$2:B2)
```

For more about the SUBTOTAL function, refer to “Creating Subtotals,” later in this chapter.

Copying and Deleting Filtered Data

Some of the standard spreadsheet operations work differently with a filtered list. For example, you might use the Format → Row → Hide command to hide rows. If you then copy a range that includes those hidden rows, all the data gets copied (even the hidden rows). But when you copy data in an AutoFiltered list, only the visible rows are copied.

Similarly, you can select and delete the visible rows in the table, and the rows hidden by AutoFiltering will not be affected.

About the SUBTOTAL Function

The SUBTOTAL function is very versatile. It's unique in that it is the only Excel function that ignores cells in hidden rows. There is one caveat, however: The rows must be hidden as a result of autofiltering or an outline. Simply hiding rows manually will have no effect on the results calculated by the SUBTOTAL function.

The first argument for the SUBTOTAL function determines the actual function used. For example, when the first argument is 1, the SUBTOTAL function works like the AVERAGE function. The following table shows the possible values for the first argument for the SUBTOTAL function:

Value	Function
1	AVERAGE
2	COUNT
3	COUNTA
4	MAX
5	MIN
6	PRODUCT
7	STDEV
8	STDEVP
9	SUM
10	VAR
11	VARP

Using Advanced Filtering

In many cases, AutoFiltering does the job just fine. But if you run up against its limitations, you need to use advanced filtering. Advanced filtering is much more flexible than AutoFiltering, but it takes a bit of up-front work to use it. Advanced filtering provides you with the following capabilities:

- ◆ You can specify more complex filtering criteria.

- ◆ You can specify computed filtering criteria.
- ◆ You can extract a copy of the rows that meet the criteria to another location.

Filling in the Gaps

When you import data, you can end up with a worksheet that looks something like the one in the accompanying figure. In this example, an entry in column A applies to several rows of data. If you sort such a list, you can end up with a mess and you won't be able to tell who sold what.

Monthly Sales Report				
Sales Rep	Month	Units Sold	Amount	New Clients
Bob	Jan	324	\$22,366	5
	Feb	331	\$22,839	6
	Mar	290	\$20,010	3
Karen	Jan	189	\$13,041	12
	Feb	234	\$16,146	11
	Mar	398	\$27,462	6
Elizabeth	Jan	541	\$37,329	16
	Feb	212	\$14,628	21
	Mar	681	\$46,989	7
Stan	Jan	771	\$53,199	14
	Feb	322	\$22,218	3
	Mar	821	\$56,649	11

When you have a small list, you can enter the missing cell values manually. But if you have a huge database, you need a better way of filling in those cell values. Here's how:

1. Select the range (A3:A14 in this example).
2. Press Ctrl+G to display the Go To dialog box.
3. In the Go To dialog box, click Special.
4. Select the Blanks option.
5. In the formula bar, type = followed by the address of the first cell with an entry in the column (=A3 in this example), and press Ctrl+Enter.
6. Reselect the range and choose Edit → Copy.
7. Select Edit → Paste Special, choose the Values option, and click OK.

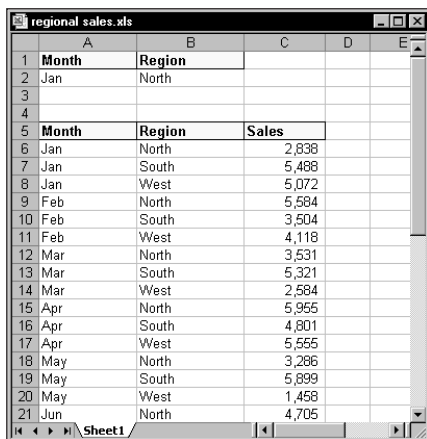
Setting Up a Criteria Range

Before you can use the advanced filtering feature, you must set up a *criteria range*, a designated range on a worksheet that conforms to certain requirements. The criteria range holds the information that Excel uses to filter the list. It must conform to the following specifications:

- ◆ It must consist of at least two rows, and the first row must contain some or all field names from the list.
- ◆ The other rows of the criteria range must consist of your filtering criteria.

You can put the criteria range anywhere in the worksheet, or even in a different worksheet. You should avoid putting it in rows where you placed the list. Because Excel may hide some of these rows when filtering the list, you may find that your criteria range is no longer visible after filtering. Therefore, you should generally place the criteria range above or below the list.

Figure 9-5 shows a criteria range, located in A1:B2, above the list that it uses. Notice that the criteria range does not include all of the field names from the list. You can include only the field names for fields that you use in the selection criteria.



	A	B	C	D	E
1	Month	Region			
2	Jan	North			
3					
4					
5	Month	Region	Sales		
6	Jan	North	2,838		
7	Jan	South	5,488		
8	Jan	West	5,072		
9	Feb	North	5,584		
10	Feb	South	3,504		
11	Feb	West	4,118		
12	Mar	North	3,531		
13	Mar	South	5,321		
14	Mar	West	2,584		
15	Apr	North	5,955		
16	Apr	South	4,801		
17	Apr	West	5,555		
18	May	North	3,286		
19	May	South	5,899		
20	May	West	1,458		
21	Jun	North	4,705		

Figure 9-5: A criteria range for a list

In this example, the criteria range has only one row of criteria. The fields in each row of the criteria range (except for the header row) are joined with an AND operator. Therefore, after applying the advanced filter, the list shows only the rows in which the Month column equals *Jan* AND the Region column equals *North*. You may find specifying criteria in the criteria range a bit tricky. I discuss this topic in detail later in this chapter. See “Specifying Advanced Filter Criteria.”

Extracting Unique Records From a List

A common question among Excel users is, "How can I get rid of duplicate records in a list?"

Perhaps the easiest solution uses advanced filtering. Activate any cell within your list and choose Data → Filter → Advanced filter. In the Advanced Filter dialog box, select Copy to another location and specify a new location in the Copy to box (the new location must be on the same worksheet). Then, place a check mark next to Unique records only. Click OK and you'll have a copy of your list, without the duplicate records. By the way, this is the only Advanced Filter operation that does not require a criteria range.

Filtering a List

To perform the filtering, select any cell within your list. Then choose Data → Filter → Advanced filter. Excel displays the Advanced Filter dialog box, shown in Figure 9-6. Excel guesses your List range (you can change it if necessary), but you need to specify the criteria range. To filter the list in place (i.e., hide rows that don't qualify), select the option labeled Filter the list, in-place. If you select the Copy to another location option, you need to specify a range in the Copy to box. Click OK, and Excel filters the list by the criteria that you specify.

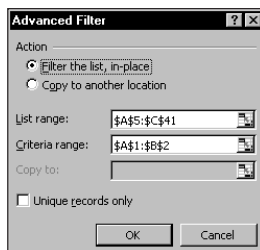


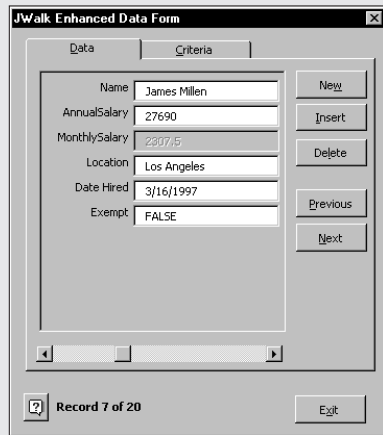
Figure 9-6: The Advanced Filter dialog box

When you copy filtered records to another location (in other words, when you select the Copy to another location option), you can specify which columns to include in the copy. Before displaying the Advanced Filter dialog box, copy the desired field labels to the first row of the area where you plan to paste the filtered rows. In the Advanced Filter dialog box, specify a reference to the copied column labels in the Copy to box. The copied rows then include only the columns for which you copied the labels.

Working with Data in a List

Excel's Data → Form command displays a dialog box to help you work with a list. This dialog enables you to enter new data, delete rows, and search for rows that match certain criteria.

Excel's Data Form is handy, but by no means ideal. If you like the idea of using a dialog box to work with data in a list, check out my Enhanced Data Form add-in. It offers many advantages over Excel's Data Form.



After you install the add-in, activate any cell in a list and then choose Data → JWALK Enhanced Data Form. Data that makes up the current record appears in the dialog box. Use the horizontal scrollbar (or the Previous/Next buttons) to scroll through the database. Changes you make to the data are written to the database, and undo is available. The form handles an unlimited number of fields, and a wildcard-capable search window permits quick retrieval of the desired record based on any field.



You can access the JWALK Enhanced Data Form add-in on the companion CD-ROM.

Specifying Advanced Filter Criteria

Microsoft's enhancements to list-related features in Excel have focused exclusively on AutoFiltering. The use of a separate criteria range for advanced filtering originated with the original version of Lotus 1-2-3. Excel adapted this method, and it

has never been changed, despite the fact that specifying advanced filtering criteria remains one of the most confusing aspects of Excel. This section presents plenty of examples to help you understand how to create a criteria range that extracts the information you need.

The examples in this section use the list shown in Figure 9-7. This list, which has 125 records and eight fields, was designed to use a good assortment of data types: values, text strings, logicals, and dates. The list occupies the range A8:H133 (rows above the list are used for the criteria range).

	A	B	C	D	E	F	G	H
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000		4	3	1,905	Condo	FALSE
19	\$259,900	12/3/2000	N. County	3	2.5	2,122	Condo	FALSE
20	\$405,000	12/6/2000	N. County	2	3	2,444	Single Family	TRUE
21	\$236,900	12/7/2000	S. County	2	2	1,483	Condo	FALSE
22	\$240,000	12/7/2000	S. County	3	2.5	1,595	Condo	FALSE
23	\$304,900	12/9/2000	S. County	4	3	2,350	Single Family	FALSE
24	\$349,900	12/13/2000	N. County	4	3	2,290	Single Family	TRUE
25	\$249,000	12/21/2000	Central	4	3	1,940	Single Family	TRUE
26	\$229,500	12/24/2000	Central	4	3	2,041	Single Family	FALSE

Figure 9-7: This list contains information about real estate listings.



The workbook shown in Figure 9-7 is available on the companion CD-ROM.

Specifying a Single Criterion

The examples in this section use a single-selection criterion. In other words, the contents of a single field determine the record selection.



You also can use AutoFiltering to perform this type of filtering.

To select only the records that contain a specific value in a specific field, enter the field name in the first row of the criteria range, and the value to match in the second row. Figure 9-8, for example, shows the criteria range (A1:A2) that selects records containing the value 4 in the Bedrooms field.

	A	B	C	D	E	F	G	H
1	Bedrooms							
2	4							
3								
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/03/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/05/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/07/2000	S. County	2	2	1,562	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/01/2000	S. County	4	3	2,800	Condo	TRUE

Figure 9-8: The criteria range (A1:A2) selects records that describe properties with four bedrooms.

Note that the criteria range does not need to include all of the fields from the list. If you work with different sets of criteria, you may find it more convenient to list all of the field names in the first row of your criteria range.

USING COMPARISON OPERATORS

You can use comparison operators to refine your record selection. For example, you can select records based on any of the following:

- ◆ Homes that have at least four bedrooms
- ◆ Homes with a square footage less than 2,000
- ◆ Homes with a list price of no more than \$200,000

To select the records that describe homes that have at least four bedrooms, make the following entries in the criterion range:

A1: Bedrooms
A2: >=4

Table 9-1 lists the comparison operators that you can use with text or value criteria. If you don't use a comparison operator, Excel assumes the equal sign operator (=).

TABLE 9-1 COMPARISON OPERATORS

Operator	Comparison Type
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
< >	Not equal to

Table 9-2 shows examples of some criteria that use comparison operators.

TABLE 9-2 EXAMPLES OF COMPARISON OPERATORS

Criteria	Selects
>100	Records that contain a value greater than 100
<>0	Records that contain a value not equal to 0
=500	Records that contain a value of 500 (omitting the equal sign gives the same result)
<5000	Records that contain a value less than 5000
>=5000	Records that contain a value less than or equal to 5000

USING WILDCARD CHARACTERS

Criteria that use text also can make use of two “wildcard” characters: an asterisk (*) matches any number of characters; a question mark (?) matches any single character. Table 9-3 shows examples of criteria that use text. Some of these are a bit counter-intuitive. For example, to select records that match a single character, you must enter the criterion as a formula (refer to the last entry in the table).

TABLE 9-3 EXAMPLES OF TEXT CRITERIA

Criteria	Selects
= "January"	Records that contain the text <i>January</i> (and nothing else). You must enter this exactly as shown: as a formula, with an initial equal sign.
January	Records that begin with the text <i>January</i> .
C	Records that contain text that begins with the letter C.
<>C*	Records that contain any text, except text that begins with the letter C.
>K	Records that contain text that begins with the letters L through Z.
County	Records that contain text that includes the word <i>COUNTY</i> .
Sm*	Records that contain text that begins with the letters <i>SM</i> .
s*s	Records that contain text that begins with S and have a subsequent occurrence of the letter S.
s?s	Records that contain text that begins with S and has another S as its third character. Note that this does <i>not</i> select only three-character words.
= "s*s"	Records that contain text that begins and ends with S. You must enter this exactly as shown: as a formula, with an initial equal sign.
<>*c	Records that contain text that does not end with the letter C.
<>?????	All records that don't contain exactly five letters.
<>*c*	Records that do not contain the letter C.
~?	Records that contain a single question mark character.
=	Records that contain a blank.
<>	Records that contain any nonblank entry.
= "c"	Records that contain the single character C. You must enter this exactly as shown: as a formula, with an initial equal sign.



The text comparisons are not case sensitive. For example, *se** matches *Seligman*, *seller*, and *SEC*.

Specifying Multiple Criteria

Often, you may want to select records based on criteria that use more than one field or multiple values within a single field. These selection criteria involve logical OR or AND comparisons. Following are a few examples of the types of multiple criteria that you can apply to the real estate database:

- ◆ A list price less than \$250,000, and square footage of at least 2,000
- ◆ Single-family home with a pool
- ◆ At least four bedrooms, at least three bathrooms, and square footage less than 3,000
- ◆ A home listed for no more than one month, with a list price greater than \$300,000
- ◆ A condominium with square footage between 1,000 and 1,500
- ◆ A single-family home listed in the month of March

To join criteria with an AND operator, use multiple columns in the criteria range. Figure 9-9 shows a criteria range that selects records with a list price of less than \$250,000 and a square footage of at least 2,000.

	A	B	C	D	E	F	G	H
1	ListPrice	SqFt						
2	<25000	>=2000						
3								
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000	N. County	4	3	1,905	Condo	FALSE

Figure 9-9: This criteria range uses multiple columns that select records using a logical AND operation.

Figure 9-10 shows another example. This criteria range selects records that were listed in the month of March. Notice that the field name (Date Listed) appears twice in the criteria range. The criteria selects the records in which the Date Listed date is greater than or equal to March 1 AND the Date Listed date is less or equal to March 31.



The criteria shown in Figure 9-9 may not work properly for systems that don't use the U.S. date formats. To ensure compatibility with different date systems, use the DATE function to define such criteria, as in the following formulas

```
= ">=" & DATE(2001,3,1)
```

```
= "<=" & DATE(2001,3,31)
```

	A	B	C	D	E	F	G	H
1	Date Listed	Date Listed						
2	>=3/1/2001	<=3/31/2001						
3								
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE

Figure 9-10: This criteria range selects records that describe properties that were listed in the month of March.

To join criteria with a logical OR operator, use more than one row in the criteria range. A criteria range can have any number of rows, each of which joins with the others via an OR operator. Figure 9-11 shows a criteria range (A1:C3) with two rows of criteria.

	A	B	C	D	E	F	G	H
1	Type	SqFt	ListPrice					
2	Condo	>=1800						
3	Single Family		<210000					
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000		4	3	1,905	Condo	FALSE

Figure 9-11: This criteria range has two sets of criteria, each of which is in a separate row.

In this example, the filtered list shows the rows that meet either of the following conditions:

- ◆ A condo with a square footage of at least 1,800, OR
- ◆ A single-family home with a list price under \$210,000



You cannot perform this type of filtering using AutoFiltering.

Specifying Computed Criteria

Using computed criteria can make filtering even more powerful. Computed criteria filter the list based on one or more calculations. Figure 9-12 shows a criteria range that selects records in which the list price is less than the average list price of all records. The formula in cell B2 is as follows:

```
=ListPrice>AVERAGE(A:A)
```



This formula will generate a #NAME? error if you are not using the Accept labels in formulas option. This setting is specified in the Calculation tab of the Options dialog box. If you are not using this option, the #NAME? error will not cause any problems.

	A	B	C	D	E	F	G	H
1		Above Avg						
2		TRUE						
3								
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000		4	3	1,905	Condo	FALSE

Figure 9-12: This criteria range uses computed criteria.

Keep these following points in mind when using computed criteria:

- ◆ Computed criteria formula are always logical formulas: They must return either TRUE or FALSE.
- ◆ You can use the field label in your formula. In the preceding example, ListPrice is not a named range. It is a field label in the database. Alternatively, you can use a reference to the cell in the first data row in the field of interest (not a reference to the cell that contains the field name). In this example, the cell in the first data row for the ListPrice field is cell A9. The following formula returns the same result as the previous example:

```
=A9>AVERAGE(A:A)
```
- ◆ Ignore the values returned by formulas in the criteria range. These refer to the first row of the list. Sometimes, using a field label in the formula results in an error value such as #NAME? or #VALUE!. You can just ignore this error. It does not affect how the list is filtered.
- ◆ When you use computed criteria, do not use an existing field label in your criteria range. In Figure 9-12, notice that cell B1 contains *Above Avg*, which is not a field name from the list. A computed criteria essentially computes a new field for the list. Therefore, you must supply a new field name in the first row of the criteria range. Or, if you prefer, you can simply leave the field name cell blank.
- ◆ You can use a reference to an entire column in a computed criteria formula. In the preceding example, the AVERAGE function used A:A as its argument. If you do so, the criteria formula must be in a different column than the column referenced. Failure to do so results in a circular reference. If you prefer, you can simply use the actual address of the column within your list.
- ◆ You can use any number of computed criteria and mix and match them with noncomputed criteria.
- ◆ If your computed formula refers to a value outside the list, use an absolute reference rather than a relative reference. For example, use \$C\$1 rather than C1.

COMPUTED CRITERIA EXAMPLES

Figure 9-13 shows another example of computed criteria. This criteria selects records in which the sum of the bedrooms and bathrooms is greater than 8. The label in cell A1 is descriptive and does not affect the filtering.

Notice that the computed criteria formula returns an error value because the formula refers to field names. The filtering works correctly, despite the error.

=Bedrooms+Baths>8

Alternatively, you can write this formula, which refers to the first data row in the list:

=D9+E9>8

Using this formula does not return an error, but the formula isn't as easy to understand.

	A	B	C	D	E	F	G	H
1	8+ Rooms							
2	#VALUE!							
3								
4								
5								
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000		4	3	1,905	Condo	FALSE

Figure 9-13: This criteria range uses computed criteria.

Following is another example of a computed criteria formula. This formula selects the records listed within the past 60 days.

=B9>TODAY()-60

=Date Listed>TODAY()-60

USING ARRAYS WITH COMPUTED CRITERIA

Excel also supports arrays in computed criteria formulas. To see how this may be useful, consider a situation in which you want to identify properties that don't have a "half bath." Filter out records that have 3.5, 4.5, or some other noninteger value in the Baths field. Figure 9-14 displays one example. The criteria range, A1:A5, uses four OR criteria to make the selection.

	A	B	C	D	E	F	G	H
1	Baths							
2		2						
3		3						
4		4						
5		5						
6								
7								
8	ListPrice	Date Listed	Area	Bedrooms	Baths	SqFt	Type	Pool
9	\$350,000	11/3/2000	N. County	3	2.5	1,991	Condo	FALSE
10	\$215,000	11/5/2000	Central	3	1.75	2,157	Single Family	TRUE
11	\$315,000	11/7/2000	S. County	2	2	1,552	Condo	FALSE
12	\$379,000	11/11/2000	N. County	4	3	3,000	Single Family	FALSE
13	\$248,500	11/21/2000	?	4	2.5	2,101	Single Family	TRUE
14	\$297,500	11/23/2000	S. County	4	3.5	2,170	Single Family	FALSE
15	\$259,900	11/27/2000	N. County	4	3	1,734	Condo	FALSE
16	\$325,000	12/1/2000	S. County	4	3	2,800	Condo	TRUE
17	\$208,750	12/3/2000	S. County	4	3	2,207	Single Family	TRUE
18	\$227,500	12/3/2000		4	3	1,905	Condo	FALSE

Figure 9-14: Using four OR criteria to select records with noninteger bathrooms

Another option uses this single-computed criteria formula:

```
=OR(Baths={2,3,4,5,6,7})
```

This formula returns TRUE if the value in the Bath field equals any of the values in the array.

Using Database Functions with Lists

To create formulas that return results based on filtering criteria, use Excel's database worksheet functions. These functions all begin with the letter D, and are listed in the Database category of the Insert Function dialog box.

Table 9-4 lists Excel's database functions. Each of these functions operates on a single field in the database.

TABLE 9-4 EXCEL'S DATABASE WORKSHEET FUNCTIONS

Function	Description
DAVERAGE	Returns the average of database entries that match the criteria
DCOUNT	Counts the cells containing numbers from the specified database and criteria
DCOUNTA	Counts nonblank cells from the specified database and criteria
DGET	Extracts from a database a single field from a single record that matches the specified criteria

Function	Description
DMAX	Returns the maximum value from selected database entries
DMIN	Returns the minimum value from selected database entries
DPRODUCT	Multiplies the values in a particular field of records that match the criteria in a database
DSTDEV	Estimates the standard deviation of the selected database entries (assumes that the data is a sample from a population) of selected database entries
DSTDEVP	Calculates the standard deviation of the selected database entries, based on the entire population of selected database entries
DSUM	Adds the numbers in the field column of records in the database that match the criteria
DVAR	Estimates the variance from selected database entries (assumes the data is a sample from a population)
DVARP	Calculates the variance, based on the entire population of selected database entries

The database functions all require a separate criteria range, which is specified as the last argument for the function. The database functions use exactly the same type of criteria range as discussed earlier in “Specifying Advanced Filter Criteria.”

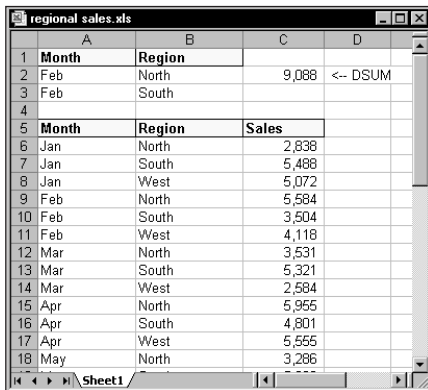
Refer to Figure 9-15. The formula in cell C2, which follows, uses the DSUM function to calculate the sum of values in a list that meet certain criteria. Specifically, the formula returns the sum of the Sales column for records in which the Month is “Feb” and the Region is “North” or the Region is “South.”

```
=DSUM(Database,3,Criteria)
```

In this case, the list is named *Database*, 3 is the field number of the column you are summing, and *Criteria* is the name of the criteria range (A1:B3).

Following is an alternate version of this formula that uses the field name instead of the field number. This version is easy to read and will continue to function if a new field is inserted before column 3.

```
=DSUM(Database,"Sales",Criteria)
```

	A	B	C	D
1	Month	Region		
2	Feb	North	9,088	<-- DSUM
3	Feb	South		
4				
5	Month	Region	Sales	
6	Jan	North	2,838	
7	Jan	South	5,488	
8	Jan	West	5,072	
9	Feb	North	5,584	
10	Feb	South	3,504	
11	Feb	West	4,118	
12	Mar	North	3,531	
13	Mar	South	5,321	
14	Mar	West	2,584	
15	Apr	North	5,955	
16	Apr	South	4,801	
17	Apr	West	5,555	
18	May	North	3,286	

Figure 9-15: Using the DSUM function to sum a list using a criteria range.



You may find it cumbersome to set up a criteria range every time you need to use a database function. Fortunately, Excel provides some alternative ways to perform conditional sums and counts. Refer to Chapter 7 for examples that use SUMIF, COUNTIF, and various other techniques.

If you're an array formula aficionado, you might be tempted to use a literal array in place of the criteria range. In theory, the following array formula *should* work (and would eliminate the need for a separate criteria range). Unfortunately, the database functions do not support arrays, and this formula simply returns a #VALUE! error.

```
{=DSUM(Database,3,{"Month","Region";"Feb","North"})}
```



In the original release of Excel 97, the database functions do not work correctly if the first argument refers to a range that contains more than 32,768 rows. Excel 97 SR-1 corrected this problem.

Working with a Lotus 1-2-3 File?

If you open a 1-2-3 file in Excel, be aware that Excel evaluates the database criteria ranges differently. This may affect the results obtained when using advanced filtering and database functions.

For example, in 1-2-3, a criteria such as "John" finds only rows with cells that contain the text "John." When you open a 1-2-3 file in Excel, the "transition formula evaluation" is in effect. If you don't change this setting, the criteria ranges will be evaluated as they are in 1-2-3.

But if you select Tools → Options, and clear the Transition formula evaluation check box (in the Transition tab of the Options dialog box), Excel evaluates the criteria range using its rules (which are different). For example, the "John" criteria finds any rows that contain cells with text beginning with "John"; this includes cells that contain "John," "John Smith," and "Johnson."



Appendix A contains more information about working with 1-2-3 files.

Summarizing a List with a Data Table

This section describes a technique that you can use to summarize the information in a database. It uses the Data → Table command to create a dynamic summary table. A pivot table is often your best choice for this type of thing, but this technique offers one advantage: The data table is updated automatically (you do not need to refresh it, as in a pivot table).

Figure 9-16 shows part of a simple sales list that occupies five columns. The list contains a monthly sales total (column E) for each sales representative, along with the number of sales contacts made (column D) and the sales rep's region (either North or South, in column C). For example, in January, Bob (a sales rep for the North region), made 58 contacts for total sales of \$283,800.

	A	B	C	D	E
1	Month	Sales Rep	Region	Contacts	Sales
2	Jan	Bob	North	58	283,800
3	Jan	Frank	North	35	507,200
4	Jan	Paul	South	25	107,600
5	Jan	Randy	South	47	391,600
6	Jan	Mary	South	39	226,700
7	Feb	Bob	North	44	558,400
8	Feb	Jill	North	46	350,400
9	Feb	Frank	North	74	411,800
10	Feb	Paul	South	29	154,200
11	Feb	Randy	South	45	258,000
12	Feb	Mary	South	52	233,800
13	Mar	Bob	North	30	353,100
14	Mar	Jill	North	44	532,100
15	Mar	Frank	North	57	258,400
16	Mar	Paul	South	13	286,000
17	Mar	Randy	South	14	162,200
18	Mar	Mary	South	36	134,300
19	Apr	Bob	North	54	595,500
20	Apr	Jill	North	44	480,100
21	Apr	Frank	North	79	555,500
22	Apr	Paul	South	36	328,200
23	Apr	Randy	South	31	154,200
24	Apr	Mary	South	22	200,600
25	May	Bob	North	63	328,600
26	May	Jill	North	70	589,900

Figure 9-16: A data table is a good way to summarize this list.

The list contains 76 records, and the entire list (A1:E77) is named *Database*. Range G1:H2 stores a criteria range for the list. This range is named *Criteria*. The goal is to create a summary table that shows key information by month. Figure 9-17 shows the summary table in G8:K23 – created using the Data → Table command.

	A	B	C	D	E	F	G	H	I	J	K	
1	Month	Sales Rep	Region	Contacts	Sales		Month	Region				
2	Jan	Bob	North	58	283,800		Jan	North				
3	Jan	Frank	North	35	507,200							
4	Jan	Paul	South	25	107,600							
5	Jan	Randy	South	47	391,600							
6	Jan	Mary	South	39	226,700							
7	Feb	Bob	North	44	558,400							
8	Feb	Jill	North	46	350,400				Sales Reps	Contacts	Sales	Sales/Contact
9	Feb	Frank	North	74	411,800				2	93	791,000	8,505
10	Feb	Paul	South	29	154,200		Jan		2	93	791,000	8,505
11	Feb	Randy	South	45	258,000		Feb		3	164	1,320,600	8,052
12	Feb	Mary	South	52	233,800		Mar		3	131	1,143,600	8,730
13	Mar	Bob	North	30	353,100		Apr		3	177	1,631,100	9,215
14	Mar	Jill	North	44	532,100		May		3	173	1,064,300	6,152
15	Mar	Frank	North	57	258,400		Jun		3	132	1,001,200	7,585
16	Mar	Paul	South	13	286,000		Jul		3	166	872,300	5,255
17	Mar	Randy	South	14	162,200		Aug		3	127	1,082,100	8,520
18	Mar	Mary	South	36	134,300		Sep		3	148	1,239,300	8,374
19	Apr	Bob	North	54	595,500		Oct		3	147	962,100	6,545
20	Apr	Jill	North	44	480,100		Nov		4	162	1,004,522	6,201
21	Apr	Frank	North	79	555,500		Dec		4	205	1,219,163	5,947
22	Apr	Paul	South	36	328,200							
23	Apr	Randy	South	31	154,200		TOTALS			1,825	13,331,305	7,423
24	Apr	Mary	South	22	200,600							

Figure 9-17: Use the Data → Table command to create this summary table.



The workbook shown in Figure 9-17 is available on the companion CD-ROM. For comparison, the workbook also contains a pivot table summary, plus a table that uses array formulas (as described in Chapter 7).

To create this data table:

1. Enter the month names in G10:G21.
2. Enter the descriptive labels shown in H8:K8.
3. Enter the formulas from Table 9-5 into cells in row 9.
4. Select the range G9:K21.
5. Choose Data → Table. Excel displays the Table dialog box shown in Figure 9-18.
6. In the Table dialog box, enter G2 into the field labeled Column input cell (leave the Row input cell field empty).
7. Click OK.

TABLE 9-5 FORMULAS TO ENTER

Cell	Formula
H9	=DCOUNTA(Database,"Sales Rep",Criteria)
I9	=DSUM(Database,"Contacts",Criteria)
J9	=DSUM(Database,"Sales",Criteria)
K9	=J9/I9

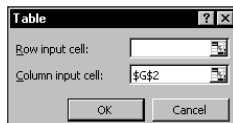


Figure 9-18: The Table dialog box, used for creating a data table.

Excel inserts a single array formula into H10:K21. The formula is as follows:

```
=TABLE(,G2)
```

This formula uses the information in the cells to the left (G10:G21) and above (H9:K9) to perform calculations. It evaluates the formulas in row 9, substituting the corresponding month in column G. In other words, the single criteria range is being treated as if it were a series of criteria ranges.

You can enter a region name (either North or South) in cell H2 and the data table will show the information for that region. If H2 is blank, the data table shows information for all regions.

Creating Subtotals

Excel's Data → Subtotals command is a handy tool that inserts formulas into a list automatically. These formulas use the SUBTOTAL function, which actually does more than simply sum data. To use this feature, your list must be sorted, because the formulas are inserted whenever the value in a specified field changes.

Figure 9-19 shows an example of a list that is appropriate for subtotals. This list is sorted by the Month field, and then by the Region field.

	A	B	C	D	E	F
	Month	State	Region	Contacts	Sales	
2	Jan	California	West	58	283,800	
3	Jan	Washington	West	35	507,200	
4	Jan	Oregon	West	39	226,700	
5	Jan	New York	East	25	107,600	
6	Jan	New Jersey	East	47	391,600	
7	Feb	California	West	44	556,400	
8	Feb	Washington	West	74	411,800	
9	Feb	Oregon	West	46	360,400	
10	Feb	New York	East	52	233,800	
11	Feb	New Jersey	East	29	154,200	
12	Mar	California	West	30	353,100	
13	Mar	Washington	West	57	258,400	
14	Mar	Oregon	West	44	532,100	
15	Mar	New York	East	36	134,300	
16	Mar	New Jersey	East	14	162,200	
17						

Figure 9-19: This list is a good candidate for subtotals, which are inserted at each change of the month and at each change of the region.

To insert subtotal formulas into a list automatically, move the cell pointer anywhere in the list and choose Data → Subtotals. You will see the Subtotal dialog box, shown in Figure 9-20.



Figure 9-20: The Subtotal dialog box automatically inserts subtotal formulas into a sorted list.

The Subtotal dialog box offers the following choices:

- ◆ **At each change in:** This drop-down list displays all fields in your list. You must have sorted the list by the field that you choose.
- ◆ **Use function:** Choose from 11 functions (Sum is the default).
- ◆ **Add subtotal to:** This list box shows all the fields in your list. Place a check mark next to the field or fields that you want to subtotal.
- ◆ **Replace current subtotals:** If checked, Excel removes any existing subtotal formulas and replaces them with the new subtotals.
- ◆ **Page break between groups:** If checked, Excel inserts a manual page break after each subtotal.
- ◆ **Summary below data:** If checked, Excel places the subtotals below the data (the default). Otherwise, the subtotal formulas appear above the data.
- ◆ **Remove All:** This button removes all subtotal formulas in the list.

When you click OK, Excel analyzes the list and inserts formulas as specified—and even creates an outline for you. Figure 9-21 shows a worksheet after adding two sets of subtotals: one that summarizes by month, and another that summarizes by region. You can, of course, use the SUBTOTAL function in formulas that you create manually. Using the Data → Subtotals command is usually easier.

	A	B	C	D	E
1	Month	State	Region	Contacts	Sales
2	Jan	California	West	58	283,800
3	Jan	Washington	West	35	507,200
4	Jan	Oregon	West	39	226,700
5			West Total	132	1,017,700
6	Jan	New York	East	25	107,600
7	Jan	New Jersey	East	47	391,600
8			East Total	72	499,200
9	Jan Total			204	1,516,900
10	Feb	California	West	44	558,400
11	Feb	Washington	West	74	411,800
12	Feb	Oregon	West	46	350,400
13			West Total	164	1,320,600
14	Feb	New York	East	52	233,800
15	Feb	New Jersey	East	29	154,200
16			East Total	81	388,000
17	Feb Total			245	1,708,600
18	Mar	California	West	30	353,100
19	Mar	Washington	West	57	258,400
20	Mar	Oregon	West	44	532,100
21			West Total	131	1,143,600
22	Mar	New York	East	36	134,300
23	Mar	New Jersey	East	14	162,200
24			East Total	50	296,500
25	Mar Total			181	1,440,100
26					
27	Grand Total			630	4,665,600
28			Grand Total	630	4,665,600
29					

Figure 9–21: Excel adds the subtotal formulas automatically—and even creates an outline.



If you add subtotals to a filtered list, the subtotals may no longer be accurate when you remove the filter.

The formulas all use the SUBTOTAL worksheet function. For example, the formula in cell E9 (total sales for January) is as follows:

```
=SUBTOTAL(9, E2:E7)
```

Although this formula refers to two other cells that contain a SUBTOTAL formula (E5 and E8), those cells are not included in the sum to avoid double-counting.

You can use the outline controls to adjust the level of detail shown. Figure 9–22, for example, shows only the summary rows from the subtotaled list. These rows contain the SUBTOTAL formulas.

	A	B	C	D	E
1	Month	State	Region	Contacts	Sales
5			West Total	132	1,017,700
8			East Total	72	499,200
9	Jan Total			204	1,516,900
13			West Total	164	1,320,600
16			East Total	81	388,000
17	Feb Total			245	1,708,600
21			West Total	131	1,143,600
24			East Total	50	296,500
25	Mar Total			181	1,440,100
26					
27	Grand Total			630	4,665,600
28			Grand Total	630	4,665,600
29					

Figure 9-22: Using the outline controls to hide the detail and display only the summary rows

Summary

This chapter presented various formula techniques relevant to working with a list. A list (also known as a worksheet database) is an organized collection of information. The first row contains field names, and subsequent rows contain data (records). AutoFiltering presents a useful method of filtering a list using simple criteria; for more complex criteria, you need to use advanced filtering, which requires a criteria range. This chapter also discussed Excel's database functions (which also require a criteria range) and the SUBTOTAL function.

Chapter 10 covers a wide variety of miscellaneous calculations.

Chapter 10

Miscellaneous Calculations

IN THIS CHAPTER

- ◆ Conversion factors for a wide variety of measurement units
- ◆ Formulas for calculating the various parts of a right triangle
- ◆ Calculations for area, surface, circumference, and volume
- ◆ Matrix functions to solve simultaneous equations
- ◆ Formulas that demonstrate various ways to round numbers

THIS CHAPTER CONTAINS REFERENCE information that may be useful to you at some point. Consider it a cheat sheet to help you remember the stuff you may have learned, but have long since forgotten.

Unit Conversions

You know the distance from New York to London in miles, but your European office needs the numbers in kilometers. What's the conversion factor? The information in this section contains many useful conversion factors that you can use in your formulas.



Excel's CONVERT function (available only when you install the Analysis ToolPak add-in) can calculate many unit conversions (refer to the online help for complete details). In some cases, however, you may find it more efficient to create your own conversion formulas so you don't need to rely on the Analysis ToolPak. To create your own conversion formula, you need to know the specific conversion factor for the measurement units.

Using the Unit Conversion Tables

To convert from one measurement unit to another, locate the appropriate conversion table in this section and determine the conversion factor. For example, to convert meters to inches, use the Distance Conversion Factors table. Refer to the

third row of the table (labeled *Meter*) and then locate the column labeled *Inch*. The meter-to-inch conversion factor is 39.37007874.

You can then use the conversion factor in a formula. For example, if cell A1 contains the value in meters, enter the following formula to convert it to inches:

```
=A1*39.37007874
```

Converting Metric Units

To convert to or from other metric units, you need to use an additional metric conversion factor from Table 10-1. To use this table, multiply the basic metric unit by the metric conversion factor. For example, consider the meter unit of distance measurement. A kilometer is 1 meter times 1E+03, or 1,000 meters. A millimeter, conversely, is 1 meter times 1E-03, or 1/1,000 meters.



The companion CD-ROM includes a workbook that contains all the conversion tables in this chapter.



In some cases, the values shown in the tables in this chapter are rounded. The conversion tables in the workbook contain values with full precision. For increased accuracy, make sure that you use the values in the workbook.

TABLE 10-1 METRIC CONVERSION FACTORS

Metric Prefix	Metric Conversion Factor
Exa	1E+18
Peta	1E+15
Tera	1E+12
Giga	1E+09
Mega	1E+06
Kilo	1E+03
Hecto	1E+02

Metric Prefix	Metric Conversion Factor
Deci	1E-01
Centi	1E-02
Milli	1E-03
Micro	1E-06
Nano	1E-09
Pico	1E-12
Femto	1E-15
Atto	1E-18

If you want to convert from a metric unit to a nonmetric unit, *multiply* the conversion factor by the metric conversion factor. If you convert from a nonmetric unit to a metric unit, *divide* the conversion factor by the metric conversion factor.

For example, suppose cell A1 contains a value in millimeters and you need to convert it to inches. Multiply the value in A1 by the meter-to-inch conversion factor (39.37007874) and multiply the result by the metric conversion factor (1E-03). The resulting formula is as follows:

```
=A1*39.37007874*1E-03
```

You can, of course, simplify the formula by replacing the second multiplication operation with its result:

```
=A1*0.03937007874
```

Now, assume cell A1 contains a value in inches and you need to convert it to millimeters. In this case, the inch-to-meter distance unit conversion factor is 0.0254 and the metric conversion factor is 1E-03. The formula to convert from inches to millimeters is as follows:

```
=A1*0.0254/1E-03
```

Or, in simpler terms:

```
=A1*25.4
```

Distance Conversions

Table 10-2 shows conversion factors for six common units of measurement. For details on using this table, see the subsection, “Using the Unit Conversion Tables,” earlier in the chapter.

Weight Conversions

Table 10-3 shows conversion factors for three common units of weight. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in the chapter.

Liquid Measurement Conversions

Table 10-4 shows conversion factors for eight common liquid measurement units. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in the chapter.

Surface Conversions

Table 10-5 shows conversion factors for seven common units of surface (or area). For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in the chapter.

Volume Conversions

Table 10-6 shows conversion factors for four common volume measurement units. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in this chapter.

Force Conversions

Table 10-7 shows conversion factors for three common units of force. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in this chapter.

Energy Conversions

Table 10-8 shows conversion factors for nine common units of energy. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in this chapter.

Time Conversions

Table 10-9 shows conversion factors for five common units of time. For details on using this table, see the subsection “Using the Unit Conversion Tables” earlier in this chapter.

TABLE 10-2 DISTANCE CONVERSION FACTORS

	Foot	Inch	Meter	Nautical Mile	Statute Mile	Yard
Foot	1	12	0.3048	0.000164579	0.000189394	0.3333333333
Inch	0.0833333333	1	0.0254	1.37149E-05	1.57828E-05	0.0277777778
Meter	3.280839895	39.37007874	1	0.000539957	0.000621371	1.093613298
Nautical mile	6076.115486	72913.38583	1852	1	1.150779448	2025.371828
Statute mile	5280	63360	1609.344	0.868976242	1	1759.999999
Yard	3	36	0.9144	0.000493737	0.000568182	1

TABLE 10-3 WEIGHT CONVERSION FACTORS

	Gram	Ounce	Pound
Gram	1	0.035274	0.002205
Ounce	28.34952	1	0.0625
Pound	453.5923	16	1

TABLE 10-4 LIQUID MEASUREMENT CONVERSION FACTORS

	Cup	Fluid Ounce	Gallon	Liter	Pint	Quart	Table-spoon	Teaspoon
Cup	1	8	0.0625	0.23664	0.5	0.25	16	48
Fluid ounce	0.125	1	0.007813	0.02958	0.0625	0.03125	2	6
Gallon	16	128	1	3.786235	8	4	256	768
Liter	4.225833	33.80667	0.264115	1	2.112917	1.056458	67.61333	202.84
Pint	2	16	0.125	0.473279	1	0.5	32	96
Quart	4	32	0.25	0.946559	2	1	64	192
Tablespoon	0.0625	0.5	0.003906	0.01479	0.03125	0.015625	1	3
Teaspoon	0.020833	0.166667	0.001302	0.00493	0.010417	0.005208	0.333333	1

TABLE 10-5 SURFACE MEASUREMENT CONVERSION FACTORS

	Acre	Hectare	Square Foot	Square Inch	Square Meter	Square Mile	Square Yard
Acre	1	0.404685642	43560	6272640	4046.856422	0.0015625	4839.999997
Hectare	2.471053815	1	107639.1042	15500031	10000	0.0038861022	11959.90046
Square Foot	2.29568E-05	9.2903E-06	1	144	0.09290304	3.58701E-08	0.111111111
Square Inch	1.59423E-07	6.4516E-08	0.006944444	1	0.00064516	2.49098E-10	0.000771605
Square Meter	0.000247105	1E-04	10.76391042	1550.0031	1	3.86102E-07	1.195990046
Square Mile	640	258.998811	27878400	4014489600	2589988.11	1	3097599.998
Square Yard	0.000206612	8.36127E-05	9	1296	0.836127361	3.22831E-07	1

TABLE 10-6 VOLUME MEASUREMENT CONVERSION FACTORS

	Cubic Foot	Cubic Inch	Cubic Meter	Cubic Yard
Cubic Foot	1	1728	0.028316847	0.037037037
Cubic Inch	0.000578704	1	1.63871E-05	2.14335E-05
Cubic Meter	35.31466672	61023.74409	1	1.307950618
Cubic Yard	27	46656	0.764554859	1

TABLE 10-7 FORCE CONVERSION FACTORS

	Dyne	Newton	Pound Force
Dyne	1	0.00001	2.25E-06
Newton	100000	1	0.224809
Pound force	444822.2	4.448222	1

TABLE 10-8 ENERGY CONVERSION FACTORS

	BTU	Calorie (IT)	Calorie (Th'mic)	Electron Volt	Erg	Foot-pound	Horsepower-hour	Joule	Watt-hour
BTU	1	251.9966	252.1655	6.59E+21	1.06E+10	25036.98	0.000393	1055.058	0.293072
Calorie (IT)	0.003968	1	1.00067	2.61E+19	41867928	99.35441	1.56E-06	4.186795	0.001163
Calorie (Th'mic)	0.003966	0.99933	1	2.61E+19	41839890	99.28787	1.56E-06	4.183991	0.001162
Electron volt	1.52E-22	3.83E-20	3.83E-20	1	1.6E-12	3.8E-18	5.97E-26	1.6E-19	4.45E-23
Erg	9.48E-11	2.39E-08	2.39E-08	6.24E+11	1	2.37E-06	3.73E-14	1E-07	2.78E-11
Foot-pound	3.99E-05	0.010065	0.010072	2.63E+17	421399.8	1	1.57E-08	0.04214	1.17E-05
Horsepower-hour	2544.426	641186.8	641616.4	1.68E+25	2.68E+13	63704732	1	2684517	745.6997
Joule	0.000948	0.238846	0.239006	6.24E+18	9999995	23.73042	3.73E-07	1	0.000278
Watt-hour	3.412133	859.8459	860.4221	2.25E+22	3.6E+10	85429.48	0.001341	3599.998	1

TABLE 10-9 TIME CONVERSION FACTORS

	Day	Hour	Minute	Second	Year
Day	1	24	1440	86400	0.002738
Hour	0.041667	1	60	3600	0.000114
Minute	0.000694	0.016667	1	60	1.9E-06
Second	1.16E-05	0.000278	0.016667	1	3.17E-08
Year	365.25	8766	525960	31557600	1

Temperature Conversions

This section presents formulas for conversion among three units of temperature: Fahrenheit, Celsius, and Kelvin. Temperature conversions, unlike the unit conversions discussed previously in this chapter, do not use a simple conversion factor. Rather, you need to use a formula to calculate the conversion. The formulas in Table 10-10 assume that the temperature for conversion is in a cell named *temp*.

TABLE 10-10 TEMPERATURE CONVERSION FORMULAS

Type of Conversion	Formula
Fahrenheit to Celsius	$=(temp-32)*(5/9)$
Fahrenheit to Kelvin	$=(temp-32)*(5/9)+273$
Celsius to Fahrenheit	$=(temp*1.8)+32$
Celsius to Kelvin	$=temp+273$
Kelvin to Celsius	$=temp-273$
Kelvin to Fahrenheit	$=((temp-273)*1.8)+32$

Solving Right Triangles

A right triangle has six components: three sides and three angles. Figure 10-1 shows a right triangle with its various parts labeled. Angles are labeled A, B, and C; sides are labeled Hypotenuse, Base, and Height. Angle C is always 90 degrees (or $\pi/2$ radians). If you know any two of these components (excluding Angle C, which is always known), you can use formulas to solve for the others.

Need to Convert Other Units?

This chapter, of course, doesn't list every possible unit conversion factor. To calculate other unit conversions, you need to find the appropriate conversion factor. The Internet is a good source for such information. Use any Web search engine and enter search terms that correspond to the units you use. Likely, you'll find the information you need.

Also, you can download a copy of Josh Madison's popular (and free) Convert software. This excellent program can handle just about any conceivable unit conversion you throw at it. The URL is as follows:

<http://www.joshmadison.com/software/>

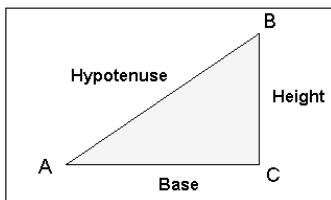
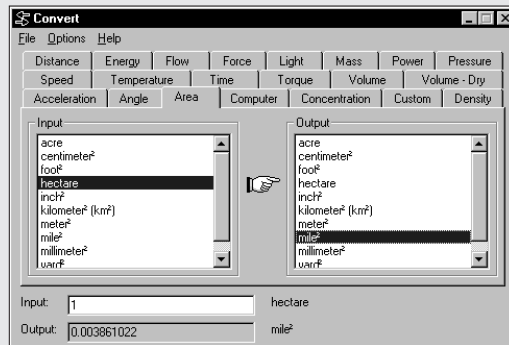


Figure 10-1: A right triangle's components

The Pythagorean theorem states that

$$\text{Height}^2 + \text{Base}^2 = \text{Hypotenuse}^2$$

Therefore, if you know two sides of a right triangle, you can calculate the remaining side. The formula to calculate a right triangle's height (given the length of the hypotenuse and base) is shown below.

$$=\text{SQRT}(\text{hypotenuse}^2 - \text{base}^2)$$

The formula to calculate a right triangle's base (given the length of the hypotenuse and height) is as follows:

```
=SQRT((hypotenuse^2)-(height^2))
```

The formula to calculate a right triangle's hypotenuse (given the length of the base and height) is as follows:

```
=SQRT( (height^2)+(Base_Length^2))
```

Other useful trigonometric identities are:

SIN(A) = Height/Hypotenuse

SIN(B) = Base/Hypotenuse

COS(A) = Base/Hypotenuse

COS(B) = Height/Hypotenuse

TAN(A) = Height/Base

SIN(A) = Base/Height



Excel's trigonometric functions all assume that the angle arguments are in radians. To convert degrees to radians, use the RADIANS function. To convert radians to degrees, use the DEGREES function.

If you know the height and base, you can use the following formula to calculate the angle formed by the hypotenuse and base (Angle A).

```
=ATAN(height/base)
```

The preceding formula returns radians. To convert to degrees, use this formula:

```
=DEGREES(ATAN(height/base))
```

If you know the height and base, you can use the following formula to calculate the angle formed by the hypotenuse and height (Angle B):

```
=PI()/2-ATAN(height/base)
```

The preceding formula returns radians. To convert to degrees, use this formula:

```
=90-DEGREES(ATAN(height/base))
```



The companion CD-ROM contains a workbook with formulas that calculate various parts of a right triangle, given two known parts. These formulas give you some insight on working with right triangles.

Figure 10-2 shows a workbook containing formulas to calculate the various parts of a right triangle.

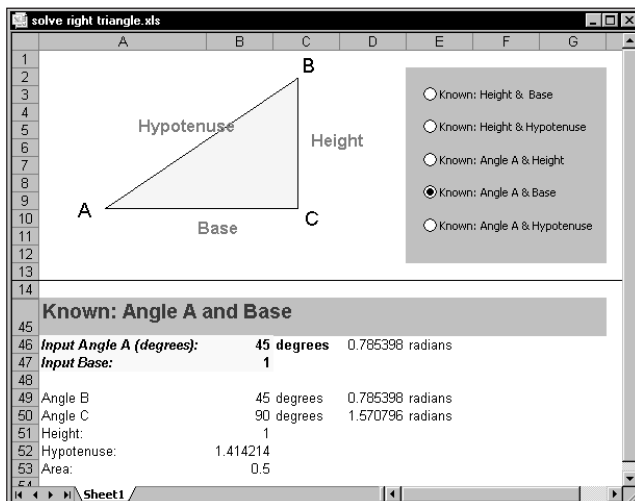


Figure 10-2: This workbook is useful for working with right triangles.

Area, Surface, Circumference, and Volume Calculations

This section contains formulas for calculating the area, surface, circumference, and volume for common two- and three-dimensional shapes.

Calculating the Area and Perimeter of a Square

To calculate the area of a square, square the length of one side. The following formula calculates the area of a square for a cell named *side*.

```
=side^2
```

To calculate the perimeter of a square, multiply one side by 4. The following formula uses a cell named *side* to calculate the perimeter of a square.

```
=side*4
```

Calculating the Area and Perimeter of a Rectangle

To calculate the area of a rectangle, multiply its height by its base. The following formula returns the area of a rectangle, using cells named *height* and *base*.

```
=height*base
```

To calculate the perimeter of a rectangle, multiply the height by 2, and add it to the width multiplied by 2. The following formula returns the perimeter of a rectangle, using cells named *height* and *width*.

```
=(height*2)+(width*2)
```

Calculating the Area and Perimeter of a Circle

To calculate the area of a circle, multiply the square of the radius by π . The following formula returns the area of a circle. It assumes that a cell named *radius* contains the circle's radius.

```
=PI()*(radius^2)
```

The radius of a circle is equal to one-half of the diameter.

To calculate the circumference of a circle, multiply the diameter of the circle by π . The following formula calculates the circumference of a circle using a cell named *diameter*.

```
=diameter*PI()
```

The diameter of a circle is the radius times 2.

Calculating the Area of a Trapezoid

To calculate the area of a trapezoid, add the two parallel sides, multiply by the height, and then divide by 2. The following formula calculates the area of a trapezoid, using cells named *side* and *height*.

```
=((side*2)*height)/2
```

Calculating the Area of a Triangle

To calculate the area of a triangle, multiply the base by the height, and then divide by 2. The following formula calculates the area of a triangle, using cells named *base* and *height*.

```
=(base*height)/2
```

Calculating the Surface and Volume of a Sphere

To calculate the surface of a sphere, multiply the square of the radius by π , and then multiply by 4. The following formula returns the surface of a sphere, the radius of which is in a cell named *radius*.

```
=PI()*(radius^2)*4
```

To calculate the volume of a sphere, multiply the cube of the radius by 4 times π , and then divide by 3. The following formula calculates the volume of a sphere. The cell named *radius* contains the sphere's radius.

```
=((radius^3)*(4*PI()))/3
```

Calculating the Surface and Volume of a Cube

To calculate the surface area of a cube, square one side and multiply by 6. The following formula calculates the surface of a cube using a cell named *side*, which contains the length of a side of the cube.

```
=(side^2)*6
```

To calculate the volume of a cube, raise the length of one side to the third power. The following formula returns the volume of a cube, using a cell named *side*.

```
=side^3
```

Calculating the Surface and Volume of a Cone

The following formula calculates the surface of a cone (including the surface of the base). This formula uses cells named *radius* and *height*.

```
=PI()*radius*(SQRT(height^2+radius^2)+radius)
```

To calculate the volume of a cone, multiply the square of the radius of the base by π , multiply by the height, and then divide by 3. The following formula returns the volume of a cone, using cells named *radius* and *height*.

```
=(PI()*(radius^2)*height)/3
```

Calculating the Volume of a Cylinder

To calculate the volume of a cylinder, multiply the square of the radius of the base by π , and then multiply by the height. The following formula calculates the volume of a cylinder, using cells named *radius* and *height*.

```
=(PI()*(radius^2)*height)
```

Calculating the Volume of a Pyramid

Calculate the area of the base, then multiply by the height and divide by 3. This next formula calculates the volume of a pyramid. It assumes cells named *width* (the width of the base), *length* (the length of the base), and *height* (the height of the pyramid).

```
=(width*length*height)/3
```

Solving Simultaneous Equations

This section describes how to use formulas to solve simultaneous linear equations. The following is an example of a set of simultaneous linear equations.

$$\begin{aligned} 3x + 4y &= 8 \\ 4x + 8y &= 1 \end{aligned}$$

Solving a set of simultaneous equations involves finding the values for x and y that satisfy both equations. For this set of equations, the solution is as follows:

$$\begin{aligned} x &= 7.5 \\ y &= -3.625 \end{aligned}$$

The number of variables in the set of equations must be equal to the number of equations. The preceding example uses two equations with two variables. Three equations are required to solve for three variables (x , y , and z).

The general steps for solving a set of simultaneous equations follow. See Figure 10-3, which uses the equations presented at the beginning of this section.

1. Express the equations in standard form. If necessary, use simple algebra to rewrite the equations such that the variables all appear on the left side of the equal sign. The two equations that follow are identical, but the second one is in standard form.

$$\begin{aligned} 3x - 8 &= -4y \\ 3x + 4y &= 8 \end{aligned}$$

2. Place the coefficients in an n -by- n range of cells, where n represents the number of equations. In Figure 10-3, the coefficients are in the range G6:H7.
3. Place the constants (the numbers on the right side of the equal sign) in a vertical range of cells. In Figure 10-3, the constants are in the range J6:J7.

- Use an array formula to calculate the inverse of the coefficient matrix. In Figure 10-3, the following array formula is entered into the range G10:H11.

```
{=MINVERSE(G6:H7)}
```

- Use an array formula to multiply the inverse of the coefficient matrix by the constant matrix. In Figure 10-3, the following array formula is entered into the range H14:H15. This range holds the solution.

```
{=MMULT(G10:H11,J6:J7)}
```



Chapter 16 demonstrates how to use iteration to solve some simultaneous equations.

simultaneous equations.xls											
	A	B	C	D	E	F	G	H	I	J	K
1	Linear equations in standard form:										
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											

Figure 10-3: Using formulas to solve simultaneous equations



You can access the workbook shown in Figure 10-3 on the companion CD-ROM. This workbook solves simultaneous equations with two or three variables.

Rounding Numbers

Excel provides quite a few functions that round values in various ways. Table 10-11 summarizes these functions.



It's important to understand the difference between rounding a value and formatting a value. When you format a number to display a specific number of decimal places, formulas that refer to that number use the actual value, which may differ from the displayed value. When you round a number, formulas that refer to that value use the rounded number.

TABLE 10-11 EXCEL'S ROUNDING FUNCTIONS

Function	Description
CEILING	Rounds a number up (away from zero) to the nearest specified multiple
DOLLARDE*	Converts a dollar price expressed as a fraction into a decimal number
DOLLARFR*	Converts a dollar price expressed as a decimal into a fractional number
EVEN	Rounds a number up (away from zero) to the nearest even integer
FLOOR	Rounds a number down (toward zero) to the nearest specified multiple
INT	Rounds a number down to make it an integer
MROUND*	Rounds a number to a specified multiple
ODD	Rounds a number up (away from zero) to the nearest odd integer
ROUND	Rounds a number to a specified number of digits
ROUNDDOWN	Rounds a number down (toward zero) to a specified number of digits
ROUNDUP	Rounds a number up (away from zero) to a specified number of digits
TRUNC	Truncates a number to a specified number of significant digits

*This function is available only when the Analysis ToolPak add-in is installed.



Chapter 6 contains examples of rounding time values.

The following sections provide examples of formulas that use various types of rounding.

Basic Rounding Formulas

The ROUND function is useful for basic rounding to a specified number of digits. You specify the number of digits in the second argument for the ROUND function. For example, the formula that follows returns 123.40 (the value is rounded to one decimal place).

```
=ROUND(123.37,1)
```

If the second argument for the ROUND function is zero, the value is rounded to the nearest integer. The formula that follows, for example, returns 123.00.

```
=ROUND(123.37,0)
```

The second argument for the ROUND function can also be negative. In such a case, the number is rounded to the left of the decimal point. The following formula, for example, returns 120.00.

```
=ROUND(123.37,-1)
```

The ROUND function rounds either up or down. But how does it handle a number such as 12.5, rounded to no decimal places? You'll find that the ROUND function rounds such numbers away from zero. The formula that follows, for instance, returns 13.0.

```
=ROUND(12.5,0)
```

The next formula returns -13.00 (the rounding occurs away from zero).

```
=ROUND(-12.5,0)
```

To force rounding to occur in a particular direction, use the ROUNDUP or ROUNDDOWN functions. The following formula, for example, returns 12.0. The value rounds down.

```
=ROUNDDOWN(12.5,0)
```

The formula that follows returns 13.0. The value rounds up to the nearest whole value.

```
=ROUNDUP(12.43,0)
```

Rounding to the Nearest Multiple

The MROUND function (part of the Analysis ToolPak add-in) is useful for rounding values to the nearest multiple. For example, you can use this function to round a number to the nearest 5. The formula below returns 135.

```
=MROUND(133,5)
```

Rounding Dollar Values

Often, you need to round dollar values to the nearest penny. For example, a calculated price may be something like \$45.78923. In such a case, you'll want to round the calculated price to the nearest penny. This may sound simple, but there are actually three ways to round such a value:

- ◆ Round it up to the nearest penny.
- ◆ Round it down to the nearest penny.
- ◆ Round it to the nearest penny (the rounding may be up or down).

The following formula assumes a dollar and cents value is in cell A1. The formula rounds the value to the nearest penny. For example, if cell A1 contains \$12.421, the formula returns \$12.42.

```
=ROUND(A1,2)
```

If you need to round the value up to the nearest penny, use the CEILING function. The following formula rounds the value in cell A1 up to the nearest penny. If, for example, cell A1 contains \$12.421, the formula returns \$12.43.

```
=CEILING(A1,0.01)
```

To round a dollar value down, use the FLOOR function. The following formula, for example, rounds the dollar value in cell A1 down to the nearest penny. If cell A1 contains \$12.421, the formula returns \$12.42.

```
=FLOOR(A1,0.01)
```

To round a dollar value up to the nearest nickel, use this formula:

```
=CEILING(A1,0.05)
```

Working with Fractional Dollars

The DOLLARFR and DOLLARDE functions are useful when working with fractional dollar value, as in stock market quotes. To access these functions, you must install the Analysis ToolPak add-in.

Consider the value \$9.25. You can express the decimal part as a fractional value (\$9 1/4, \$9 2/8, \$9 4/16, and so on). The DOLLARFR function takes two arguments: the dollar amount and the denominator for the fractional part. The following formula, for example, returns 9.1 (the .1 decimal represents 1/4).

```
=DOLLARFR(9.25,4)
```



It's important to understand that you cannot use the value returned by the DOLLARFR function in other calculations. In the preceding example, the result of the function will be interpreted as 9.1, not 9.25. To perform calculations on such a value, you need to convert it back to a decimal value by using the DOLLARDE function.

The DOLLARDE function converts a dollar value expressed as a fraction to a decimal amount. It also uses a second argument to specify the denominator of the fractional part. The following formula, for example, returns 9.25.

```
=DOLLARDE(9.1,4)
```



The DOLLARDE and DOLLARFR functions aren't limited to dollar values. For example, you can use these functions to work with feet and inches. You might have a value that represents 8.5 feet. Use the following formula to express this value in terms of feet and inches. The formula returns 8.06 (which represents 8 feet, six inches).

```
=DOLLARFR(8.5,12)
```

Another example is baseball statistics. A pitcher may work 6 and 2/3 innings, and this is usually represented as 6.2. The following formula displays 6.2:

```
=DOLLARFR(6+2/3,3)
```

Using the INT and TRUNC Functions

On the surface, the INT and TRUNC functions seem similar. Both convert a value to an integer. The TRUNC function simply removes the fractional part of a number.

The INT function rounds a number down to the nearest integer, based on the value of the fractional part of the number.

In practice, INT and TRUNC return different results only when using negative numbers. For example, the following formula returns -14.0.

```
=TRUNC(-14.2)
```

The next formula returns -15.0 because -14.3 is rounded down to the next lower integer.

```
=INT(-14.2)
```

The TRUNC function takes an additional (optional) argument that's useful for truncating decimal values. For example, the formula that follows returns 54.33 (the value truncated to two decimal places).

```
=TRUNC(54.3333333,2)
```

Rounding to an Even or Odd Integer

The ODD and EVEN functions are provided for situations in which you need to round a number up to the nearest odd or even integer. These functions take a single argument and return an integer value. The EVEN function rounds its argument up to the nearest even integer. The ODD function rounds its argument up to the nearest odd integer. Table 10-12 shows some examples of these functions.

TABLE 10-12 RESULTS USING THE EVEN AND ODD FUNCTIONS

Number	EVEN Function	ODD Function
-3.6	-4	-5
-3.0	-4	-3
-2.4	-4	-3
-1.8	-2	-3
-1.2	-2	-3
-0.6	-2	-1
0.0	0	1
0.6	2	1

Continued

TABLE 10-12 RESULTS USING THE EVEN AND ODD FUNCTIONS (Continued)

Number	EVEN Function	ODD Function
1.2	2	3
1.8	2	3
2.4	4	3
3.0	4	3
3.6	4	5

Rounding to n Significant Digits

In some cases, you may need to round a value to a particular number of significant digits. For example, you might want to express the value 1,432,187 in terms of two significant digits (that is, as 1,400,000). The value 9,187,877 expressed in terms of three significant digits is 9,180,000.

If the value is a positive number with no decimal places, the following formula does the job. This formula rounds the number in cell A1 to two significant digits. To round to a different number of significant digits, replace the 2 in this formula with a different number.

```
=ROUNDDOWN(A1,2-LEN(A1))
```

For non-integers and negative numbers, the solution gets a bit trickier. The formula that follows provides a more general solution that rounds the value in cell A1 to the number of significant digits specified in cell A2. This formula works for positive and negative integers and non-integers.

```
=ROUND(A1,A2-1-INT(LOG10(ABS(A1))))
```

For example, if cell A1 contains 1.27845 and cell A2 contains 3, the formula returns 1.28000 (the value, rounded to three significant digits).

Summary

This chapter covered several topics: unit conversions, trigonometric formulas, calculations for various two- and three-dimensional shapes, simultaneous equations, and rounding.

In the next chapter, we turn to array formulas.

Part III

Financial Formulas

CHAPTER 11

Introducing Financial Formulas

CHAPTER 12

Discounting and Depreciation Financial Functions

CHAPTER 13

Advanced Uses of Financial Functions and Formulas

Chapter 11

Introducing Financial Formulas

IN THIS CHAPTER

- ◆ Introducing the fundamental concept of time value of money
- ◆ Using Excel's basic financial functions PV, FV, NPER, PMT, and RATE
- ◆ Converting nominal and effective interest rates
- ◆ Calculating effective cost of loans using different rate quotation systems
- ◆ Calculating cumulative payments of interest and principal using the CUMIPMT and CUMPRINC functions
- ◆ Matching different interest and payment frequencies
- ◆ Understanding the limitations of the PV, FV, NPER, PMT, RATE, CUMIPMT, and CUMPRINC functions

IT'S A SAFE BET that the most common use of Excel is to perform calculations involving money. Every day, people make hundreds of thousands of financial decisions based on the numbers that are calculated in a spreadsheet. These decisions range from simple (*Can I afford to buy a new car?*) to complex (*Will purchasing XYZ Corporation result in a positive cash flow in the next 18 months?*). This is the first of three chapters that discuss financial calculations that you can perform with the assistance of Excel.

Excel's Basic Financial Functions

This chapter presents many examples that use Excel's five basic financial functions. The syntax for these functions is shown here (arguments in bold are required arguments):

- ◆ PV(rate, nper, pmt, fv, type)
- ◆ FV(rate, nper, pmt, pv, type)
- ◆ PMT(rate, nper, pv, fv, type)

- ◆ RATE(nper, pmt, pv, fv, type, guess)
- ◆ NPER(rate, pmt, pv, fv, type)

As you'll see, these functions are extremely flexible, and are useful for a wide variety of problems. To use these function effectively, you will need to understand three basic concepts:

- ◆ Signing of money flows as positive or negative
- ◆ The basic concept of time value of money
- ◆ The concept of equivalent interest rates

These concepts are all covered in this chapter and will be put to further use in subsequent chapters.

Basic Terminology

- ◆ Present Value (PV): This is the *principal* amount. If you invest \$5,000 in a bank CD (certificate of deposit), this amount represents the principal, or present value, of the money you invested. If you borrow \$15,000 to purchase a car, this amount represents the principal or present value of the loan. Present Value may be positive or negative.
- ◆ Future Value (FV): This is the principal plus interest. If you invest \$5,000 for five years and earn 6% annual interest, you receive \$6,312.38 at the end of the five-year term. The amount is the future value of your \$5,000 investment. If you take out a three-year auto loan for \$15,000 and pay 7% annual interest, you pay a total of \$16,673.16. This amount represents the principal plus the interest you paid. Future Value may be either positive or negative.
- ◆ Payment (PMT): This is either principal, or principal plus interest. If you deposit \$100 per month into a savings account, \$100 is the payment. If you have a monthly mortgage payment of \$825, the \$825 is made up of principal and interest.
- ◆ Interest Rate: Interest is a percentage of the principal, usually expressed on an annual basis. For example, you might earn 5.5% annual interest on a bank CD. Or your mortgage loan may have a 7.75% interest rate.
- ◆ Period: This represents the point in time when interest is paid or earned. For example, a bank CD that pays interest quarterly or an auto loan that requires monthly payments.
- ◆ Term: This is the amount of time of interest. A 12-month bank CD has a term of one year. A 30-year mortgage loan has a term of 30 years.

Signing of Money Flows Convention

Look at your bank statement, and it will become very apparent that money flows! When dealing with Excel's financial functions, it is critical that you understand how to "sign" cash flows. In other words, do you use a positive sign or a negative sign?

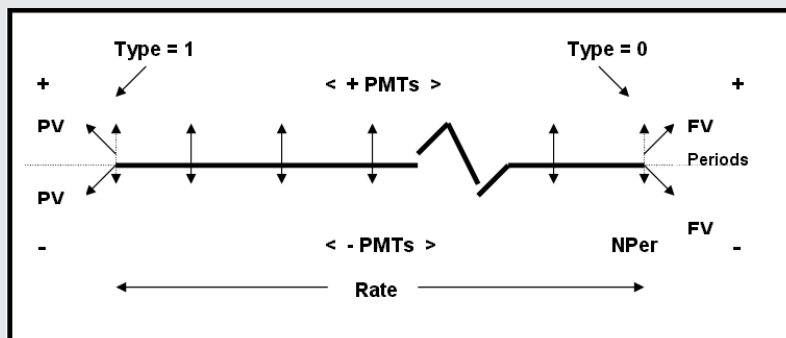
The Relationship between NPER, PMT, and RATE

Excel "knows" nothing about different time periods such as months, weeks, or years. It merely counts them and expects you to label them appropriately and to make sure that you don't mix them up.

The accompanying diagram represents the time value of the money concept used by the Excel functions PV, FV, PMT, NPER, and RATE. The arrows represent flows of money, and their direction (positive or negative). Any solvable problem consists of four known variables and one unknown variable. The unknown variable is the function name, and the known variables represent the function arguments.

The diagram must be in balance in terms of discounted or accumulated negative and positive flows. The concept allows only a single rate of interest, which must be the effective rate for the period of time measured by NPER. Similarly, only one level of payment is allowed, and that must be a payment per period of time measured by NPER. The Type argument in the concept shows whether payments are in advance or in arrears.

If you can fill in four of the five variables, Excel can solve the problem. There's one exception: If payments are involved, Excel needs to know when the payments occur (that is, the Type argument).



To solve financial problems using Excel's basic financial functions, you need to perform two preliminary steps:

1. Determine the perspective of the owner of the cash flows. For example, in a simple accumulation problem, are you looking at it from the perspective of the depositor or the bank? In a mortgage problem, are you the borrower or the lender? When calculating the value of a series of future payments, are you the purchaser (paying out for the right to receive), or are you the seller (receiving a payment for giving up that right)?
2. Determine whether any particular present value, payment, or future value comes towards you (positive sign), or goes away from you (negative sign).

When you have a firm handle on these two points, you'll be able to use Excel's financial functions to create effective financial formulas – and be able to interpret the results returned by the formulas.

Generally, money that comes in to you is signed positive. Money that goes away from you is signed negative. For example, if a present value problem returns a negative value, it means that this amount is paid out at time-period zero. If it is positive, the money is received. Consider an example of calculating mortgage payments. If you are the borrower, the loan “comes towards you,” and the calculated payments have a negative sign (which indicates that you pay them out). When calculating the rate of interest on a mortgage loan, you must take care to sign the loan value and the payments properly. Otherwise, Excel will assume that they are all in one direction and will generate an error. For example, a formula may display #NUM!, which indicates an infinitely high rate of return (everything comes towards you and nothing is paid out for it).

Accumulation, Discounting, and Amortization Functions

This section contains a number of examples that demonstrate the use of Excel's five basic functions to solve accumulation and amortization problems. Although we tend to look at amortization and accumulation as separate problems, they are essentially the same. In fact, the only difference is in the signing of the cash flows.

We can classify these problems into simple and complex problems. In simple problems, we are dealing with only two of the three cash variables (present value, payment, and future value). In complex problems, we are dealing with all three. Although we classify these as simple and complex problems, Excel still requires a value for all three of the cash variables. Therefore, we use zero for the “missing” element.

Simple Accumulation Problems

This section contains seven examples that demonstrate simple accumulation problems.



All of the examples in this section are available on the companion CD-ROM.

EXAMPLE 1

How much does \$1,000 accumulate to after three years, at 7% interest per year?

Figure 11-1 shows this problem set up on a worksheet.

	A	B	C	D	E	F	G	H
1	Example 1							
2	Question							
3								
4								
5								
6	Function							
7								
8								
9								
10								
11								
12	Data required for FV function							
13								
14	Rate			7%				
15	Nper			3				
16	PMT			0				
17	PV			(\$1,000.00)				
18	Type			0	Irrelevant			
19								
20	Answer							
21								
22	FV				\$1,225.04			

Figure 11-1: Calculating a future value

Function required: `FV(rate, nper, pmt, pv, type)`

This formula returns \$1,225.04:

`=FV(7%, 3, 0, -1000, 0)`



The formula examples in this chapter use hard-coded values for function arguments. The examples on the companion CD-ROM use cell references for the function arguments.

Note that this problem is stated from the perspective of the depositor. Therefore, the initial deposit (the pv argument) is negative. No regular payments are made, so the pmt argument is 0. With no payments, the type argument is irrelevant.



When entering numeric data as function arguments, make sure that you don't insert thousands separators. For example, type **1000**, not **1,000**. Depending on your regional settings, the thousands separator may be the same character as the argument separator.

EXAMPLE 2

If \$1,000 has accumulated to \$2,000 in eight years, what has been the average annual growth rate?

Function required: RATE(nper, pmt, pv, fv, type, guess)

This formula returns 9.050773%:

```
=RATE(8,0,-1000,2000,0)
```

This example is from the perspective of the depositor, so the pv argument is negative and the fv argument (a right to receive) is positive. Because the term was expressed in years, the rate is the effective rate per annum.

EXAMPLE 3

If I deposit \$100,000 and can earn 14% per annum, how long will it take me to become a millionaire?

Function required: NPER(rate, pmt, pv, fv, type)

This formula returns 17.573:

```
=NPER(14%,0,-100000,1000000,0)
```

This example is from the perspective of a depositor. Therefore, the pv argument is negative and the fv argument (the right to receive the \$1 million) is positive. Because the rate is quoted in annual effective terms, the result is in years.

EXAMPLE 4

If I have \$10,573.45 in my account and I have earned 1% interest per month for 12 months, what was the original deposit?

Function required: PV(rate, nper, pmt, fv, type)

This formula returns -\$9,383.40:

```
=PV(1%,12,0,10573.45,0)
```

With no regular payments, the *pmt* argument is 0 and the *type* argument is irrelevant. Because the \$10,573.45 in the account is a right to receive, the *fv* argument takes a positive sign and the calculated present value is negative.

EXAMPLE 5

If I deposit \$300 per month (starting today) in an account earning 1% per month, how much will I have after two years?

Function required: **FV(rate, nper, pmt, pv, type)**

This formula returns \$8,172.96:

`=FV(1%,24,-300,0,1)`

In this example, the term is quoted in years, but the interest and payments are monthly. This requires a preliminary calculation. The most direct approach is to convert years to months. Another option is to convert the interest rate to an annual effective rate, and then convert the \$300 to an equivalent amount per year. This would produce the same result, but it is an overly complicated approach.

Note that payments start “today” and are, therefore, in advance. Consequently, the *type* argument is 1. No present balance is stated, so the *pv* argument is 0.

In all of the preceding examples, the questions can be rephrased such that the negatives become positives, and the positives become negatives. Therefore, Example 1 can be rephrased as follows.

EXAMPLE 6

If I borrow \$1,000 for three years at 7% interest, how much do I have to pay back?

Function required: **FV(rate, nper, pmt, pv, type)**

This formula returns -\$1,225.04:

`=FV(7%,3,0,1000,0)`

Here the question is from the perspective of the borrower, and the formula has been modified such that the initial borrowing (the *pv* argument) is positive. No regular payments are made, so the *pmt* argument is 0. With no payments, the *type* argument is irrelevant.

Examples 2 through 5 can also be rephrased as such: The depositor becomes the borrower, and the borrower becomes the depositor.

EXAMPLE 7

If \$1,000 has accumulated to \$3,000 in eight years, what has been the average annual growth rate?

Function required: **RATE(nper, pmt, pv, fv, type, guess)**

This formula returns 14.720269%:

`=RATE(8,0,-1000,3000,0)`

This example is from the perspective of the depositor. Therefore, the *pv* argument is negative and the *fv* argument (a right to receive) is positive. Because the term was expressed in years, the rate is the effective rate per annum. With no regular payments, the *pmt* argument is 0 and the *type* argument is irrelevant.



An important feature of financial calculations is that they can be cross-checked to establish the accuracy of the answer. This can be done “off spreadsheet” using a financial calculator, or it can be done using the underlying formula or another function.

The following steps demonstrate a method to verify the result of 14.720269% for this example:

1. Calculate how much \$1,000 accumulates to in eight years at the calculated rate. This formula returns \$3,000:

$$=FV(14.720269\%, 8, 0, -1000, 0)$$
2. Calculate the present value of \$3,000, discounting at the calculated rate for eight years. The following formula returns -\$1,000:

$$=PV(14.720269\%, 8, 0, 3000, 0)$$
3. Calculate how long it takes \$1,000 to accumulate to \$3,000 at the calculated rate. The following formula returns eight:

$$=NPER(14.720269\%, 0, -1000, 3000, 0)$$
4. Calculate the result using the following formula, which returns 14.720269%:

$$=(3000 / -1000)^{(1/8)} - 1$$

One technique for cross-checking is to compare the check calculation with the original data in such a way that the method produces an error of 0. In all of the previous checks, subtracting the original data from the check calculation produces an error of zero. If all calculations are checked and errors calculated this way, the sum of all errors on a spreadsheet will approach zero. It is unlikely to be exactly zero because of rounding errors.



The examples on the CD-ROM contain error-checking formulas.

Complex Accumulation Problems

This section describes four examples of complex accumulation problems. There are two types of complex accumulation problems:

- ◆ Problems that have non-zero values for any two of the key parameters (present value, payment, and future value), and require a solution for the third parameter.
- ◆ Problems that have non-zero inputs for all three parameters (present value, payment, and future value), and require a solution for either RATE or NPER.



All of the examples in this section are available on the companion CD-ROM, along with a cross-check to ensure their accuracy.

EXAMPLE 8

With a beginning balance of \$5,500 and payments of \$500 per month (at the end of each month), how much will I accumulate over three years if I earn 0.75% per month?

Figure 11-2 shows this example, set up on a worksheet.

	A	B	C	D	E	F	G	H	I
1	Example 8								
2	Question								
3									
4	With a beginning balance of \$5,500 and payments of \$500 per month (at the end of each month), how much will I accumulate over three years if I earn 0.75% per month?								
5									
6	Function								
7									
8	FV(Rate,Nper,PMT,PV,Type)								
9									
10	Data required for FV function								
11									
12	Rate		0.750%						
13	Nper		36						
14	PMT		-\$ 500.00						
15	PV		-\$ 5,500.00						
16	Type		0						
17									
18	Answer								
19									
20	FV		\$27,773.91						
21									

Figure 11-2: Calculating a future value

Function required: FV(rate, nper, pmt, pv, type)

This formula returns \$27,773.91:

```
=FV(.75%,36,-500,-5500,0)
```

The negative sign for the pv argument may be confusing, because it represents a current balance (a right to receive). However, because we are looking forward in time, it is treated as a deposit. Payments and rates are quoted on a monthly basis; therefore, the term of three years must be converted to months. The FV is returned as positive, which is a right to receive.

EXAMPLE 9

My account balance five years ago was \$25,000, and I have added \$4,500 at the end of each year. The present balance is \$70,000. What has been my average annual return?

Function required: RATE(nper, pmt, pv, fv, type, guess)

This formula returns 10.9382%:

```
=RATE(5,-4500,-25000,70000,0,0)
```



RATE is a particularly powerful function, because the solution can only be obtained by iteration. Only rarely is it necessary to insert a guess rate as the optional sixth argument. If omitted, Excel supplies the default guess of 0.

EXAMPLE 10

My account has an overdraft of \$12,000 and I deposit \$1,000 at the end of each month. How long will it take me to become a millionaire if I earn an average return of 0.6% per month?

Function required: NPER(rate, pmt, pv, fv, type)

The following formula returns 337.78 months:

```
=NPER(6%,-1000,12000,1000000,0)
```

Note that the question is phrased such that the overdraft is a deposit. Therefore, it requires the negative sign for the pv argument.

If the overdraft is viewed as a loan, the future value would be positive. In such a case, two calculations would be required if the overdraft rate was not equal to the deposit rate. First we would calculate time taken to achieve zero balance, and then we would calculate the time to achieve \$1 million.

Using rates of 0.8% for overdraft and 0.6% for deposits, this formula returns 337.96 months:

```
=NPER(0.8%, -1000, 12000, 0, 0) + NPER(0.6%, -1000, 0, 1000000, 0)
```

EXAMPLE 11

I deposit \$1,000 per month (at the end of each month) and intend to do so for the next ten years. If I need to accumulate \$1,000,000, how much should I deposit now if the account earns 0.7% per month?

Function required: PV(rate, nper, pmt, fv, type)

This formula returns \$351,972.24:

```
=PV(0.7%, 120, -1000, 1000000, 0)
```

We need to convert years to months to ensure matching of the pmt, rate, and nper arguments.

If you've worked through the first 11 examples, you should be getting the hang of the process:

1. Determine the function required.
2. Determine the signs of pmt, pv, and fv inputs.
3. Ensure that periods of time for rate, nper, and pmt are the same (or convert them to make them the same).
4. Insert the arguments in the correct order (preferably by using cell references).
5. Consider the meaning of the answer.
6. Determine which function or calculations are required for a cross-check.
7. Ensure that the error approaches zero.

Simple Discounting Problems

You can think of discounting as “accumulation in reverse.” Rather than accumulating a present value to a future value, we are determining the present worth of a future amount.

As with accumulations, we can have problems that involve two or three of the monetary values of PV, FV, or PMT. Where only two are involved, we call it *simple discounting* and with all three involved, we call it *complex discounting*.



All of the examples in this section are available on the companion CD-ROM. Each example also contains a cross-check to ensure the accuracy of the calculation.

EXAMPLE 12

What is the present value of the right to receive \$25,000 in five years, discounting at 6.5% per annum?

Figure 11-3 shows this example, set up on a worksheet.

	A	B	C	D	E	F	G	H	I
1	Example 12								
2	Question								
3									
4	What is the present value of the right to receive \$25,000 in five years, discounting at 6.5% per annum?								
5									
6	Function								
7									
8	PV(Rate,Nper,PMT,FV,Type)								
9	But in this case there are no payments PMT takes value of 0 and Type is 0 (but irrelevant)								
10									
11									
12	Data required for FV function								
13									
14	Rate		6.5%						
15	Nper		5						
16	PMT		0						
17	PV		\$25,000.00						
18	Type		0	irrelevant					
19									
20	Answer								
21									
22	PV		-\$18,247.02						

Figure 11-3: Calculating a present value

Function required: PV(rate, nper, pmt, fv, type)

This formula returns -\$18,247.02:

```
=PV(6.5%,5,0,25000,0)
```

Note the logic of the signs. If we have a right to receive, the fv argument is positive—we must pay out in the present to receive this positive right in the future. With no payment, the type argument is irrelevant.

The accuracy of the computation can be assured by cross-checking the answer with another function. In this case, we might check whether \$18,247.02 will accumulate to \$25,000 in five years at 6.5%. The following cross-check formula does indeed return \$25,000:

```
=FV(6.5%,5,0,-18247.02,0)
```

EXAMPLE 13

A property yields a rental of \$25,000 for the next 25 years. If I discount at 8%, how much should I pay? Assume a zero value after 25 years and that rent is paid annually in arrears.

Function required: PV(rate, nper, pmt, fv, type)

The following formula returns -\$266,869.40:

```
=PV(8%,25,25000,0,0)
```

This result can be checked using the RATE function. This formula returns 8.00%:

```
=RATE(25,25000,-266869.40,0,0)
```

Typically, real estate payments are made in advance. In such a case, Example 13 would be modified by making the Type argument 1.

EXAMPLE 14

Assume that the Example 13 rent of \$25,000 is received in perpetuity. If we discount at 8%, how much should we pay?

This is an example of a discounting problem that Excel can't solve using its functions. The problem is that we can't use "perpetuity" as the nper argument. The solution is to use a very long time period, such as 1,000 years. The result is certainly accurate enough for most purposes.

Function required: PV(rate, nper, pmt, fv, type)

The following formula returns -\$312,500.00:

```
=PV(8%,1000,25000,0,0)
```

Another option is to use a formula to calculate the present value:

```
PV = PMT/RATE
```

For this example, the following formula returns \$312,500.00:

```
=25000/0.08
```

Note that the sign is different because the formula has not adopted the strict sign convention.

If rent is paid in advance, we merely adapt this "cheating" approach by using 1 for the Type argument. The following formula returns \$337,500.00:

```
=PV(8%,1000,25000,0,1)
```

The formula approach is varied, and the general formula for valuing income in advance is as follows:

$$PV = PMT * (1 + RATE) / RATE$$

For this example, the following formula returns \$337,500.00:

$$=25000 * (1 + .08) / .08$$

Other examples can be expressed in discounting terms, but were covered earlier in the “Simple Accumulation Problems” section.

EXAMPLE 15

A property currently worth \$2,000,000 is subject to a lease at a peppercorn rent for five years. A purchaser has paid \$1,750,000 for it. Assuming no future growth in value, what was the discount rate?



A peppercorn rent is a nominal rent that is intended to demonstrate that a property is leasehold and not freehold.

Function required: `RATE(nper, pmt, pv, fv, type, guess)`

The following formula returns 2.706609%:

$$=RATE(5, 0, -1750000, 2000000, 0)$$

The payment today represents a negative present value. The value in five years is a (positive) right to receive.

To check the answer, use this formula (which returns \$2,000,000.03):

$$=FV(2.706609\%, 5, -1750000, 0)$$

The rounding error is caused by hard-coding the rate to only six decimal places. Normally, the argument would be a cell reference, not a hard-coded value.

EXAMPLE 16

A leasehold interest in a property was recently sold for \$230,000. The lease had four years to run, and rent was payable at \$6,000 per month in advance without rent review or escalation. If we accept a yield of 0.75%, what profit rent is shown by the transaction? Profit rent is the rental value minus the rent paid.

Function required: PMT(rate, nper, pv, fv, type)
 The following formula returns \$5,680.95:

```
=PMT(0.75%, 48, -230000, 0, 1)
```

Adding the rent paid (\$6,000) produces a rental value of \$11,680.95.

Complex Discounting Problems

Complex discounting problems involve the use of all three monetary amounts: present value, payment, and future value. The examples of complex discounting in this section are essentially re-expressions of the complex accumulation problems.



All the examples in this section are available on the companion CD-ROM.

EXAMPLE 17

If I discount at 0.75% per month, how much should I pay for a property yielding \$25,000 per month in advance (which I estimate will be worth \$5,000,000 in five years)?

Function required: PV(rate, nper, pmt, fv, type)
 The following formula returns -\$4,406,865.34:

```
=PV(0.75%, 60, 25000, 5000000, 1)
```

This example uses a rate per month, and payments are monthly. Therefore, the nper argument has been converted to months.

We can check this calculation by using the RATE function. The following formula returns 0.75%:

```
=RATE(60, 25000, -4406865.34, 5000000, 1)
```

EXAMPLE 18

I paid \$1,200,000 for a property that yields a rent of \$12,000 per month in advance. If I sell it in five years for \$1,500,000, what yield will I receive?

Function required: RATE(nper, pmt, pv, fv, type, guess)
 The following formula returns 1.29136%:

```
=RATE(60, 12000, -1200000, 1500000, 1)
```


This result can be verified by using the PV function. The following formula returns $-\$1,200,000.00$:

```
=PV(1.29136%,60,12000,1500000,1)
```

It's important to understand that the rent is quoted monthly in advance, but the term is five years. This discrepancy is resolved by converting the years to months. Therefore, the formula returns a monthly rate of interest.



Note that the rent is not converted to an annual rent. This is because a rent of \$12,000 per month in advance is not the same as a rent of \$144,000 per annum in advance. To achieve the equivalent annual amount we would need to know the rate of discount — which is the one piece of information we are trying to calculate.

EXAMPLE 19

A property has been purchased for \$1,600,000. It yields a rent of \$10,000 per month in advance. If I am to secure a yield of 1% per month, what must the property be worth in five years when I plan to sell it?

Function required: FV(rate, nper, pmt, pv, type)

This formula returns \$2,081,851.05:

```
=FV(1%,60,10000,-1600000,1)
```

This result can be verified using the following formula (which returns $-\$1,600,000$):

```
=PV(1%,60,10000,2081851.05,1)
```

Amortization Problems

Amortization is the term given to the process of paying back loans. This chapter, in fact, has already covered most of the calculations required, but the problems were expressed in terms of accumulation.

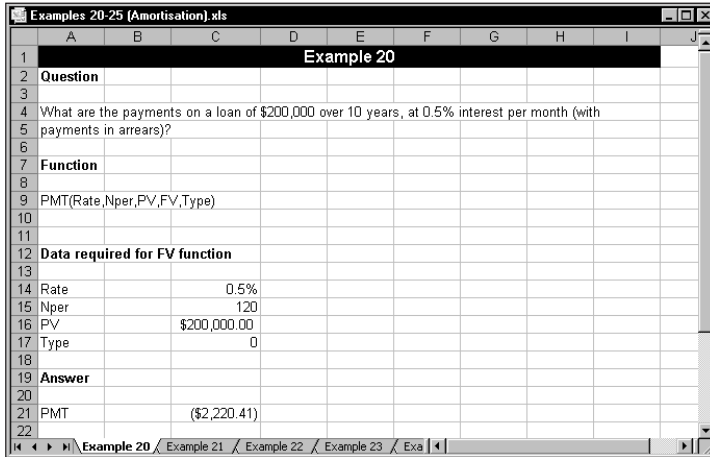


All the basic examples in this section are available on the companion CD-ROM.

EXAMPLE 20

What are the payments on a loan of \$200,000 over 10 years, at 0.5% interest per month (with payments in arrears)?

This example is illustrated in Figure 11-4.



	A	B	C	D	E	F	G	H	I
1	Example 20								
2	Question								
3									
4	What are the payments on a loan of \$200,000 over 10 years, at 0.5% interest per month (with								
5	payments in arrears)?								
6									
7	Function								
8									
9	PMT(Rate,Nper,PV,FV,Type)								
10									
11									
12	Data required for FV function								
13									
14	Rate		0.5%						
15	Nper		120						
16	PV		\$200,000.00						
17	Type		0						
18									
19	Answer								
20									
21	PMT		(\$2,220.41)						
22									

Figure 11-4: Calculating a loan payment

Function required: PMT(rate, nper, pv, fv, type)

The following formula returns \$2,220.41:

```
=PMT(0.5%,120,200000,0,0)
```

This result can be verified by using the PV function to calculate the loan amount. The following formula returns \$200,000:

```
=PV(0.5%,120,-2220.41,0,0)
```

In this example, the loan is fully repaid after 10 years, and the fv argument is zero. Also note that the payments are to be monthly, and the monthly loan rate has been quoted. Therefore, the 10-year term is converted to months.

EXAMPLE 21

I can afford payments of \$2,500 per month, and can borrow at 0.45% (per month) over 20 years. How much can I afford to borrow on a fully redeemable mortgage?

Function required: PV(rate, nper, pmt, fv, type)

This formula returns \$366,433.74:

```
=PV(0.45%,240,-2500,0,0)
```

Note that, with mortgages, we always assume payments are in arrears and that the type argument is 0. Also note that the rate of interest (and the payments) are monthly. Therefore, the term of 20 years must be converted to months.

You can check the answer by using the calculated answer to determine the rate on a mortgage of \$366,433.74 over 240 months. The following formula returns 0.45%:

```
=RATE(240, -2500, 366433.74, 0, 0)
```

EXAMPLE 22

I currently owe \$150,000 on a mortgage, and make payments of \$1,900 per month. The current interest rate is 0.45% per month. How long will it take to repay the loan?

Function required: NPER(rate, pmt, pv, fv, type)

The following formula returns 97.76:

```
=NPER(0.45%, -1900, 150000, 0, 0)
```

Because interest and payments are monthly, the formula returns the amortization period in months. This answer, although correct in mathematical terms, has a practical implication. Payments are actually made on exact monthly anniversaries. This calculation implies that the loan somehow gets repaid 0.76 of the way through the 98th month. In reality, you have a choice: make an additional payment at the end of 97 months, or make a reduced level payment after 98 months. These options can be calculated using the FV function.

To calculate the additional payment at the end of 97 months, calculate the amount due using this formula (which returns $-\$1,429.85$):

```
=FV(0.45%, 97, -1900, 150000, 0)
```

Therefore, the final payment after 97 months is $-\$3,329.85$ (that is, the normal payment of $-\$1,900$ plus $-\$1,429.85$).

To calculate the reduced payment after 98 months, use this formula (which returns $+\$463.72$):

```
=FV(0.45%, 98, -1900, 150000, 0)
```

Therefore, the final payment after 98 months is $-\$1,436.28$ (that is, the normal payment of $-\$1,900$ plus $\$463.72$).



A relatively frequent problem arises where the payment is less than the amount of the interest portion on the outstanding balance. In this example, the outstanding loan is \$150,000, and interest in the first month is \$675 ($\$150,000 * 0.45\%$). If the payment is less than this amount, the outstanding

balance will continue to increase, and the loan will extend to infinity (rather than *seem* to last for infinity). If this happens, the NPER function returns the error message #NUM!.

EXAMPLE 23

A consumer credit agreement provides that I borrow \$1,000 and pay \$100 per month in advance for 12 months. What is the rate of interest?

Function required: RATE(**nper**, **pmt**, **pv**, **fv**, **type**, **guess**)

The following formula returns 3.503153%:

```
=RATE(12, -100, 1000, 0, 1)
```

Before you start to think how generous this agreement is, remember that payments are per month. Therefore, the result is the monthly effective rate!

The annual effective equivalent rate is 51.16%, calculated as follows:

```
=((1+0.03503153)^12)-1
```

The annual rate, based on the nominal compounded monthly basis, returns 42.05%, calculated as follows:

```
=3.503153 * 12
```



There is a large difference between the annual effective rate and the equivalent nominal rate compounded monthly. The size of the difference increases with the level of the rates used.

EXAMPLE 24

I borrow \$300,000 on a balloon mortgage over 15 years, with monthly payments on \$100,000. The balance of \$200,000 is due at the end of the term. The rate of interest is 0.4% per month, and payments are made monthly in arrears. What will the payments be?

A common type of mortgage (used to increase the amount that can be borrowed) is the so-called “balloon” mortgage. The loan is divided into two elements: 1) the “payment” element, where payments fully redeem part of the loan by the end of the term, and 2) the “balloon” element. During the loan term, interest only (no principal) is paid on the balloon element. The principal balance is paid as a lump sum at the end of the loan.

The ability to use an *fv* argument in the *PV*, *PMT*, *RATE*, and *NPER* functions make it relatively easy to perform balloon mortgage calculations.

Function required: *PMT*(*rate*, *nper*, *pv*, *fv*, *type*)

The following formula returns $-\$1,580.41$:

```
=PMT(0.4%,180,300000,-200000,0)
```

Note that the total mortgage of $\$300,000$ is used for the *pv* argument.

This calculation can be checked using the calculated payment to determine the *PV*. This formula returns $\$299,999.43$ (the rounding error is caused by using a rounded payment amount):

```
=PV(0.4%,180,-1580.41,-200000,0)
```

The payments on a balloon basis can be compared with payments on a traditional mortgage. This formula returns $\$202,509.64$ (traditional mortgage):

```
=PV(0.4%,180,-1580.41,0,0)
```

And payments for the $\$300,000$ traditional mortgage are $-\$2,341.24$, calculated with this formula:

```
=PMT(0.4%,180,300000,0,0)
```

The previous amortization calculation examples can be modified for balloon mortgages by providing an *fv* argument in the *PV*, *PMT*, *NPER*, and *RATE* functions.

You can also calculate the balloon mortgage element itself with the *FV* function. This is a calculation that requires a careful interpretation of the sign of the result. If the *FV* function returns a positive value, that means that the original mortgage has been overpaid and this amount is now due to the borrower. If it returns a negative amount, this is the amount of the balloon element. A balloon element will exist in cases where the amount of the payments do not fully pay the loan during the mortgage term at the quoted interest rate.

Typically, these calculations are made in two stages. First, calculate the payment on the normal amortization loan (usually in accordance with lender rules). Second, calculate how much “balloon” element an additional payment will allow. Example 25 provides the details.

EXAMPLE 25

If the bank insists on an amortization of $\$200,000$ of a loan, how much extra can I borrow on the balloon mortgage basis if I can afford payments of $\$3,000$ per month? The term of the loan is 10 years, and the current rate is 0.4% per month.

Function required: `PMT(rate, nper, pv, fv, type)`

The first step is to calculate the payment for a \$200,000 normal amortization loan. The following formula returns $-\$2,101.81$:

```
=PMT(0.4%,120,200000,0,0)
```

If payments of \$3,000 are affordable, the additional amount of \$898.19 can be paid as interest on the balloon element (that is, \$3,000 – \$2,101.81). The balloon element can now be calculated because the amount of interest is known. This formula, which represents the balloon element, returns \$224,546.88:

```
=898.19 / 0.4%
```

The calculation can be checked by calculating the payment based on a total mortgage of \$424,546.88 with a balloon element of \$224,546.88. The following formula returns $-\$3,000$:

```
=PMT(0.4%,120,424546.88,-224546.88,0)
```

Converting Interest Rates

The previous examples have been conveniently expressed to allow easy matching of the interest rate with the payment frequency and total term. Often, however, interpreting a financial problem will be more difficult. There are two situations in which interest rate conversions must be made:

- ◆ When you must do calculations involving a frequency of payments or a number of time periods, and the rate that you are required to use does not match the frequency of payments or time period.
- ◆ When you have done calculations involving a frequency of payments or a number of time periods, and you need to express the resulting interest rate in terms of a rate per year or some other period of time.

To create accurate formulas, you will need to understand the principle of equivalence of interest rates. Stated simply, any given interest rate for one period of time is equivalent to another interest rate for a different period of time.

Methods of Quoting Interest Rates

There are three commonly used methods of quoting interest rates:

- ◆ *Nominal rate*: The interest is quoted on an annual basis, along with a compounding frequency per year. For example, the commonly quoted APR of, say, 6% compounded monthly, where 0.5% is charged per month.

- ◆ *Annual effective rate*: A rate of interest in which the given rate represents the percentage earned in one year. For example, with a 10% annual effective rate, \$1,000 earns \$100 interest at the end of a year.
- ◆ *Periodic effective rate*: A rate of interest in which the given rate represents the percentage earned during a period of less than a year. For example, with a rate of 3% per half year, \$300 earns \$9 after six months.

An interest rate quoted using any of these three methods can be converted to any of the other three methods. For example, consider an interest rate of 1% per month on \$100. In the first month, the investment earns \$1 in interest. If the interest credited is not withdrawn, it will be added to the principal, and the subsequent interest will be based on the new balance. A 1% monthly interest rate is equivalent to a 12.6825% per annum interest rate (the effective rate). This is calculated by using the following formula:

$$=(1+0.01)^{12} - 1$$

Another example of a nominal rate is an interest rate quoted as 6% per annum, compounded quarterly. This means that 1.5% (that is, 6% / 4) is paid or received every three months.

Most banks and financial institutions quote interest on a nominal basis compounded monthly. However, when reporting returns from investments or when comparing interest rates, it is common to quote annual effective returns, which makes it easier to compare rates. For example, we know that 12% per annum compounded monthly is more than 12% per annum compounded quarterly – but we don't know (without an intermediate conversion calculation) how *much* more it is.

Converting Interest Rates Using the Financial Functions Add-in

As you will see, 10 different conversions may be required in converting among Nominal, Annual Effective, and Periodic Effective systems.



The companion CD-ROM contains an add-in (named Financial Functions), written by Norman Harker. This add-in provides custom functions (written in VBA) to calculate interest rate conversions. You'll also find a workbook that demonstrates the use of these functions. In addition, these functions are used in many of the examples in this and subsequent chapters. For your convenience, the VBA functions are defined in the example workbooks. Therefore, you do not need to install the add-in to work with the example workbooks.

When using the Financial Functions add-in, you can either enter the function manually, or use Excel's Insert Function dialog box (the functions are located in the Financial category). Table 11-1 lists the 10 interest rate conversion functions contained in the Financial Functions add-in. The table also shows (where applicable) the equivalent Excel formula.

TABLE 11-1 CUSTOM VBA INTEREST RATE CONVERSION FUNCTIONS

Add-in Function	Description	Equivalent Excel Formula
Effx_Nomx (Effx, Freqx)	Converts an Effective rate for a period of less than a year to the equivalent Nominal rate for that frequency.	(none)
Effx_AnnEff (Effx, Freqx)	Converts an Effective rate for a frequency of less than a year to an equivalent Annual Effective rate.	=EFFECT(Effx* Freqx, Freqx)
Effx_Nomy(Effx, Freqx, Freqy)	Converts an Effective rate for a frequency of less than a year to an equivalent Nominal rate for a different frequency.	=NOMINAL(EFFECT(Effx* Freqx, Freqx), Freqy)
Effx_Effy(Effx, Freqx, Freqy)	Converts an Effective rate for a frequency of less than a year to an equivalent Effective rate for a different frequency, which is also less than a year.	=NOMINAL(EFFECT(Effx *Freqx, Freqx, Freqy) Freqx), Freqy)/Freqy
Nomx_Effx (Nomx, Effx)	Converts a Nominal rate to the equivalent Effective rate for the frequency of the Nominal rate.	(none)
Nomx_AnnEff (Nomx, Freqx)	Converts a Nominal rate to the equivalent Annual Effective rate.	=EFFECT(Nomx, Freqx)

Continued

TABLE 11-1 CUSTOM VBA INTEREST RATE CONVERSION FUNCTIONS (Continued)

Add-in Function	Description	Equivalent Excel Formula
Nomx_Nomy(Nomx, Freqx, Freqy)	Converts a Nominal rate for a frequency to an equivalent Nominal rate (for a different frequency).	=NOMINAL(EFFECT(Nomx, Freqx), Freqy)
Nomx_Effy(Nomx, Freqx, Freqy)	Converts a Nominal rate to an equivalent Effective rate for a frequency of less than a year, which is not the frequency of the given Nominal rate.	=NOMINAL(EFFECT(Nomx, Freqx), Freqy)/Freqy
AnnEff_Effx(AnnEff, Freqx)	Converts an Annual Effective rate to an equivalent Effective rate for a frequency of less than a year.	=NOMINAL(AnnEff, Freqx)/Freqx
AnnEff_Nomx(AnnEff, Freqx)	Converts an Annual Effective rate to an equivalent Nominal rate.	=NOMINAL(AnnEff, Freqx)

The function names and arguments may appear confusing at first, but you will soon get the hang of them. The name of each function is made up of three parts:

- ◆ The interest rate you have (Effx, AnnEff, or Nomx). Note that the compounding frequency of the effective and nominal rates are denoted by x .
- ◆ The linking symbol, which is an underscore character (_).
- ◆ The interest rate you want (Effx, Effy, AnnEff, Nomx, or Nomy). Again, compounding frequencies are denoted by x (if it is the same as the frequency of the rate you have), or y (if it is different).

The ordering of arguments is also easy to master:

- ◆ The first argument is always the interest rate you have.
- ◆ The second argument is always the Freqx, which is the frequency of the Effx or Nomx rate. Note that every conversion function uses a Freqx argument, and it is always the second argument.

- ◆ If there is a second known frequency other than x or annual, there is a third argument, Freqy.

Effective Cost of Loans

Lending institutions typically advertise their “headline” rates to make them appear as low as possible. A savvy borrower is able to interpret these rates to determine how much the loan is really costing. The only safe and constant comparison is to look at the effective cost in terms of the annual effective interest rate, or some other common rate such as the annual nominal rate compounded monthly.

This section presents four examples that demonstrate how to calculate the effective cost of loans.



All of the examples in this section are available on the companion CD-ROM. These examples use the custom VBA interest rate conversion functions.

Impact of Fees and Charges upon Effective Interest

In addition to the interest on a mortgage, banks often charge “points,” or set-up fees, and account service fees. These fees add to the effective cost of the loan. But by how much?

EXAMPLE 26

A bank quotes a mortgage rate of 7% nominal compounded monthly, and you are interested in borrowing \$150,000 over 10 years with monthly payments. The bank charges an up-front loan arrangement fee of 2% of the loan, plus an account service fee of \$25 per month. What is the annual effective cost of the loan?

Figure 11-5 shows a worksheet that’s set up to solve this problem. The known information is entered into the Base Data section of the worksheet. Table 11-2 lists the key formulas that perform the calculations. For clarity, the formulas are shown using actual values rather than cell references.

	A	B	C	D	E	F	G	H
1	Effective Cost of Mortgage Loans							
2	A bank quotes a mortgage rate of 7% nominal compounded monthly, and you are interested in borrowing \$150,000 over 10 years with monthly payments. The bank charges an up-front loan arrangement fee of 2% of the loan, plus an account service fee of \$25 per month. What is the annual effective cost of the loan?							
3								
4								
5								
6	Base Data							
7	Borrowing	\$150,000.00						
8	Nominal Rate	7%						
9	Compounding Frequency	12						
10	Term of Years	10						
11	Set up fee % of loan	2%						
12	Account Service Fee	\$25.00						
13								
14								
15	Calculations							
16	Set up fee	\$3,000.00						
17	Effective borrowing	\$147,000.00						
18	Loan term periods	120						
19	Loan rate per period	0.583333%			Check			
20	Loan payment	(\$1,741.63)			\$150,000.00			
21	Loan payment + account fee	(\$1,766.63)						
22	Effective cost of loan	0.648500% per month			\$150,000.00			
23	Annual Effective cost	8.065647%						
24								

Figure 11-5: This worksheet calculates the effective cost of a loan.

TABLE 11-2 FORMULAS USED IN FIGURE 11-5

Cell	Calculation	Formula (Using Actual Values)
B16	Set-up fee	=\$150,000 * 2%
B17	Effective borrowing	=\$150,000 - \$3,000
B18	Loan term periods	=10 * 12
B19	Loan rate period	=Nomx_Effx(7%,12)
B20	Loan payment	=PMT(0.583333%,120,150000,0,0)
B21	Loan payment + fee	=-\$1,741.63-\$25
B22	Effective cost of the loan	=RATE(120, -1766.63,147000,0,0)



Cell B19 uses a custom VBA function.

The payments are based on the loan amount of \$150,000, but the effective cost is based upon the fact that, after deducting the set-up fee, the borrower receives only \$147,000. Similarly, actual payments are higher by the amount of the account service fee.

The impact of these costs varies according to the term: The shorter the term, the greater the impact. If a mortgage is not capable of being transferred to a new house when the borrower moves, the calculation should be based on the likely time that the mortgage will last – usually about seven years.

“Flat” Rate Loans

Many consumer credit agreements use a loan agreement in which a percentage of the loan is added to the loan, and payments are based on the aggregate of the loan amount plus the flat interest divided by the number of payments. You can use Excel’s RATE function to calculate the effective costs of such loans.

EXAMPLE 27

*A consumer finances his car purchase with a flat rate loan of \$15,000 over 18 months. Interest of 10% * 1.5 of this amount is added to the loan and he pays 1/18 of this amount each month in advance for 18 months. What is the effective cost of the loan?*

The easiest way to solve this function is to use the Effx_AnnEff function (a custom VBA function). The following formula returns 3.62%:

```
=Effx_AnnEff(RATE(18, -17250/18, 15000, 0, 1), 12)
```

Note that if the term of such a loan is only 12 months, the rate is slightly more than double the flat rate. Most states and countries have legislated that such loan agreements shall have the annual nominal rate compounded monthly stated clearly in the loan agreement.

Interest-Free Loans

Another interesting calculation is the effective cost of a so-called “interest-free” loan offer. In making these calculations, you need to know the price for which you could get the product elsewhere (without the interest-free package).

EXAMPLE 28

A consumer buys a hi-fi system at a list price of \$3,000 on “interest-free” terms over 12 months, with the payments in advance. He could have purchased an identical system for \$2,500 cash or on normal credit terms. What is the effective cost of this loan?

Again, the Effx_AnnEff VBA function provides the simplest solution. This formula returns 51.16%:

```
=Effx_AnnEff(RATE(12, -(3000/12), 2500, 0, 1), 12)
```

Such calculations are often more difficult when the equivalent cash price is subjective (for example, the used car market).

You can perform similar calculations for other types of agreement, such as “Pay 25% down today, no more to pay for 12 months.” Again, the key is to establish the equivalent cash price, and then compare the calculations with that price, rather than a price that is inflated by the retailer who’s offering the credit.

Most states and countries have consumer credit legislation that governs the quotation of interest rates. In many localities, the only major regulation of interest-free type agreements is that the retailer may not offer the same product at a cash price different from that quoted in the interest-free agreement.

“Annual Payments / 12” Loan Costs

A practice that is rooted in the precalculator days is to calculate payments on an “annual in arrears” basis, and to charge the borrower 1/12 of that amount each month. That calculation was facilitated by preprepared tables of monthly payments per \$1,000 of loan. The practice prevails (especially in UK Building Societies) partly because it produces a lower advertised rate than Nominal or Effective rate regimes.

EXAMPLE 29

A bank offers a mortgage of \$100,000 at a rate of 7% over 10 years, where payments per month are based on 1/12 of the annually calculated payment being paid monthly in arrears. What is the annual effective cost?

The following formula (which uses the `Effx_AnnEff` VBA function) returns .7522% (the per annum effective rate):

```
=Effx_AnnEff(RATE(10*12,PMT(7%,10,100000,0,0)/12,100000,0,0),12)
```

Calculating the Interest and Principal Components

This section discusses four Excel functions that enable you to:

- ◆ Calculate the interest or principal components of a particular payment (the `IPMT` and `PPMT` functions)
- ◆ Calculate cumulate interest or principal components between any two time periods



The examples in this section are available on the companion CD-ROM.

Using the IPMT and PPMT Functions

You may need to know (or simply be curious about) how much of a particular payment constitutes interest, and how much of the payment goes toward the principal. This information might be useful in determining tax effects on interest payments. If you've studied any of the loan amortization examples, you know that the interest element is not constant over the life of a loan. Rather, the interest component decreases, while the principal component increases.



If you've created an amortization schedule, these functions are not particularly useful, because you can simply refer to the schedule. The IPMT and PPMT functions are most useful when you need to determine the interest/principal breakdown of a particular payment.

The syntax for these two functions is as follows (bold arguments are required):

`IPMT(rate, per, nper, pv, fv, type)`

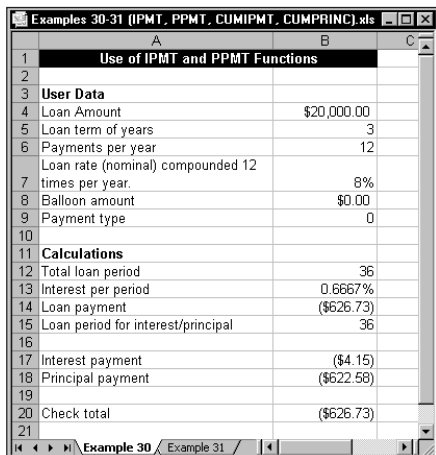
`PPMT(rate, per, nper, pv, fv, type)`

As with all amortization functions, the **rate**, **per**, and **nper** must match in terms of the time period. If the loan term is measured in months, the **rate** argument must be the effective rate per month, and the **per** argument (that is, the period of interest) must be a particular month.

EXAMPLE 30

A consumer obtains a three-year car loan (monthly payments) for \$20,000 at an annual rate of 8%. What are the interest and principal portions for the final loan payment?

Figure 11-6 shows the solution, set up in a worksheet.



	A	B	C
1	Use of IPMT and PPMT Functions		
2			
3	User Data		
4	Loan Amount	\$20,000.00	
5	Loan term of years	3	
6	Payments per year	12	
7	Loan rate (nominal) compounded 12 times per year.	8%	
8	Balloon amount	\$0.00	
9	Payment type	0	
10			
11	Calculations		
12	Total loan period	36	
13	Interest per period	0.6667%	
14	Loan payment	(\$626.73)	
15	Loan period for interest/principal	36	
16			
17	Interest payment	(\$4.15)	
18	Principal payment	(\$622.58)	
19			
20	Check total	(\$626.73)	
21			

Figure 11-6: This worksheet calculates the interest and principal components for any periods of a loan.

Function required: $\text{IPMT}(\text{rate}, \text{per}, \text{nper}, \text{pv}, \text{fv}, \text{type})$

This formula calculates the interest portion of the final payment, and returns $-\$4.15$:

$=\text{IPMT}(8\%/12, 36, 36, 20000, 0, 0)$

The following formula calculates the principal portion of the final payment, and returns $-\$622.58$:

$=\text{PPMT}(8\%/12, 36, 36, 20000, 0, 0)$

By the end of the loan term, practically all of the payment goes toward the principal. To compare this with the first loan period, change the per argument to 1. After doing so, the formulas return $-\$133.33$ (interest) and $-\$493.39$ (principal).



You can check the calculations by using the PMT function (which returns the total payment, interest plus principal). The following formula returns $-\$626.73$, which is the loan payment amount (and the sum of the two previous formulas):

$=\text{PMT}(8\%/12, 36, -20000, 0)$

Using the CUMIPMT and CUMPRINC Functions

The IPMT and PPMT functions can be useful. But, more often, you will need to know the interest or principal component for a group of consecutive periods. In this case, the CUMIPMT and CUMPRINC functions are of greater service. These functions are useful for creating annualized amortization schedules, and for establishing qualifying interest for tax return purposes.

The syntax for these functions is shown here (all arguments are required):

```
CUMIPMT(rate,nper,pv,start_period,end_period,type)
```

```
CUMPRINC(rate,nper,pv,start_period,end_period,type)
```



These functions are available only when the Analysis ToolPak add-in is installed.

EXAMPLE 31

A consumer is borrowing \$250,000 on a mortgage, repayable over 10 years at 5.6% nominal compounded monthly with payments monthly in arrears. What will the payments of interest and principal be in the first year of the loan?

The following formula, for principal payments, returns \$13,512.31:

```
=CUMIPMT(Nomx_Effx(5.6%,12),10*12,250000,1,12,0)
```

The following formula returns \$19,194.42 (total interest payments):

```
=CUMPRINC(Nomx_Effx(5.6%,12),10*12,250000,1,12,0)
```

We can check these answers using the PMT function to calculate the aggregate of the payments. The following formula returns \$32,706.74, which is the aggregate of the preceding results:

```
=PMT(Nomx_Effx(5.6%,12),10*12,250000,0,0)*12
```



These formulas all use the Nomx_Effx custom VBA function.

Matching Different Interest and Payment Frequencies

Previous examples involved nominal interest compounding frequencies that match the frequency of payments. Thus, for example, we might have a quoted nominal rate compounded monthly with payments that are also monthly. As usual, the real world isn't always as cooperative.

EXAMPLE 32

A bank quotes a nominal rate compounded monthly of 6.3%, but allows payments weekly at the equivalent interest rate. If I borrow \$300,000 over 10 years, what will the weekly payments be?

The easy way to resolve such problems is to use the custom `Nomx_Effy` interest conversion function. This formula returns \$777.51:

```
=PMT(Nomx_Effy(6.3%,12,52),10*52,300000,0,0)
```

EXAMPLE 33

We have set up annual accounts, but need to handle a monthly outgoing of \$12,500. Rather than annualize by multiplying by 12, what is the equivalent annual amount using a deposit rate of 7% per annum nominal compounded monthly? The monthly payment is in arrears, and the equivalent amount is to be calculated at the end of each year.

First, calculate the monthly effective rate (using a custom VBA function). The following formula returns 0.58333%:

```
=Nomx_Effx(7%,12)
```

Then, calculate the equivalent annual amount using the `FV` function. This formula returns -\$154,907.29:

```
=-FV(0.58333%,12,-12500,0,0)
```



In this example, the signs can be confusing. Normally we would treat the outgoing as a negative and return a positive future value. However, we will be using the result as an outgoing, so the signs are reversed. This can be done either by using -12,500 as the outgoing, or by reversing the sign of the result by using `-FV` (as in the example).

If the equivalent amount is to be calculated in advance, we would use the same principles and apply the `PV` function.

Limitations of Excel's Financial Functions

Excel's primary financial functions (PV, FV, PMT, RATE, NPER, CUMIPMT, and CUMPRINC) are very useful, but they have two common limitations:

- ◆ They can handle only one level of interest rate.
- ◆ They can handle only one level of payment.

For example, the NPER function cannot handle the variations in payments that arise with credit card calculations. In such calculations, the monthly payment is based upon a reducing outstanding balance, and may also be subject to a minimum amount rule.

The common solution to the problem of varying payments is to create a cash flow schedule and use other financial functions that can handle multiple payments and rates. Examples of the process appear in the next two chapters. Briefly, the functions involved are:

- ◆ FVSCHEDULE, which handles accumulation of a Present Value at different rates and which, when used in a formula, can calculate the present value of a future amount at different rates.
- ◆ IRR, which handles the calculation of a single rate from regular cash flows.
- ◆ NPV, which handles the calculation of the sum of the present values of regular cash flows and which by formula can handle the sum of accumulated values of regular cash flows.
- ◆ MIRR, which is a specialist IRR aimed at avoiding the multiple IRR problem by applying different rates to negative and positive regular cash flows.
- ◆ XIRR, which handles the calculation of a single rate from irregular cash flows.
- ◆ XNPV, which handles the calculation of the sum of the present values of irregular cash flows and which, in a formula, can handle the sum of accumulated values of irregular cash flows.

In a situation that involves only one or two variations, it may be possible to avoid cash flow construction by using formulas nested in or applied to the basic amortization formulas.

Deferred Start to a Series of Regular Payments

In some cases, a series of cash flows may have a deferred start. We can calculate the PV of a regular series of cash flows with a deferred start by using a formula like this:

```
=PV(RATE,NPER,PMT,FV,Type)*(1+RATE)^-DEFER_PER
```

Here, DEFER_PER represents the number of periods for which the first cash flow is deferred.

EXAMPLE 34

I want to borrow money on a deferred payment basis. The deferment period will be one year. Thereafter, the loan will be for 10 years with monthly payments in arrears. The interest rate is 8% per annum effective. The loan is to be secured on a property that I am building, and the bank is prepared to lend, subject to payments not exceeding 75% of the estimated income of \$9,500 per month. How much can I borrow?

The following formula uses the custom AnNeff_Effx function, and returns \$550,422.02:

```
=PV(AnnEfx_Effx(8%,12,10*12,-9500*75%,0,0)*(1+AnnEfx_Effx(8%,12))^-12
```

Valuing a Series of Regular Payments

We can extend the basic principle of discounting successive, but different, levels of payment by chaining the PV functions. For example, if PV1, PV2, and PV3 represent different present values of series of payments for time periods NPER1, NPER2, and NPER3, the discounted value of all series of payments can be found by:

```
PV1 + PV2(1+I)^-NPER1 + PV3(1+I)^-(NPER1+NPER2)
```

EXAMPLE 35

What is the present value of a property yielding an income of \$5,000 per month for four years, rising to \$6,500 per month for the next three years, and rising to \$8,500 per month for the final three years? After 10 years, the property will be worth an estimated \$1,300,000. A discount rate of 10% per annum may be assumed and all payments are in advance.

The following formula returns -\$978,224.54:

```
=PV(AnnEfx_Effx(10%,12),48,5000,0,1) +
PV(AnnEfx_Effx(10%,12),36,6500,0,1)*
(1+AnnEfx_Effx(10%,12))^-48 +
PV(AnnEfx_Effx(10%,12),36,8500,1300000,1)*
(1+AnnEfx_Effx(10%,12))^- (48+36)
```

Note how the final value of \$1,300,000 has been nested in the final PV function.

The same answer could be achieved by “nesting” the successive Present Value inside the preceding function as future values. But remembering that as the PV at that time represents a right to the future income stream, the sign would have to be reversed. The following formula returns \$978,224.54:

```
=PV(AnnEff_Effx(10%,12),48,5000,-
PV(AnnEff_Effx(10%,12),36,6500,-
PV(AnnEff_Effx(10%,12),36,8500,1300000,1),1),1)
```

Of these two approaches, the first formula (using the basic discounting formulas) looks easier as a method; it looks easier to build using the megaformula technique or to break up into three cells that are then added together.

The following formula returns \$200,344.00:

```
=PV(AnnEff_Effx(10%,12),48,5000,0,1)
```

This formula returns \$139,559.07:

```
=PV(AnnEff_Effx(10%,12),36,6500,0,1)*(1+AnnEff_Effx(10%,12))^-48
```

This formula returns \$638,331.47:

```
=PV(AnnEff_Effx(10%,12),36,8500,1300000,1)*(1+AnnEff_Effx(10%,12))^-
(48+36)
```

And the total of the three elements checks at \$978,224.54.

Subject to exceptions involving just one or two changes in the series of payments, the solution will be to set up a cash flow schedule. This will be covered after the next chapter because we first have to outline the basic tools of NPV and IRR.

Summary

This chapter introduced the financial functions and provided the basic concepts of time value of money and equivalent interest rates. The chapter presented a series of examples that used the key financial functions for accumulations, discounting, and loan amortization.

The next chapter presents examples that use Excel for depreciation calculations, and introduces the techniques of calculating net present values (NPV) and internal rates of return (IRR).

Chapter 12

Discounting and Depreciation Financial Functions

IN THIS CHAPTER

- ◆ Using the NPV and IRR functions
- ◆ Understanding the various approaches for cash flows
- ◆ Using cross-checking to verify results
- ◆ Dealing with multiple internal rates of return
- ◆ Understanding the limitations of IRRs and NPVs
- ◆ Extending NPV analysis using more than one rate
- ◆ Using the NPV function to calculate accumulated values
- ◆ Using the depreciation functions

THE NPV (NET PRESENT VALUE) and IRR (Internal Rate of Return) functions are perhaps the most commonly used of the financial analysis tools. This chapter provides many examples of using these functions for various types of financial analysis.

Using the NPV Function

The NPV function returns the sum of any series of regular cash flows, discounted to the present day using a single discount rate. The syntax for Excel's NPV function is shown here (arguments in bold are required):

```
NPV(rate, value1, value2, ...)
```

Cash inflows are represented as positive values, and cash outflows are negative values. The NPV function is subject to the same restrictions that apply to financial functions such as PV, PMT, FV, NPER, and RATE. The only exception is that the payment amounts may vary.

If the discounted negative flows exceed the discounted positive flows, the function will return a negative amount. Similarly, if discounted positive flows exceed discounted negative flows, the NPV function will return a positive amount.

If the NPV is positive, this indicates that at period zero, the investor could pay out up to this additional amount and still achieve the discount rate. If the NPV is negative, then the investor does not get the required discount rate. That rate is often called a *hurdle rate*. The implication of a negative NPV is that the investor is paying out too much. The “right price” requires the addition of the shortfall to the Time 0 cash flow.

The discount rate used must be a single effective rate for the period used for the cash flows. Therefore, if flows are set out monthly, you must use the monthly effective rate.

Definition of NPV

Excel’s NPV function assumes that the first cash flow is received at the *end* of the first period. It is important to understand that this differs from the definition used by most financial calculators, and it is also at odds with the definition used by institutions such as the Appraisal Institute of America (AAI). For example, the AAI defines NPV as the difference between the present value of positive cash flows and the present value of negative cash flows.

If you use Excel’s NPV function without making an adjustment, the result will not adhere to this definition.

Therefore, when using Excel’s NPV function, you will need to take into account the time Point 0 cash flow. For this reason, the procedure to adopt when calculating NPV using Excel is as follows:

- ◆ Treat the number of periods as points in time rather than the time period between points.
- ◆ Always include a Point 0, even if cash flows do not arise until the end of period 1 (Point 1).
- ◆ Use a formula like the one shown here to include the Point 0 cash flow:

`=NPV(Rate,Range)*(1+Rate)`

If you use this procedure, your calculations will adhere to the accepted definitions of NPV, and the results will coincide with those made on your trusty financial calculator. By the way, it’s not that Microsoft got it wrong. The online help clearly states that the first cash flow in the range is assumed to be received at the end of the first period. If you use the previous formula and always have a Time 0 period (even if it is \$0), you will always get the correct answer.

NPV Function Examples

This section contains a number of examples that demonstrate the NPV function.



All of the examples in this section are available on the companion CD-ROM.

EXAMPLE 1

Figure 12-1 shows a worksheet set up to calculate the net present value for a series of cash flows in the range B6:B13.

NPV Example				
Discount Rate:		10%		
---- Cross-Check ----				
Time	Cash Flow	Time	Cash Flow	
0	(\$200,000.00)	0	(\$166,370.86)	
1	\$40,000.00	1	\$40,000.00	
2	\$30,000.00	2	\$30,000.00	
3	\$20,000.00	3	\$20,000.00	
4	\$50,000.00	4	\$50,000.00	
5	\$20,000.00	5	\$20,000.00	
6	\$50,000.00	6	\$50,000.00	
7	\$30,000.00	7	\$30,000.00	
NPV =		(\$33,629.14)	NPV =	\$0.00

Figure 12-1: This worksheet uses the NPV function.

The NPV calculation in cell B15 uses the following formula. This formula returns $-\$33,629.14$:

$$=NPV(B3, B6:B13) * (1+B3)$$

The worksheet in Figure 12-1 also shows a method of cross-checking the NPV calculation. Column E contains a duplicate of the original cash flow, with one exception. The Point 0 cash flow is equal to the original Point 0 cash flow, minus the calculated NPV. In this example, the Point 0 cash flow is $-\$166,370.86$. The cross-check formula in cell E15, shown here, returns $\$0.00$:

$$=NPV(B3, E6:E13) * (1+B3)$$

How does the cross-check work? The discount rate of 10% is used to calculate the surplus or deficit that results from a desired 0% return. In this case, the surplus is calculated as $\$33,629.14$. That surplus is expressed in present value (Point 0) terms. If the surplus is deducted from the Point 0 flow, then there should be no surplus. In other words, if the reversed sign NPV is added to the Time 0 flow, the NPV at the same rate must be 0. If it is 0, this means that the required discount rate was met.

EXAMPLE 2

This example, shown in Figure 12-2, calculates the net present value of a cash flow that begins at the end of the first period.

Time	Cash Flow	Present Value Factor	Present Values
0	\$0.00	1	\$0.00
1	\$40,000.00	0.909090909	\$36,363.64
2	\$30,000.00	0.826446281	\$24,793.39
3	\$20,000.00	0.751314801	\$15,026.30
4	\$50,000.00	0.683013455	\$34,150.67
5	\$20,000.00	0.620921323	\$12,418.43
6	\$50,000.00	0.56447393	\$28,223.70
7	\$30,000.00	0.513158118	\$15,394.74
	NPV=	\$166,370.86	\$166,370.86

Figure 12-2: This worksheet calculates the NPV for a cash flow that begins at the end of the first period.

The NPV calculation, in cell B16, uses the following formula:

$$=NPV(B3, B7 : B14) * (1 + B3)$$

The calculations indicate that we can afford to pay \$166,370.86 for the cash flow, in order to meet a criterion rate of return of 10%.

This example uses another method of cross-checking the result (columns C and D). Column C contains formulas that calculate the present value factor of each cash flow. The formula in cell C7 is:

$$=(1 + B3)^{-A7}$$

The present values are calculated in column D, by multiplying each cash flow by its corresponding present value factor. The formula in cell D7 is:

$$=C7 * B7$$

Column D contains all the present values calculated, and the sum of that column is the sum of the present values. By definition, the sum of the present values (cell D16) should equal the NPV.

EXAMPLE 3

This example (see Figure 12-3) calculates the net present value of a cash flow with an initial (Time 0) positive cash flow.

	A	B	C	D	E
1	Calculating Value of a Cash Flow with Initial Flow				
2					
3	Rate:	10%			
4					
5	Time	Cash Flow			
6	0	\$40,000.00			
7	1	\$40,000.00			
8	2	\$30,000.00			
9	3	\$30,000.00			
10	4	\$20,000.00			
11	5	\$20,000.00			
12	6	\$15,000.00			
13	7	\$15,000.00			
14					
15	NPV=	\$165,939.65			
16					
17					

Figure 12-3: This worksheet calculates the net present value for a cash flow that has an initial flow.

The net present value calculation is in cell B15, which contains the following formula:

$$=NPV(B3,B6:B13)*(1+B3)$$

The calculation indicates that we can pay \$165,939.65 for the right to receive the cash flow, and receive a criterion rate of return of 10%. In this case, however, we pay out \$165,939.65 and have the immediate right to receive the Point 0 cash flow of \$40,000.

This example might seem unusual, but it is common in real estate situations in which rent is paid in advance. In practice, completion rarely coincides with a rent payment date, and the balance of rent previously paid covering the period after the completion date is allowed for in the completion statement.

If we do not know the value, we put 0 in the capital column at period 0, and the NPV represents the value using the required discount rate. If we know the quoting price, we can put that in as a negative at period 0, and the NPV then represents how much more or less we should pay to get the required discount rate.

EXAMPLE 4

This example (see Figure 12-4) calculates a net present value where there is a terminal value, and where cash flows are in advance.

This example is a typical real estate cash flow of rentals payable annually in advance, with an assumed sale after seven years for \$450,000. Pay attention to both ends of the cash flow. In this case, the investor is assumed to receive the first rental of \$30,000 immediately, and will also get the \$40,000 payment made at the end. That might not accord with the facts, and if the last payment is not receivable, you must make it \$0.

	A	B	C	D	E
1	Cash Flows with Terminal Values				
2					
3	Rate:	10%			
4					
5	Time	Income Flow	Capital Flow	Cash Flow	
6	0	\$30,000.00		\$30,000.00	
7	1	\$30,000.00		\$30,000.00	
8	2	\$30,000.00		\$30,000.00	
9	3	\$35,000.00		\$35,000.00	
10	4	\$35,000.00		\$35,000.00	
11	5	\$35,000.00		\$35,000.00	
12	6	\$40,000.00		\$40,000.00	
13	7	\$40,000.00	\$450,000.00	\$490,000.00	
14					
15			NPV=	\$428,026.29	
16					
17					

Figure 12-4: This worksheet demonstrates cash flows with a terminal value.

The NPV calculation in cell D15 is:

$$=NPV(B3, D6:D13) * (1+B3)$$

EXAMPLE 5

This example, shown in Figure 12-5, is similar to Example 4, but it uses a formula (in cell B14) to add the terminal value to the final cash flow.

	A	B	C
1	Cash Flows with Terminal Values		
2			
3	Rate:	10%	
4	Terminal Value	\$ 450,000.00	
5			
6	Time	Income Flow	
7	0	\$30,000.00	
8	1	\$30,000.00	
9	2	\$30,000.00	
10	3	\$35,000.00	
11	4	\$35,000.00	
12	5	\$35,000.00	
13	6	\$40,000.00	
14	7	\$490,000.00	
15			
16	NPV=	\$428,026.29	
17			

Figure 12-5: This worksheet demonstrates cash flows with terminal values.

The formula in cell B16 is:

$$=NPV(B3, B7:B14) * (1+B3)$$

Examples 4 and 5 differ only in the way the data is organized. If you want to separate capital and revenue flows, the approach used in Example 4 is preferable. Separating revenue and capital items (as in Example 4) makes it perfectly clear that the flows are correct without your having to examine the formula.

EXAMPLE 6

This example is a simplistic valuation model that uses initial and terminal flows (see Figure 12-6). It represents a typical investment example in which the aim is to determine if, and by how much, an asking price exceeds a criterion rate of return.

Time	Income Flow	Capital Flow	Cash Flow
0	\$30,000.00	(\$280,000.00)	(\$250,000.00)
1	\$30,000.00		\$30,000.00
2	\$30,000.00		\$30,000.00
3	\$35,000.00		\$35,000.00
4	\$35,000.00		\$35,000.00
5	\$35,000.00		\$35,000.00
6	\$40,000.00		\$40,000.00
7	\$40,000.00	\$450,000.00	\$490,000.00

Figure 12-6: This worksheet demonstrates cash flows with terminal values.

The following formula indicates that, at \$280,000 asking price, the discounted positive cash at the criterion rate of return is \$148,026.29:

$$=NPV(B3, D8:D15) * (1+B3)$$

Put another way, the investor could pay \$428,026.29 and still achieve the criterion rate of return of 10%.

EXAMPLE 7

In the previous examples, the discount rate conveniently matched the time periods used in the cash flow. Often, you'll be faced with a mismatch of rate and time periods. The most common situation occurs when the criterion rate of return is an annual effective rate, and cash flows are monthly or quarterly.

The simplest solution is to use the AnnEff_Effx function (which is also used in some of the examples in Chapter 11). This is a custom VBA function that makes it very easy to convert an interest rate to the monthly effective basis required by a monthly cash flow.



The AnnEff_Effx function is defined in the example workbook on the CD-ROM. The interest rate conversion functions are also available in the Financial Functions add-in (also on the CD-ROM).

Figure 12-7 shows a rental of \$12,000 paid quarterly in advance. It also shows an initial price of \$700,000 and a sale (after three years) for \$900,000. Note that because rent is paid in advance, the purchaser gets a cash adjustment to the price. However, at the end of three years (12 quarters), the same rule applies, and the rent payable for the next quarter is received by the new owner. If we discount at 7% per annum effective, this shows an NPV of \$166,099.72.

Often, rental flows are annualized. This might sound a bit peculiar. However, before the advent of calculators and computers, this was the approach adopted by appraisers who used precalculated tables of annual constants that they applied to the aggregate annual rent. Figure 12-8 shows the same data, but this time we have adopted the approach of assuming that the rent of \$48,000 per annum is paid annually in arrears. Still discounting at 7% per annum effective, we get an NPV of \$160,635.26.

Period	Revenue Flow	Capital Flow	Cash Flow
0	\$12,000.00	(\$700,000.00)	(\$688,000.00)
1	\$12,000.00		\$12,000.00
2	\$12,000.00		\$12,000.00
3	\$12,000.00		\$12,000.00
4	\$12,000.00		\$12,000.00
5	\$12,000.00		\$12,000.00
6	\$12,000.00		\$12,000.00
7	\$12,000.00		\$12,000.00
8	\$12,000.00		\$12,000.00
9	\$12,000.00		\$12,000.00
10	\$12,000.00		\$12,000.00
11	\$12,000.00		\$12,000.00
12		\$900,000.00	\$900,000.00
NPV:			\$166,099.72

Figure 12-7: Calculating the NPV using quarterly cash flows

Period	Revenue Flow	Capital Flow	Cash Flow
0		(\$700,000.00)	(\$700,000.00)
1	\$48,000.00		\$48,000.00
2	\$48,000.00		\$48,000.00
3	\$48,000.00	\$900,000.00	\$948,000.00
NPV:			\$160,635.26
Error due to annualization			\$5,464.46
			3.29%

Figure 12-8: Calculating the NPV by annualizing quarterly cash flows

Using the NPV Function to Calculate Accumulated Amounts

This section presents two examples that use the NPV function to calculate future values or accumulations. These examples take advantage of the fact that:

$$FV = PV * (1 + Rate)$$

EXAMPLE 8

The data for this example is shown in Figure 12-9. The net present value calculation is performed by the formula in cell B15:

$$=NPV(B3,B6:B13)*(1+B3)$$

The future value is calculated using the following formula (in cell B17):

$$=NPV(B3,B6:B13)*(1+B3)*(1+B3)^7$$

Time	Cash Flow	Interest	Cumulative Balance
0	\$100,000.00		\$100,000.00
1	\$40,000.00	\$10,000.00	\$150,000.00
2	\$30,000.00	\$15,000.00	\$195,000.00
3	\$20,000.00	\$19,500.00	\$234,500.00
4	\$50,000.00	\$23,450.00	\$307,950.00
5	\$20,000.00	\$30,795.00	\$368,745.00
6	\$50,000.00	\$35,874.50	\$444,619.50
7	\$30,000.00	\$44,461.95	\$519,081.45

NPV = \$266,370.86

Future Value = \$519,081.45 (Checks with final balance)

Figure 12-9: Calculating FV using the NPV function

The result is verified in column D, which calculates a running balance of the interest. The results of the future value calculation matches the cumulative interest. Interest is calculated using the interest rate multiplied by the previous month's balance. The running balance is the sum of the previous balance, interest, and the current month's cash flow.

It is important to properly sign the cash flows. Then, if the running balance for the previous month is negative, the interest will be negative. Signing the flows properly and using addition is preferable to using the signs in the formulas for interest and balance.

EXAMPLE 9

Chapter 11 covers the use of the PMT function to calculate payments equivalent to a given present value. Similarly, we can use the NPV function, nested in a PMT function, to calculate an equivalent single-level payment to a series of changing payments.

This is a typical problem where we require a time-weighted average single payment to replace a series of varying payments. An example is an agreement in which a schedule of rising rental payments is replaced by a single payment amount. In the example shown in Figure 12-10, the following formula (in cell C27) returns \$10,923.24, which is the payment amount that would substitute for the varying payment amounts in column B:

```
=PMT(C7,C6,-B25,0,C8)
```

The example in this section gives the user flexibility in choice of rate type and frequency of the income flow. Data validation is used to allow the user to select either Effective or Nominal in cell C3. This type of calculation is frequently used to calculate alternatives of fixed and stepped rentals.

Calculating Equivalent Payments with NPV	
Interest Rate Basis:	Effective
Compounding Frequency:	1
Interest Rate:	7%
Cash Flow Frequency:	12
Interest Rate Per Period:	0.56541% Monthly
Payment Type:	1
Time	Cash Flow
0	\$10,000.00
1	\$10,000.00
2	\$10,000.00
3	\$10,500.00
4	\$10,500.00
5	\$10,500.00
6	\$11,250.00
7	\$11,250.00
8	\$11,250.00
9	\$12,000.00
10	\$12,000.00
11	\$12,000.00
12	
NPV=	\$127,100.53
Equivalent Single Payment:	\$10,923.24
Check:	\$127,100.53

Figure 12-10: Calculating equivalent payments with NPV

Using the IRR Function

Excel's IRR function returns the discount rate that makes the net present value of an investment zero. In other words, the IRR function is a special-case NPV, and we will use that feature in designing an automatic cross-check.

The syntax of the IRR function is:

```
IRR(range, guess)
```



The range argument must contain values. Empty cells are not treated as zero. If the range contains empty cells or text, the IRR function does not return an error. Rather, it will return an incorrect result. Thus, if range B1:B40 contains text in cells B11:B20, the IRR will calculate on the basis of 30 consecutive cash flows. This is especially dangerous if the text is misleading: a blank, "-", "nil", "zero", or (worst) "O" (the uppercase "o").

In most cases, the IRR can only be calculated by iteration. The guess argument, if supplied, acts as a "seed" for the iteration process. It has been found that a guess of -0.9 will always produce an answer. Other guesses, such as 0, usually (but not always) produce an answer.

An essential requirement of the IRR function is that there must be both negative and positive income flows: To get a return, there must be an outlay and there must be a payback. There is no essential requirement for the outlay to come first. For a loan analysis using IRR, the loan amount will be positive (and come first) and the repayments that follow will be negative.

The IRR is a very powerful tool, and its uses extend beyond simply calculating the return from an investment. This function can be used in any situation in which we need to calculate a time- and money-weighted average return.

EXAMPLE 10

This example sets up a basic matrix for IRR calculations (see Figure 12-11). This example demonstrates the perennial problem of a cash flow frequency returning an IRR for that frequency. Thus, if cash flows are monthly, the function will return the monthly IRR. The example uses data validation to allow the user to select the type of flow (1, 2, 4, 12, 13, 26, 52, 365, 366). That choice determines the appropriate interest conversion calculation, and also affects the labels in row 5, which contain formulas that reference the text in cell D3.

Monthly Number	Monthly Income Flow	Capital Flow	Net Monthly Flow
0	\$0.00	(\$2,000,000.00)	(\$2,000,000.00)
1	\$50,000.00		\$50,000.00
2	\$50,000.00		\$50,000.00
3	\$50,000.00		\$50,000.00
4	\$50,000.00		\$50,000.00
5	\$50,000.00		\$50,000.00
6	\$50,000.00		\$50,000.00
7	\$50,000.00		\$50,000.00
8	\$50,000.00		\$50,000.00
9	\$50,000.00		\$50,000.00
10	\$50,000.00		\$50,000.00
11	\$50,000.00		\$50,000.00
12	\$50,000.00	\$2,500,000.00	\$2,550,000.00
IRR of Net Quarterly Flow			4.14958% Monthly
IRR p.a.			62.88844% per annum
Check NPV			(\$0.00)

Figure 12-11: This worksheet allows the user to select the time period for the cash flows.

The following formula, in cell D22, is a validity check:

$$=NPV(D20, D6:D18) * (1 + D20)$$

The IRR is the rate at which the discounting of the cash flow produces an NPV of zero. The formula in cell D22 uses the IRR in an NPV function applied to the same cash flow. The NPV discounting at the IRR (per quarter) is \$0.00 – so the calculation checks.

EXAMPLE 11

You may have a need to calculate an average growth rate, or average rate of return. Because of compounding, a simple arithmetic average does not yield the correct answer. Even worse, if the flows are different, an arithmetic average will not take these variations into account.

A solution uses the IRR function to calculate a *geometric* average rate of return. This is simply a calculation that determines the single percentage per period that exactly replaces the varying ones.

Example 11 (see Figure 12-12) shows the IRR function being used to calculate a geometric average return based upon index data (in column B). The calculations of the growth rate for each year are in column C. For example, the formula in cell C5 is:

$$=(B5 - B4) / B4$$

The remaining columns show the geometric average growth rate between different periods. The formulas in Row 10 use the IRR function to calculate the internal rate of return. For example, the formula in cell F10, which returns 5.241%, is:

$$=IRR(F4:F8, -0.9)$$

In other words, the growth rates of 5.21%, 4.86%, and 5.66% are equivalent to a geometric average growth rate of 5.241%.

The IRR calculation takes into account the direction of flow, and places a greater value on the larger flows.

IRR to Calculate Geometric Average Growth							
Year	Index	Growth P.A.	1996-1997	1996-1998	1996-1999	1996-2000	
1996	100.00		-100.00	-100.00	-100.00	-100.00	
1997	105.21	5.21%	105.21	0	0	0	
1998	110.32	4.86%		110.32	0	0	
1999	116.56	5.66%			116.56	0	
2000	119.94	2.90%				119.94	
Average Growth Since 1996			5.210%	5.033%	5.241%	4.650%	

Figure 12-12: Using the IRR function to calculate geometric average growth

EXAMPLE 12

Figure 12-13 shows a worksheet that uses the present value IRR check. This check is based on the definition of IRR: The sum of positive and negative discounted flows is 0.

The net present value is calculated in cell B16:

$$=NPV(D3, B6:B14) * (1+D3)$$

The internal rate of return is calculated in cell B17:

$$=IRR(B6:B14, -0.9)$$

In column C, formulas calculate the present value. They use the IRR (calculated in cell B17) as the discount rate, and use the period number (in column A) for the exponent. For example, the formula in cell C6 is:

$$=B6 * (1 + \$B\$17)^{-A6}$$

The sum of the values in column C is 0.

The formulas in column D use the discount rate (in cell D3) to calculate the present values. For example, the formula in cell D6 is:

$$=B6*(1+D\$3)^{-A6}$$

The sum of the values in column D is equal to the net present value.

For serious applications of NPV and IRR functions, it is an excellent idea to use this type of cross-checking.

Period	Flow	PV IRR Check	PV NPV Check
0	(\$100,000.00)	(\$100,000.00)	(\$100,000.00)
1	\$14,000.00	\$13,570.24	\$12,727.27
2	\$14,000.00	\$13,153.68	\$11,570.25
3	\$14,000.00	\$12,749.90	\$10,518.41
4	\$14,000.00	\$12,358.52	\$9,562.19
5	\$14,000.00	\$11,979.15	\$8,692.90
6	\$15,000.00	\$12,440.62	\$8,467.11
7	\$15,000.00	\$12,058.92	\$7,697.37
8	\$15,000.00	\$11,688.75	\$6,997.61
NPV	(\$23,766.89)		
IRR	3.167%	0	(\$23,766.89)
IRR Error Message		Validity Checked	
NPV Error Message		Validity Checked	

Figure 12-13: Checking IRR and NPV using sum of PV approach

Multiple Rates of IRR and the MIRR Function

In standard cash flows, there is only one sign change: from negative to positive, or from positive to negative. However, there are cash flows in which the sign can change more than once. In those cases, it is possible that more than one IRR can exist.

EXAMPLE 13

Figure 12-14 shows an example that has two IRR calculations, each of which uses a different “seed” value for the guess argument. As you can see, the formula produces different results.

Multiple Internal Rate of Returns					
Seed (1)	11.00000000%				
Seed (2)	-90.00%				
Period	Flow	Interest @ 13.88%	Balance	Interest @ 7.04%	Balance
0	(\$14,375.00)	\$0.00	(\$14,375.00)	\$0.00	(\$14,375.00)
1	\$6,250.00	(\$1,995.53)	(\$10,120.53)	(\$1,012.58)	(\$9,137.58)
2	\$6,250.00	(\$1,404.93)	(\$5,275.46)	(\$643.65)	(\$3,531.23)
3	\$6,250.00	(\$732.34)	\$242.20	(\$248.74)	\$2,470.03
4	\$6,250.00	\$33.62	\$6,525.82	\$173.99	\$8,894.02
5	\$0.00	\$905.91	\$7,431.73	\$526.50	\$9,520.52
6	\$0.00	\$1,031.67	\$8,463.40	\$670.63	\$10,191.14
7	\$0.00	\$1,174.89	\$9,638.29	\$717.87	\$10,909.01
8	\$0.00	\$1,337.98	\$10,976.28	\$768.43	\$11,677.44
9	(\$12,500.00)	\$1,523.72	(\$0.00)	\$822.56	(\$0.00)
Total:		\$1,875.00	Total:	\$1,875.00	
NPV @ IRR					
IRR (1)	13.88197%	(\$0.00)			
IRR (2)	7.0440%	(\$0.00)			

Figure 12-14: A worksheet that demonstrates multiple IRRs

The IRR formula in cell B21 (which returns a result of 13.88%) is:

=IRR(B7:B16,B3)

The IRR formula in cell B22 (which returns a result of 7.04%) is:

=IRR(B7:B16,B4)

So which rate is correct? Unfortunately, *both* are correct. Figure 12-14 shows the interest and running balance calculations for both of these IRR calculations. Both show that the investor can pay and receive either rate of interest, and can secure a (definitional) final balance of \$0. Interestingly, the total interest received (\$1,875) is also the same.

But there's a flaw. This example illustrates a "worst-case scenario" of the practical fallacy of many IRR calculations. NPV and IRR analyses make two assumptions:

- ◆ That we can actually get the assumed (for NPV) or calculated (for IRR) interest on the outstanding balance.
- ◆ That interest does not vary according to whether the running balance is positive or negative.

The first assumption may or may not be correct. It's possible that balances could be reinvested (but in forward projections in times of changing interest rates, this might not be the case). But the real problem is with the second assumption. Banks simply do not charge the same rate for borrowing that they pay for deposits.

EXAMPLE 14

The MIRR function attempts to resolve this multiple rate of return problem. The example in this section demonstrates the use of the MIRR function.

Figure 12-15 shows a worksheet that uses the same data as in Example 13. Rates are provided for borrowing (cell B3) and for deposits (cell B4). These are used as arguments for the MIRR function (cell B19), and the result is 6.1279%, which is different from both of the IRR calculations:

```
=MIRR(B7:B16,B3,B4)
```

Period	Positive Flow	Negative Flow	Revised Flow
0	\$0.00	(\$14,375.00)	(\$20,130.35)
1	\$6,250.00	\$0.00	\$0.00
2	\$6,250.00	\$0.00	\$0.00
3	\$6,250.00	\$0.00	\$0.00
4	\$6,250.00	\$0.00	\$0.00
5	\$0.00	\$0.00	\$0.00
6	\$0.00	\$0.00	\$0.00
7	\$0.00	\$0.00	\$0.00
8	\$0.00	\$0.00	\$0.00
9	\$0.00	\$0.00	\$0.00
9	(\$12,500.00)	\$0.00	\$34,380.83
	NPVs	\$22,162.19	(\$20,130.35)
MIRR:	6.1279%	IRR (Revised Flow):	6.1279%

Figure 12-15: Multiple internal rate of return

The MIRR function works by separating out negative and positive flows, and discounting them at the appropriate rate – the finance rate (for negative flows) and the deposit rate (for positive flows).

We can replicate the MIRR algorithm by setting up a revised flow, which compares the two NPVs (refer to Figure 12-15, columns C:E). The negative flow NPV is placed at Period 0, and the positive flow is expressed as its equivalent future value (by accumulating it at the deposit rate) at the end of the investment term. The IRR of the revised flow is the same as the MIRR of the original (source) flow.

This example reveals that the methodology is suspect. In separating out negative and positive flows, the MIRR implies that interest is charged on flows. Banks, of course, charge interest on balances. An attempt at resolving the problem is shown in the next example.

EXAMPLE 15

The MIRR function uses two rates: one for negative flows, and one for positive flows. In reality, interest rates are charged on *balances* and not on flows. The example in this section applies different rates on negative and positive balances. The interest calculation uses an IF function to determine which rate to use.

When analyzing a project in which interest is paid and received, the end balance must be 0. If it is greater than 0, then we have actually received more than the stated deposit rate. If it is less than 0, then we still owe money and the finance rate has been underestimated. This example assumes a fixed finance rate and calculates the deposit rate needed to secure a 0 final balance.

In the Risk Rate Equivalent IRR method, the finance rate is fixed by the user. The interest received on positive balances is initially “seeded” by the user. Interest on negative balances is charged at the finance rate. Interest on positive balances is at the seed rate. If the seed rate is the exact return, the final balance will be 0. Excel’s Tools → Goal Seek command can be used to determine the exact rate by iterating the interest rate on positive balances to derive a final balance of 0. This is the method used in the example in Figure 12-16.

Period	Interest	Balance	Flow
0	\$0.00	(\$14,375.00)	(\$14,375.00)
1	(\$1,293.75)	(\$9,418.75)	\$4,956.25
2	(\$847.69)	(\$4,016.44)	\$5,402.31
3	(\$361.48)	\$1,872.08	\$5,888.52
4	\$160.61	\$8,282.70	\$6,410.61
5	\$710.61	\$8,993.31	\$710.61
6	\$771.58	\$9,764.88	\$771.58
7	\$837.77	\$10,602.66	\$837.77
8	\$909.65	\$11,512.31	\$909.65
9	\$967.69	(\$0.00)	(\$11,512.31)
		IRR:	0.0000%

Figure 12-16: Accumulating balance approach for multiple IRRs

The revised flow, derived from changes to the running balance, should have an IRR approaching zero. The Risk Rate Equivalent IRR may be compared with a comparator rate such as the Risk Free Rate of Return (traditionally 90-day Treasury bills).

But what does this all mean? It means that if I pay 9% on negative balances, this project gives me 8.579% rate on positive balances. The name “Risk Rate Equivalent IRR” refers to the fact that it determines how the project compares with the return on money invested in a bank or 90-day Treasury bills.

There is no requirement that the finance rate be fixed. A bank might do calculations in the same way, but fix the deposit rate and allow “Goal Seek” to calculate the equivalent lending rate.

Using the FVSCCHEDULE Function

The FVSCCHEDULE function calculates the future value of an initial amount, after applying a series of varying rates over time. Its syntax is:

FVSCCHEDULE(principal,schedule)



The FVSCCHEDULE function is available only when the Analysis ToolPak add-in is installed.

EXAMPLE 16

This example, shown in Figure 12-17, uses the FVSCCHEDULE function to calculate an accumulated amount, together with other formulas that use the base data to calculate an index and the geometric average growth rate.

This worksheet contains details of an index of share prices between 1997 and 2001, with 1997 being assigned an index of 100. This example can answer a question such as: *If we bought \$1,000 of shares in 1997, what would they be worth in 2001, and what has been the average compound growth rate?*

The share value, in cell B13, is \$1,296.81. This is the equivalent of 6.714% compounded on the initial investment of \$1,000.

	A	B	C	D	E	F
1	Use of FVSCCHEDULE Function					
2						
3	Initial Amount:	\$1,000				
4						
5		Growth	Index	Growth p.a From Base	Accumulated Value	
6	1997		100.00		\$1,000.00	
7	1998	6.50%	106.50	6.50%	\$1,065.00	
8	1999	7%	113.96	6.75%	\$1,139.55	
9	2000	8.90%	124.10	7.46%	\$1,240.97	
10	2001	4.50%	129.68	6.71%	\$1,296.81	
11	Simple Average	6.73%				
12						
13	Accumulated Amount	\$1,296.81				
14	Geometric Average Rate	6.714%				
15	Geometric Average Rate	6.714%	(Single formula approach)			
16			Error			
17	Check of Accumulation	\$1,296.81	\$0.00			
18	Check of Average	\$1,296.81	\$0.00			
19						

Figure 12-17: Using the FVSCCHEDULE function

The Accumulated Amount (cell B13) is calculated with the following formula:

```
=FVSCHEDULE(B3,B7:B10)
```



Note that the FVSCHEDULE function does *not* follow the sign convention. It returns a future value with the same sign as the present value. Also, be aware that the growth rates must be the periodic effective rates for the time periods. In the example, the time period is in years, so the growth rates are in annual terms.

The formula in cell B14 calculates the geometric average growth rate:

```
=RATE(4,0,-B3,B17,0)
```

Note that the formula uses a negative sign for the third argument (present value).

You can also calculate the geometric average rate of return by using a single formula (cell B15):

```
=RATE(4,0,-B3,FVSCHEDULE(B3,B7:B10),0)
```

This example also demonstrates a convenient way to calculate an index based on a schedule of growth rates (column C). This topic is covered in detail in the next chapter.

Depreciation Calculations

This section covers depreciation, a critical element for many investment performance analyses. Excel offers five functions to calculate depreciation of an asset over time. Depreciating an asset places a value on the asset at a point in time, based on the original value and its useful life. The function that you choose depends on the type of depreciation method that you use.

Table 12-1 summarizes Excel's depreciation functions and the arguments used by each. For complete details, consult Excel's online help system.

TABLE 12-1 EXCEL'S DEPRECIATION FUNCTIONS

Function	Depreciation Method	Arguments*
SLN	Straight-line. The asset depreciates by the same amount each year of its life.	Cost, Salvage, Life
DB	Declining balance. Computes depreciation at a fixed rate.	Cost, Salvage, Life, Period, [Month]
DDB	Double-declining balance. Computes depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods.	Cost, Salvage, Life, Period, Month, [Factor]
SYD	Sum of the year's digits. Allocates a large depreciation in the earlier years of an asset's life.	Cost, Salvage, Life, Period
VDB	Variable-declining balance. Computes the depreciation of an asset for any period (including partial periods) using the double-declining balance method or some other method you specify.	Cost, Salvage, Life, Start Period, End Period, [Factor], [No Switch]

*Arguments in brackets are optional.

The arguments for the depreciation functions are described as follows:

- ◆ **Cost:** Original cost of the asset.
- ◆ **Salvage:** Salvage cost of the asset after it has fully depreciated.
- ◆ **Life:** Number of periods over which the asset will depreciate.
- ◆ **Period:** Period in the Life for which the calculation is being made.
- ◆ **Month:** Number of months in the first year; if omitted, Excel uses 12.
- ◆ **Factor:** Rate at which the balance declines; if omitted, it is assumed to be 2 (that is, double-declining).
- ◆ **Rate:** Interest rate per period. If you make payments monthly, for example, you must divide the annual interest rate by 12.
- ◆ **No-switch:** True or False. Specifies whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation.

Figure 12-18 shows depreciation calculations using the SLN, DB, DDB, and SYD functions. The asset's original cost, \$10,000, is assumed to have a useful life of 10 years, with a salvage value of \$1,000. The range labeled Depreciation Amount shows the annual depreciation of the asset. The range labeled Value of Asset shows the asset's depreciated value over its life.

depreciation.xls					
	A	B	C	D	E
1	Asset:	Office Furniture			
2	Original Cost:	\$10,000			
3	Life (years):	10			
4	Salvage Value:	\$1,000			
5					
6	Depreciation Amount				
7	Year	SLN	DB	DDB	SYD
8	1	\$900.00	\$2,060.00	\$2,000.00	\$1,636.36
9	2	\$900.00	\$1,636.64	\$1,600.00	\$1,472.73
10	3	\$900.00	\$1,298.70	\$1,280.00	\$1,309.09
11	4	\$900.00	\$1,031.17	\$1,024.00	\$1,145.45
12	5	\$900.00	\$818.75	\$819.20	\$981.82
13	6	\$900.00	\$650.08	\$655.36	\$818.18
14	7	\$900.00	\$516.17	\$524.29	\$654.55
15	8	\$900.00	\$409.84	\$419.43	\$490.91
16	9	\$900.00	\$325.41	\$335.54	\$327.27
17	10	\$900.00	\$258.38	\$268.44	\$163.64
18					
19					
20	Value of Asset				
21	Year	SLN	DB	DDB	SYD
22	0	\$10,000.00	\$10,000.00	\$10,000.00	\$10,000.00
23	1	\$9,100.00	\$7,940.00	\$8,000.00	\$8,363.64
24	2	\$8,200.00	\$6,304.36	\$6,400.00	\$6,890.91
25	3	\$7,300.00	\$5,005.66	\$5,120.00	\$5,581.82
26	4	\$6,400.00	\$3,974.50	\$4,096.00	\$4,436.36
27	5	\$5,500.00	\$3,155.75	\$3,276.80	\$3,454.55
28	6	\$4,600.00	\$2,505.67	\$2,621.44	\$2,636.36
29	7	\$3,700.00	\$1,989.50	\$2,097.15	\$1,981.82
30	8	\$2,800.00	\$1,579.66	\$1,677.72	\$1,490.91
31	9	\$1,900.00	\$1,254.25	\$1,342.18	\$1,163.64
32	10	\$1,000.00	\$995.88	\$1,073.74	\$1,000.00
33					

Figure 12-18: A comparison of four depreciation functions



The companion CD-ROM contains the workbook shown in Figure 12-18.

Figure 12-19 shows a chart that graphs the asset's value. As you can see, the SLN function produces a straight line; the other functions produce curved lines because the depreciation is greater in the earlier years of the asset's life.

The VDB function is useful if you need to calculate depreciation for multiple periods (for example, years 2 and 3). Figure 12-20 shows a worksheet set up to calculate depreciation using the VDB function. The formula in cell B12 is:

=VDB(B2,B4,B3,B6,B7,B8,B9)

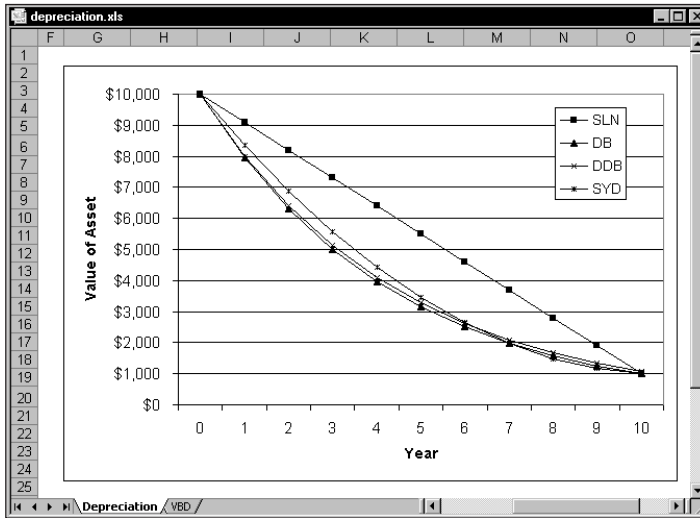


Figure 12-19: This chart shows an asset's value over time, using four depreciation functions.

	A	B	C
1	Asset:	Office Furniture	
2	Original Cost:	\$10,000	
3	Life (years):	10	
4	Salvage Value:	\$1,000	
5			
6	Starting Period:	0	
7	Ending Period:	3	
8	Factor:	2	
9	No-Switch:	TRUE	
10			
11			
12	Depreciation -->	\$4,880.00	
13			
14			
15			

Figure 12-20: Using the VBD function to calculate depreciation for multiple periods

The formula displays the depreciation for the first three years of an asset (starting period of 0 and ending period of 3).

Summary

In this chapter, we have completed assembling the basic tools required for some quite complex financial analyses.

The next chapter applies these tools and illustrates a number of very useful formulas and construction techniques.

Chapter 13

Advanced Uses of Financial Functions and Formulas

IN THIS CHAPTER

- ◆ Setting up dynamic schedules
- ◆ Creating amortization schedules
- ◆ Creating data tables
- ◆ Creating accumulation schedules
- ◆ Working with discounted cash flow
- ◆ Understanding credit card repayment calculations
- ◆ Analyzing investment performance
- ◆ Creating indices

THIS CHAPTER MAKES USE OF MUCH of the information contained in the two previous chapters. It contains useful examples of a wide variety of financial calculations.

Creating Dynamic Financial Schedules

A financial schedule is a detailed listing of cash flows. Typically, each row represents a time period (such as a month), and the information for that time period is displayed in the columns. As you are well aware, electronic spreadsheets are ideal for creating financial schedules.

The most useful type of financial schedule is a *dynamic* schedule, which uses input cells (that represent variables) to adjust itself. The best dynamic schedule is one that allows maximum flexibility, and allows the user to change any of the key variables used in the calculations. Obviously, you'll want to avoid hard-coding values within formulas. Rather, the values should be stored in cells, which are referenced by the formulas.

This task becomes a bit tricky when the schedule involves variable time periods—for example, if the user inputs the term of the loan. In such a case, the number of rows in the schedule will be variable.

Most dynamic schedules contain at least four basic sections:

- ◆ User inputs
- ◆ Intermediate calculations
- ◆ Summary output
- ◆ The schedule itself

These sections can be stored on a single worksheet, or in multiple worksheets. The remainder of this chapter presents examples of some typical financial schedules.

Creating Amortization Schedules

In its simplest form, an amortization schedule tracks the payments (including interest and principal components) and the loan balance for a particular loan. This section presents several examples of amortization schedules.

A Simple Amortization Schedule

This example uses a simple loan to demonstrate the basic concepts involved in creating a dynamic schedule. Refer to the worksheet in Figure 13-1.



This example is available on the companion CD-ROM.

USER INPUT SECTION

The user input area is the range B4:B9. In this example, cell B6 contains a simple data validation list, which allows either of two strings: Nominal or Effective. Cell C7 contains a formula that uses a custom VBA function:

```
=FreqName(B7)
```

This formula returns a text string that describes the compounding frequency entered into cell B7. All of the other cells in the user input section contain values.

Period	Payment	Interest	Principal	Balance
0				\$150,000.00
1	(\$20,476.47)	(\$3,000.00)	(\$17,476.47)	\$132,523.53
2	(\$20,476.47)	(\$2,660.47)	(\$17,826.00)	\$114,697.53
3	(\$20,476.47)	(\$2,293.95)	(\$18,182.52)	\$96,515.01
4	(\$20,476.47)	(\$1,930.30)	(\$18,546.17)	\$77,968.84
5	(\$20,476.47)	(\$1,559.38)	(\$18,917.09)	\$59,051.75
6	(\$20,476.47)	(\$1,181.03)	(\$19,295.43)	\$39,756.31
7	(\$20,476.47)	(\$795.13)	(\$19,681.34)	\$20,074.97
8	(\$20,476.47)	(\$401.50)	(\$20,074.97)	(\$0.00)
9	\$0.00	\$0.00	\$0.00	\$0.00
10	\$0.00	\$0.00	\$0.00	\$0.00

Figure 13-1: A simple amortization schedule

INTERMEDIATE CALCULATIONS

In this example, formulas perform intermediate calculations in the range B12:B14. Cell B12 uses custom VBA functions to calculate the periodic interest rate, using cells from the input section:

```
=IF(B5="Nominal",Nomx_Effy(B4,B6,B7),Effx_Effy(B4,B6,B7))
```

Cell B13 contains a simple formula that calculates the number of holding periods (that is, the number of rows in the schedule):

```
=B9*B8
```

Cell B14 uses the PMT function to calculate the periodic payment:

```
=PMT(B12,B13,B4,0,0)
```

SUMMARY INFORMATION

In this example, the summary information section contains only one formula, in cell B17. This formula calculates the total interest paid:

```
=SUM(C21:C381)
```



Placing the summary information above the schedule itself eliminates the need to scroll to the end of the worksheet.

THE SCHEDULE

The amortization schedule begins in row 20, which contains descriptive labels. The standard approach is to hard code the “zero” period and the first time period, and use formulas to derive the subsequent time periods. In this example, cells A21 and A22 contain hard-coded values. Cells A23 downward, however, contain formulas. The formula in cell A23 is:

```
=IF(A22<=$B$13,IF(A22=0,0,A22+1),0)
```

This formula is copied down to cell A381. The formula increments the time period number by 1, until the total number of time period is reached. When the period exceeds the total number of periods, the formula returns 0. In this example, this occurs in cell A30.

Each formula cell (columns B:F) in the schedule refers to the time period in its corresponding row. If the time period is not 0, the formula returns a result. Otherwise, it returns 0.

The formula in cell B22, which displays the periodic Payment, is:

```
=IF(A22=0,0,$B$14)
```

Interest is calculated by multiplying the preceding Balance by the interest rate per period. Principal repaid is equal to the Payment amount less the Interest amount. Finally, the new Balance is calculated by adding the (negative) principal repayment to the preceding balance. The Interest formula in cell C22 is:

```
=IF(A22=0,0,-E21*$B$12)
```

The Principal is calculated using the following formula (cell D22):

```
=IF(A22=0,0,B22-C22)
```

The Balance (cell E2) is calculated using this formula:

```
=IF(A22=0,0,E21+D22)
```

These formulas are copied down as far as the reasonable maximum for the term allows (in this example, they are copied down to row 381). Note that these formulas return a nonzero value only if column A contains a nonzero period.



To hide the 0's in the unused rows, you can use the Tools → Options command, select the View tab, and remove the check from Zero values. Another option is to use an empty string ("") in place of the 0 in the formulas. Yet another option is to use AutoFiltering to hide the unused rows.

Loan amortization schedules are self-checking. If everything is set up correctly, the final balance at the end of the term is 0 (or very close to 0, given rounding errors). Another check is to add the Principal components. The sum of these values should equal the original loan amount.

A Detailed Amortization Schedule

The example in this section builds on the previous example. Figure 13-2 shows a more detailed loan amortization schedule that examines the effects of loan set-up costs, account fees, and tax relief on interest.

Detailed Loan Amortization Schedule								
User Input								
Principal			\$3,000,000					
Loan Term (years)		2		Loan term * Frequency not to exceed 360				
Nominal Interest Rate p.a.			15.00%					
Payment Frequency per year		4		Loan term * Frequency not to exceed 360				
Setup Costs (% of borrowing)			2.00%					
Account Fees (% of borrowing)			0.05%					
Tax rate (%)			49.25%					
Calculated Data				Summary Information				
Total loan periods			8		Total Payments (includes fees)			\$3,599,961
Effective Borrowing			\$2,940,000		Total Interest Paid			\$527,961
Loan Payment (before account fees)			(\$440,995)		Total Fees Paid			\$72,000
Loan Payment (after account fees)			(\$442,495)		Principal Repaid (Check)			\$3,000,000
Effective Equivalent of Nominal Rate			15.8650%		Payments less principal			\$599,961
Effective Loan Cost Before Tax Relief			18.4411%		Total Tax Relief			\$295,481
Effective Loan Cost After Tax Relief			9.0004%		Payments net of tax relief and principal repaid			\$304,480
Amortization Schedule								
Loan Period	Loan Pmt Before Fees	Loan Pmt After Fees	Interest Component	Principal Component	Account Balance	Tax Relief	Cash Flow Before Tax	Cash Flow After Tax
0		(\$60,000)			\$3,000,000	\$29,550	\$2,940,000	\$2,969,550
1	(\$440,995)	(\$442,495)	\$112,500	\$328,495	\$2,671,505	\$56,145	(\$442,495)	(\$386,350)
2	(\$440,995)	(\$442,495)	\$100,181	\$340,814	\$2,330,691	\$50,078	(\$442,495)	(\$392,417)
3	(\$440,995)	(\$442,495)	\$87,401	\$353,594	\$1,977,097	\$43,784	(\$442,495)	(\$398,711)
4	(\$440,995)	(\$442,495)	\$74,141	\$366,854	\$1,610,243	\$37,253	(\$442,495)	(\$405,242)
5	(\$440,995)	(\$442,495)	\$60,384	\$380,611	\$1,229,632	\$30,478	(\$442,495)	(\$412,017)
6	(\$440,995)	(\$442,495)	\$46,111	\$394,884	\$834,748	\$23,449	(\$442,495)	(\$419,047)
7	(\$440,995)	(\$442,495)	\$31,303	\$409,692	\$425,056	\$16,155	(\$442,495)	(\$426,340)
8	(\$440,995)	(\$442,495)	\$15,940	\$425,056	\$0	\$8,589	(\$442,495)	(\$433,906)
0	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0

Figure 13-2: A detailed amortization schedule



This example is available on the companion CD-ROM.

As you examine this example, keep the following points in mind:

- ◆ Effective borrowing is defined in Chapter 11 as the amount borrowed, less the amount of set-up fees. Loan repayments are based on the loan amount, but the effective cost is based on the effective borrowing.
- ◆ The payments are calculated using the PMT function, but actual payments are adjusted by adding the amount of the account service fees.
- ◆ In this example, tax relief is allowed only on the interest component of the loan. Tax laws may vary.
- ◆ The calculation of the effective equivalent of the nominal rate uses Excel's EFFECT function.
- ◆ The Effective Loan Cost Before Tax Relief (cell D17) is calculated by using the IRR function on column H. The Effective Loan Cost After Tax Relief (cell D18) is calculated by using the IRR function on column I.
- ◆ The schedule has the capacity for a total of 360 loan periods and an error message will appear if this number is exceeded.
- ◆ The schedule is self-checking. The end balance is zero, and the total principal repaid equals the original loan amount.

A Variable Loan Rate Amortization Schedule

The amortization schedules presented in this chapter have all been based on fixed-rate loans. Many loans, however, are *variable-rate* loans and make use of varying interest rates throughout the term. Typically, these loans are structured such that payments vary along with the rate.

Figure 13-3 shows a dynamic amortization schedule for a variable-rate loan. The user can enter loan rates in column B. The main problem, of course, is that the loan rates are often based on an index, so the rates are not known in advance. In such a case, this type of amortization schedule is based on *assumptions* about the future rates.

The major change, relative to the previous example, is the use of a relatively simple formula for calculating the loan repayments before fees (column C).

Variable Rate Loan Amortization Schedule										
User Input										
3	Principal				\$3,000,000					
4	Loan Term (years)				2	Loan term * Frequency not to exceed 360				
5	Repayment Frequency per year				4	Loan term * Frequency not to exceed 360				
6	Setup costs (% of borrowing)				2.00%					
7	Account Fees (% of borrowing)				0.05%					
8	Tax rate (%)				49.25%					
Calculated Data						Summary Data				
12	Total loan periods				8	Total Repayments (includes fees)				\$3,321,416
13	Effective Borrowing				\$2,940,000	Total Interest Paid				\$249,416
14	Effective Loan Cost Before Tax Relief				9.7899%	Total Fees Paid				\$72,000
15	Effective Loan Cost After Tax Relief				4.8515%	Principal Repaid (Check)				\$3,000,000
16						Payments less principal				\$321,416
17						Total Tax Relief				\$158,298
18						Payments net of tax relief & principal				\$163,119
Amortization Schedule										
	Loan Period	Interest Rate	Loan Pmt Before Fees	Loan Pmt After Fees	Interest Component	Principal Component	Account Balance	Tax Relief	Cash Flow Before Tax	Cash Flow After Tax
22	0			(\$60,000)			\$3,000,000	\$29,550	\$2,940,000	\$2,969,550
23	1	7.00%	(\$405,129)	(\$406,629)	\$52,500	\$352,629	\$2,647,371	\$26,595	(\$406,629)	(\$380,034)
24	2	7.00%	(\$405,129)	(\$406,629)	\$46,329	\$358,800	\$2,288,571	\$23,556	(\$406,629)	(\$383,073)
25	3	7.00%	(\$405,129)	(\$406,629)	\$40,050	\$365,079	\$1,923,493	\$20,463	(\$406,629)	(\$386,165)
26	4	7.00%	(\$405,129)	(\$406,629)	\$33,661	\$371,468	\$1,552,025	\$17,317	(\$406,629)	(\$389,312)
27	5	8.00%	(\$407,599)	(\$409,099)	\$31,041	\$376,558	\$1,175,467	\$16,026	(\$409,099)	(\$393,072)
28	6	8.00%	(\$407,599)	(\$409,099)	\$23,509	\$384,089	\$791,378	\$12,317	(\$409,099)	(\$396,782)
29	7	7.50%	(\$406,852)	(\$408,352)	\$14,838	\$392,014	\$399,364	\$8,047	(\$408,352)	(\$400,305)
30	8	7.50%	(\$406,852)	(\$408,352)	\$7,488	\$399,364	\$0	\$4,427	(\$408,352)	(\$403,925)
31	0	7.50%	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0

Figure 13-3: A variable-rate loan amortization schedule



This example is available on the companion CD ROM.

Loan payments (before fees) for each period are based upon a PMT function constructed as follows: The loan rate is based on the rate for the period (in column B), divided by the loan repayment frequency. The loan term for each period is calculated as the Maximum loan term less the period number of the previous row. Thus, the loan term recalculates for every repayment in the column. The borrowing (PV) is the balance outstanding for the previous period. Again, we are recalculating the borrowing for every repayment. The resulting formula for repayments for the first period (cell C23) is:

```
=IF(A23=0,0,PMT(B23/$E$5,MAX(A22:A382)-A22,G22,0,0))
```

Cell B23 contains the interest rate for the period, and cell E5 contains the compounding frequency.



This schedule works because, at any time during the loan, the repayments calculated must exactly pay off the outstanding balance before the end of the term. If the borrower chose instead to vary the term of the loan rather than vary repayments, this approach would need to be varied by adjusting the term column with an IF function using the NPER function.

Summarizing Loan Options Using a Data Table

Excel's Data → Table command is a handy tool for summarizing various loan options. This section describes how to create one-way and two-way data tables.



A workbook that demonstrates one- and two-way data tables is available on the companion CD-ROM.

Creating a One-Way Data Table

A one-way data table shows the results of any number of calculations for different values of a single input cell.

Figure 13-4 shows a one-way data table (in B10:I13) that displays three calculations (payment amount, total payments, and total interest) for a loan, using seven interest rates ranging from 7.00% to 8.50%. In this example, the input cell is cell B2.

	A	B	C	D	E	F	G	H	I
1	Loan Amount:	\$10,000.00							
2	Annual Interest Rate:	7.25%							
3	Pmt. Period (months):	1							
4	Number of Periods:	36							
5									
6	Payment Amount:	\$309.92							
7	Total Payments:	\$11,156.95							
8	Total Interest:	\$1,156.95							
9									
10			7.00%	7.25%	7.50%	7.75%	8.00%	8.25%	8.50%
11	Payment Amount:	\$309.92	\$308.77	\$309.92	\$311.06	\$312.21	\$313.36	\$314.52	\$315.68
12	Total Payments:	\$11,156.95	\$11,115.75	\$11,156.95	\$11,198.24	\$11,239.62	\$11,281.09	\$11,322.66	\$11,364.31
13	Total Interest:	\$1,156.95	\$1,115.75	\$1,156.95	\$1,198.24	\$1,239.62	\$1,281.09	\$1,322.66	\$1,364.31
14									
15									

Figure 13-4: Using a one-way data table to display three loan calculations for various interest rates

To create this one-way data table, follow these steps:

1. Enter the formulas that return the results for use in the data table. In this example, the formulas are in B6:B8.
2. Enter various values for a single input cell in successive columns. In this example, the input value is interest rate, and the values for various interest rates appear in C10:I10.
3. Create a reference to the formula cells in the column to the left of the input values. In this example, the range B11:B13 contains simple formulas that reference other cells. For example, B11 contains the following formula:

```
=B6
```
4. Select the rectangular range that contains the entries from the previous steps. In this example, select B10:I13.
5. Select the Data → Table command. Excel displays the Table dialog box shown in Figure 13-5.

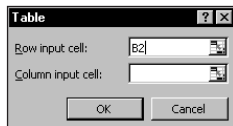


Figure 13-5: Excel's Table dialog box

6. For the Row input cell field, specify the cell reference that corresponds to the variable in your Data Table column header row. In this example, the Row input cell is B2.
7. Leave the Column input cell field empty.
8. Click OK. Excel inserts an array formula that uses the TABLE function with a single argument.
9. If you like, you can format the data table. For example, you might want to apply shading to the row and column headers.

Note that the array formula is not entered into the entire range that you selected in Step 4. The first column and first row of your selection are not changed.



When you create a data table, the leftmost column of the data table (the column that contains the references entered in Step 3) contains the calculated values for the input cell. In this example, those values are repeated in column D. You might want to “hide” the values in column B by making the font color the same color as the background.

Creating a Two-Way Data Table

A two-way data table shows the results of a single calculation for different values of two input cells. Figure 13-6 shows a two-way data table (in B10:I16) that displays a calculation (payment amount) for a loan, using seven interest rates and six loan amounts.

	A	B	C	D	E	F	G	H	I
1	Loan Amount:	\$10,000.00							
2	Annual Interest Rate:	7.25%							
3	Pmt. Period (months):	1							
4	Number of Periods:	36							
5									
6	Payment Amount:	\$309.92							
7	Total Payments:	\$11,156.95							
8	Total Interest:	\$1,156.95							
9									
10			<i>Interest Rate</i>						
10		\$309.92	7.00%	7.25%	7.50%	7.75%	8.00%	8.25%	8.50%
11		\$9,000.00	\$277.89	\$278.92	\$279.96	\$280.99	\$282.03	\$283.07	\$284.11
12		\$9,500.00	\$293.33	\$294.42	\$295.51	\$296.60	\$297.70	\$298.79	\$299.89
13	Loan Amount	\$10,000.00	\$308.77	\$309.92	\$311.06	\$312.21	\$313.36	\$314.52	\$315.68
14		\$10,500.00	\$324.21	\$325.41	\$326.62	\$327.82	\$329.03	\$330.24	\$331.46
15		\$11,000.00	\$339.65	\$340.91	\$342.17	\$343.43	\$344.70	\$345.97	\$347.24
16		\$11,500.00	\$355.09	\$356.40	\$357.72	\$359.04	\$360.37	\$361.70	\$363.03

Figure 13-6: Using a two-way data table to display payment amounts for various loan amounts and interest rates

To create this two-way data table, follow these steps:

1. Enter a formula that returns the results that will be used in the data table. In this example, the formula is in cell B6. The formulas in B7:B8 are not used.
2. Enter various values for the first input in successive columns. In this example, the first input value is interest rate, and the values for various interest rates appear in C10:I10.
3. Enter various values for the second input cell in successive rows, to the left and below the input values for the first input. In this example, the second input value is loan amount, and the values for various loan amounts are in B11:B16.

4. Create a reference to the formula that will be calculated in the table. This reference goes in the upper-left corner of the data table range. In this example, cell B10 contains the following formula:

```
=B6
```
5. Select the rectangular range that contains the entries from the previous steps. In this example, select B10:I16.
6. Select the Data → Table command. Excel displays the Table dialog box.
7. For the Row input cell field, specify the cell reference that corresponds to the first input cell. In this example, the Row input cell is B2.
8. For the Column input cell field, specify the cell reference that corresponds to the second input cell. In this example, the Row input cell is B1.
9. Click OK. Excel inserts an array formula that uses the TABLE function with two arguments.

After you create the two-way data table, you can change the calculated cell by changing the cell reference in the upper left cell of the data table. In this example, you can change the formula in cell B10 to =B8 so the data table displays total interest rather than payment amounts.



If you find that using data tables slows down the calculation of your workbook, select Tools → Options. In the Options dialog box, click the Calculation tab and change the calculation mode to Automatic except tables.

Accumulation Schedules

An accumulation schedule is similar to an amortization schedule, but the cash flows can be both incoming and outgoing. You might use an accumulation schedule to calculate details for an account with varying levels of regular contributions and withdrawals, and occasional lump sum contributions and withdrawals. Figure 13-7 shows an example of such a schedule.



This example is available on the companion CD-ROM.

Period	Regular Payments	Lump Sum Payments	Regular Withdrawals	Lump Sum Withdrawals	Interest Rate	Interest	Balance
0	\$250.00	\$5,000.00	(\$200.00)				\$5,050.00
1	\$250.00		(\$200.00)		7.00%	\$28.55	\$5,128.55
2	\$250.00		(\$200.00)		7.00%	\$29.00	\$5,207.55
3	\$250.00		(\$200.00)	(\$3,000.00)	7.00%	\$29.44	\$2,287.00
4	\$250.00		(\$200.00)		6.50%	\$12.03	\$2,349.03
5	\$250.00		(\$200.00)		6.50%	\$12.36	\$2,411.39
6	\$250.00	\$10,000.00	(\$200.00)		6.50%	\$12.69	\$12,474.08
7	\$250.00		(\$200.00)		6.50%	\$65.63	\$12,589.71
8	\$250.00		(\$200.00)		6.50%	\$66.24	\$12,705.95
9	\$250.00		(\$200.00)	(\$3,000.00)	6.00%	\$61.85	\$9,817.80
10	\$250.00		(\$200.00)		6.00%	\$47.79	\$9,915.59
11	\$250.00		(\$200.00)		6.00%	\$48.26	\$10,013.85
12	\$250.00		(\$200.00)		6.00%	\$48.74	\$10,112.60

Figure 13-7: An accumulation schedule

The most complicated part of this schedule deals with the rate of interest and interest calculation. The user inputs the interest rate in annual terms in column F and selects the type (cell C3), compounding frequency of the rate (cell C4), and the schedule frequency (cell C5).

The interest calculation depends on the choice of rate and follows the standard approach developed in Chapter 12 using custom VBA functions. The formula in cell G10, for example, is:

```
=IF($C$3="Nominal",Nomx_Effy(F10,$C$4,$C$5),Effx_Effy(F10,$C$4,$C$5))*H9
```

in this formula:

- ◆ Cell C3 is an absolute reference to the interest rate type (Nominal or Effective).
- ◆ Cell F10 is the rate for the current period.
- ◆ Cell C4 is the absolute reference to the compounding frequency of the rate.
- ◆ Cell C5 is an absolute reference to the frequency of the schedule.
- ◆ Cell H9 is the balance for the preceding period. The balance is the sum of the preceding balance, payments, and withdrawals.



This example is available on the companion CD-ROM.

We have covered only 12 periods here, but the schedule can be continued for as long as required.

Discounted Cash Flow Schedules

Discounted Cash Flow (DCF) is an investment analysis technique that uses either NPV or IRR calculations on a schedule of positive and negative cash flows. The NPV technique calculates the amount by which the discounted positive and negative flows vary. The IRR technique shows the amount of return per period of cash flow.

DCF schedules can be very extensive, and include complex calculations of the main elements. However, the basics are relatively simple and require little additional work as far as the formulas and functions are concerned.

Figure 13-8 shows a basic DCF schedule, with all of the essential elements, including:

- ◆ A flow frequency (cell C3), which is vital in terms of interpreting the IRR. The IRR (cell C7) is reported as a rate per period of flow and is used to calculate an NPV.
- ◆ An Initial Value (cell C4), which is treated as an outgoing flow and is negative.
- ◆ A Terminal Value (cell C5), which is treated as a receipt and is positive.
- ◆ A Discount Rate (cell C10) for calculating the NPV and a basis for quoting that discount rate.
- ◆ The schedule itself, which details Capital, Income, and Outgoings. These are summed to yield the Cash Flow per period.



This example is available on the companion CD-ROM.

Period	Capital Flows	Income Flows	Outgoings	Cash Flow
0	(\$1,000,000.00)	\$34,000.00	(\$8,000.00)	(\$974,000.00)
1		\$34,000.00	(\$8,000.00)	\$26,000.00
2		\$34,000.00	(\$8,000.00)	\$26,000.00
3		\$34,000.00	(\$8,000.00)	\$26,000.00
4		\$38,000.00	(\$10,000.00)	\$28,000.00
5		\$38,000.00	(\$10,000.00)	\$28,000.00
6		\$38,000.00	(\$10,000.00)	\$28,000.00
7		\$38,000.00	(\$10,000.00)	\$28,000.00
8		\$43,000.00	(\$12,500.00)	\$30,500.00
9		\$43,000.00	(\$12,500.00)	\$30,500.00
10		\$43,000.00	(\$12,500.00)	\$30,500.00
11		\$43,000.00	(\$12,500.00)	\$30,500.00
12	\$1,200,000.00			\$1,200,000.00

Figure 13-8: A discounted cash flow schedule

In this example, the flow frequency is quarterly. Therefore, the IRR is a quarterly effective IRR. To convert to the annual effective equivalent, we use the custom VBA function `Effx_AnnEff`. The formula in cell C8 is:

```
=Effx_AnnEff(C7,C3)
```

A discount rate is required for NPV calculations, and it is specified as an annual effective rate in cell C10. This must be converted for use in the NPV function. The formula in cell C11 is:

```
=NPV(AnnEff_Effx(C10,C3),E15:E27)*(1+AnnEff_Effx(C10,C3))
```

In this formula, cell C10 contains the Discount Rate, cell C3 contains the Flow Frequency, and the cash flow range (including the Time 0 flow) is E15:E27.

Recall from Chapter 12 that the following formula is used to calculate an NPV, where an initial flow is present:

```
=NPV(Rate,Range)*(1+Rate)
```

Having calculated the NPV, it is then possible to calculate a derived initial value based on the discount rate of 11%. This initial value is derived by subtracting the calculated NPV from the existing initial value of \$1 million.

This example has stripped DCF down to the bare essentials. In practice, all of those essentials might be subject to many different calculations.

Credit Card Calculations

Chapter 11 described how to use the NPER function to calculate the time required to pay off a loan based on a specified payment amount. Examples in this chapter use amortization schedules that, again, involve calculations based on a fixed payment. Even when variations of interest rate are allowed, the recalculated payments were based on a previously fixed loan term. With credit card calculations, the payment varies according to a more complex set of criteria.

Credit card calculations represent several nonstandard problems. Excel's financial functions (PV, FV, RATE, and NPER) require that the regular payments are at a single level. In addition, the PMT function returns a single level of payments. With IRR and NPV analysis, the user inserts the varying payments into a cash flow.

Credit card companies calculate payments based on the following relatively standard set of criteria:

- ◆ A minimum payment is required. For example, a credit card account might require a minimum payment of \$25.
- ◆ The payment must be at least equal to a base percentage of the outstanding debt. Usually the payment is a percentage of the outstanding balance, but not less than a specified amount.
- ◆ The payment is rounded, usually to the nearest \$0.05.
- ◆ Interest is invariably quoted at a given rate per month.

Figure 13-9 shows a worksheet set up to calculate credit card payments.



This example is available on the companion CD-ROM.

The formulas for the Payment and Interest are rather complicated – just like the terms of a credit card. This example uses a minimum payment amount of \$125, which results in a short term. If you put real data in from a credit card statement (for example, a \$25 minimum payment), you may be surprised at how long it takes to repay the whole balance if you make only minimum repayments (even with no further borrowing).

Of course, things get much more complicated when additional charges are made. In such a case, the formulas would need to account for “grace periods” for purchases (but not cash withdrawals). A further complication is that interest is calculated on the daily outstanding balance at the daily effective equivalent of the quoted rate.

Credit Card Payment Calculations			
Rules			
Loan			(\$500.00)
Effective Rate per month			1.5000%
Min % repayment			3.0000%
Min \$ repayment			\$125.00
Rounding			\$0.05
Calculations			
Annual Nominal Compounded Monthly			18.0000%
Annual Effective			19.5618%
Payments - Interest - Loan			\$0.00
Months to repay			5
CardNPER			5
CardInterest			(\$19.61)
Summary			
Total Payments Made			\$519.95
Total Interest			\$19.95
Schedule			
Term	Payment	Interest	Balance
0			(\$500.00)
1	\$125.00	(\$7.50)	(\$382.50)
2	\$125.00	(\$5.75)	(\$263.25)
3	\$125.00	(\$3.95)	(\$142.20)
4	\$125.00	(\$2.15)	(\$19.35)
5	\$19.95	(\$0.60)	\$0.00
0	\$0.00	\$0.00	\$0.00

Figure 13-9: Calculating a credit card payment schedule

XIRR and XNPV Functions

As discussed in Chapter 12, the IRR and NPV functions assume *regular* periodic cash flows. In some situations, however, the cash flows are not regular. In such a case, you can use the XIRR and XNPV functions. These functions calculate IRRs and NPVs of a cash flow against a schedule of dates, and they use a daily effective equivalent of a given or (in the case of XNPV) calculated annual effective rate.



The XIRR and XNPV functions are available only when the Analysis ToolPak add-in is installed.



The examples in this section are available on the companion CD-ROM.

The XIRR function returns the annual effective rate of return and has the following syntax (arguments in bold are required):

XIRR(**values**,**dates**,*guess*)

The syntax for the XNPV function is (all arguments are required):

XNPV(*rate*,**values**,**dates**)

Figure 13-10 shows a worksheet set up with a cash flow against a schedule of dates.

	A	B	C
1	The XIRR Function		
2			
3	Date	Flow	
4	Feb-05-2001	(\$3,000.00)	
5	Feb-25-2001	\$250.00	
6	Mar-17-2001	\$250.00	
7	Apr-06-2001	\$500.00	
8	Apr-26-2001	\$500.00	
9	May-16-2001	\$600.00	
10	Jun-05-2001	\$400.00	
11	Jun-25-2001	\$200.00	
12	Jul-15-2001	\$200.00	
13	Aug-04-2001	\$200.00	
14			
15	XIRR:	13.7506%	
16	Check XNPV:	(\$0.00)	
17			

Figure 13-10: Using the XIRR function

The formula in cell B15 is:

=XIRR(B4:B13,A4:A13)

Note that the XIRR is reported as an annual effective rate, which is based on a 365-day year assumption. The schedule of dates must be in sequence from the earliest to the latest, and there must be no repeated dates.

The XIRR calculation can be checked by using the XNPV function, discounting at the calculated XIRR. The discount rate must be input as the annual effective rate.

The formula in cell B16, which returns 0, is:

=XNPV(B15,B4:B13,A4:A13)

Figure 13-11 demonstrates the XNPV function, and shows a worksheet set up with a cash flow against a schedule of dates.

	A	B	C
1	The XNPV Function		
2			
3	Interest Rate	8.00%	
4	Rate Type	Nominal	
5	Compounding Frequency	12 Monthly	
6			
7	Annual Effective Equivalent	8.29995%	
8			
9	Date	Flow	Revised Flow
10	Feb-05-2001	\$250.00	(\$3,037.64)
11	Feb-25-2001	\$250.00	\$250.00
12	Mar-17-2001	\$250.00	\$250.00
13	Apr-06-2001	\$500.00	\$500.00
14	Apr-26-2001	\$500.00	\$500.00
15	May-16-2001	\$600.00	\$600.00
16	Jun-05-2001	\$400.00	\$400.00
17	Jun-25-2001	\$200.00	\$200.00
18	Jul-15-2001	\$200.00	\$200.00
19	Aug-04-2001	\$200.00	\$200.00
20			
21	XNPV:	\$3,287.64	
22	Check XNPV:	(\$0.00)	
23	Check XIRR:	8.3000%	
24			

Figure 13-11: Using the XNPV function

The interest rate type (cell B4) uses data validation to allow the user to select either Nominal or Effective. The conversion of the rate to the annual effective rate involves a custom VBA function. The formula in cell B7 is:

```
=IF(B4="Nominal",Nomx_AnnEff(B3,B5),Effx_AnnEff(B3,B5))
```

If a Nominal rate is specified, it is converted to the annual effective rate required by the XNPV function. If an Effective rate is specified, it will be converted to the annual effective rate.



Unlike the NPV function, there is no need to multiply the XNPV by the usual $(1 + \text{DiscountRate})$. It seems that Excel uses the standard definition of NPV (see Chapter 12). However, with daily effective rates being used, the difference is very small.

The XNPV calculation is checked by setting up a revised cash flow (in column C) with the reversed sign XNPV being added to the first cash flow. The revised flow produces an XNPV of 0 using the same discount rate and the XIRR returns the discount rate used to calculate the original XNPV.



The XIRR function has a problem when using multiple internal rates of return. In such a case, an XIRR of 0 is reported, even though the XNPV at that rate is not 0. Accordingly, where multiple IRRs are possible (if the sign changes more than once), it is essential to check the XIRR with an XNPV function. If the result is not 0, then an answer may be obtained by calculating the Present Values of each cash flow using a Goal Seek derived discount rate that produces a sum of the present values equaling 0. Fortunately, the problem is very rare even for changing sign cash flows and appears only to arise where there is a cash flow at the first date in the schedule.

Variable Rate Analysis

Variable-rate loan amortization schedules were covered earlier in this chapter. Variable rates can also be applied to other types of cash flows.

Figure 13-12 shows a worksheet set up to analyze cash flows associated with a building project. No significantly new formula or function concepts are introduced here. However, the worksheet formulas make extensive use of IF functions to build the schedule. The only value inserted into the schedule itself is the varying finance rates (column E).

variable rate analysis.xls								
	A	B	C	D	E	F	G	H
1	Variable Rate Analysis of a Project							
2								
3	Rate Type			Nominal				
4	Compounding Frequency			12				
5	Cash Flow Frequency			4				
6	Debt Finance %			60%				
7	Site Cost			(\$1,000,000.00)				
8	Project Costs			(\$2,000,000.00)				
9	Pre-Building Period			1				
10	Building Period			4				
11	Letting Period			1				
12	Sale Value			\$5,000,000.00				
13								
14	Return on Equity			33.1371%				
15								
16	Project Period	Purchase Sale	Building Costs	Debt	Finance Rate	Interest	Debt Balance	Equity
17	0	(\$1,000,000.00)	\$0.00	(\$600,000.00)	8%	\$0.00	(\$600,000.00)	(\$400,000.00)
18	1	\$0.00	\$0.00	\$0.00	8%	(\$12,080.18)	(\$612,080.18)	\$0.00
19	2	\$0.00	(\$500,000.00)	(\$300,000.00)	8%	(\$12,323.40)	(\$924,403.57)	(\$200,000.00)
20	3	\$0.00	(\$500,000.00)	(\$300,000.00)	7%	(\$18,611.60)	(\$1,243,015.17)	(\$200,000.00)
21	4	\$0.00	(\$500,000.00)	(\$300,000.00)	7%	(\$21,879.90)	(\$1,564,895.08)	(\$200,000.00)
22	5	\$0.00	(\$500,000.00)	(\$300,000.00)	7%	(\$27,545.72)	(\$1,892,440.80)	(\$200,000.00)
23	6	\$3,500,000.00	\$0.00	\$0.00	0%	(\$33,311.28)	(\$1,925,752.08)	\$1,574,247.92
24	0	\$0.00	\$0.00	\$0.00	0%	\$0.00	\$0.00	\$0.00
25	0	\$0.00	\$0.00	\$0.00	0%	\$0.00	\$0.00	\$0.00

Figure 13-12: Variable rate analysis



This example is available on the companion CD-ROM.

The project in this example is very short (for illustration purposes). Following are some points to keep in mind:

- ◆ Formulas in column B (Purchase Sale) use an IF function that inserts the sale proceeds at the end of the development.
- ◆ Formulas in column C (Building Costs) use an IF function to insert a fixed proportion of the building costs during the specified building period.
- ◆ Formulas in column D (Debt) calculate the debt change by applying the debt percentage to the amount of columns B and C.
- ◆ Column E (Finance Rate) contains the user-specified variations of interest on debt.
- ◆ Formulas in column F (Interest) calculate the interest on outstanding debt at the end of the previous period.
- ◆ Formulas in column G (Debt Balance) calculate the rolled-up debt by adding the previous debt, further drawing, and interest.
- ◆ Formulas in column H (Equity) sum the equity position. These formulas use an IF function to adjust the receipt of sale proceeds by the amount of the debt that is fully repaid at the end.
- ◆ The formula in cell D14 uses the data in column H to calculate the return on equity.

This is a highly simplified analysis of a project, but it illustrates all of the basic principles involved in far more complex cases.

Creating Indices

The final topic in this chapter demonstrates how to create an index from schedules of changing values. An index is commonly used to compare how data changes over time. An index allows easy cross-comparison between different periods and between different data sets.

For example, consumer price changes are recorded in an index in which the initial “shopping basket” is based to an index of 100. All subsequent changes are made relative to that base. Therefore, any two points show the cumulative effect of increases. Using indices also makes it easier to compare data that use vastly different scales – such as comparing a consumer price index with a wage index.

Perhaps the best approach is to use a two-step illustration:

- ◆ First, convert the second and subsequent data in the series to percentage increases from the previous item.
- ◆ Set up a column where the first entry is 100 and successive entries increase by the percentage increases previously determined.

Although a two-step approach is not required, a major advantage is that the calculation of the percentage changes is often very useful data in its own right.

The example, shown in Figure 13-13, involves rentals per square foot of different types of space between 1995 and 2001. The raw data is contained in the first table. This data is converted to percentage changes in the second table, and this information is used to create the indices in the third table.



This example is available on the companion CD-ROM.

creating indices.xls								
Creating an Index from Growth Data								
Rentals Per Square Foot								
	1995	1996	1997	1998	1999	2000	2001	
Retail	89	88	97	148	159	187	201	
Office	60	58	60	84	84	92	101	
Industrial	12	11	12	18	18	19	20	
Other	33	32	35	52	53	66	75	
All Property	38	38	40	58	60	69	74	
Growth Data								
	1995	1996	1997	1998	1999	2000	2001	Average
Retail	-0.92%	8.83%	34.52%	6.73%	15.13%	7.14%	11.39%	
Office	-2.97%	3.10%	28.68%	-0.55%	8.98%	9.01%	7.24%	
Industrial	-5.14%	9.41%	31.15%	-0.20%	7.78%	4.58%	7.36%	
Other	-2.79%	8.79%	31.76%	2.14%	20.72%	11.63%	11.47%	
All Property	-2.43%	6.29%	31.43%	2.91%	12.34%	7.68%	9.21%	
Index Data								
	1995	1996	1997	1998	1999	2000	2001	Average
Retail	100	99.08	107.83	145.05	154.81	178.24	190.97	11.39%
Office	100	97.03	100.04	128.74	128.03	139.52	152.09	7.24%
Industrial	100	94.86	103.79	136.12	135.84	146.41	153.11	7.36%
Other	100	97.21	105.76	139.35	142.33	171.83	191.80	11.47%
All Property	100	97.57	103.71	136.30	140.27	157.59	169.68	9.21%

Figure 13-13: Creating an index from growth data

The formulas for calculating the growth rates (in the second table) is simple. For example, the formula in cell C14 is:

```
= (C5 - B5) / C5
```

This formula returns -0.92%, which represents the change in retail space (from \$89 to \$88). This formula is copied to the other cells in the table (range C14:H18). This information is useful, but it is difficult to track overall performance between periods of more than a year. That's why indices are required.

Calculating the indices in the third table is also straightforward. The 1995 index is set at 100 (column B) and is the base for the indices. The formula in cell C23 is:

```
= B23 * (1 + C14)
```

This formula is copied to the other cells in the table (range C23:H27).

These indices make it possible to compare performance of, say, offices between any two years, and to track the relative performance over any two years of any two types of property. So it is clear, for example, that retail property rental grew faster than office rentals between 1995 and 2001.

The average figures (column I) are calculated using the RATE function. This results in an annual growth rate over the entire period.

Summary

This chapter provided examples of common financial analyses. The examples make use of the basic concepts of time value of money and equivalent interest rates.

This concludes the Financial Formulas section of the book. The next section covers a variety of miscellaneous calculations.

Part IV

Array Formulas

CHAPTER 14

Introducing Arrays

CHAPTER 15

Performing Magic with Array Formulas

Chapter 14

Introducing Arrays

IN THIS CHAPTER

- ◆ The definition of an array and an array formula
- ◆ One-dimensional vs. two-dimensional arrays
- ◆ How to work with array constants
- ◆ Techniques for working with array formulas
- ◆ Examples of multicell array formulas
- ◆ Examples of array formulas that occupy a single cell

ONE OF EXCEL'S MOST INTERESTING (and most powerful) features is its ability to work with arrays in a formula. When you understand this concept, you'll be able to create elegant formulas that appear to perform magic. This chapter introduces the concept of arrays, and is required reading for anyone who wants to become a master of Excel formulas. Chapter 15 continues with lots of useful examples.

Introducing Array Formulas

If you do any computer programming, you've probably been exposed to the concept of an array. An *array* is simply a collection of items operated on collectively or individually. In Excel, an array can be one-dimensional or two-dimensional. These dimensions correspond to rows and columns. For example, a *one-dimensional array* can be stored in a range that consists of one row (a horizontal array) or one column (a vertical array). A *two-dimensional array* can be stored in a rectangular range of cells. Excel doesn't support three-dimensional arrays (but its VBA programming language does).

But, as you'll see, arrays need not be stored in cells. You can also work with arrays that exist only in Excel's memory. You can then use an *array formula* to manipulate this information and return a result. An array formula can occupy multiple cells, or reside in a single cell.

This section presents two array formula examples: an array formula that occupies multiple cells, and another array formula that occupies only one cell.

A Multicell Array Formula

Figure 14-1 shows a simple worksheet set up to calculate product sales. Normally, you would calculate the value in column D (total sales per product) with a formula such as the one that follows, and then copy this formula down the column.

=B2*C2

After copying the formula, the worksheet contains six formulas in column D.

	A	B	C	D	E
1	Product	Units Sold	Unit Price	Total	
2	AR-988	3	\$50	\$150	
3	BZ-011	10	\$100	\$1,000	
4	MR-919	5	\$20	\$100	
5	TR-811	9	\$10	\$90	
6	TS-333	3	\$60	\$180	
7	ZL-001	1	\$200	\$200	
8					
9					

Figure 14-1: The range D2:D7 contains a single array formula.

Another alternative uses a *single* formula (an array formula) to calculate all six values in D2:D7. This single formula occupies six cells and returns an array of six values.

To create a single array formula to perform the calculations, follow these steps:

1. Select a range to hold the results. In this case, the range is D2:D7.
2. Enter the following formula:


```
=B2:B7*C2:C7
```
3. Normally, you press Enter to enter a formula. Because this is an array formula, however, press Ctrl+Shift+Enter.

The formula is entered into all six of the selected cells. If you examine the formula bar, you'll see the following:

```
{=B2:B7*C2:C7}
```

Excel places brackets around the formula to indicate that it's an array formula.

This formula performs its calculations and returns a six-item array. The array formula actually works with two other arrays, both of which happen to be stored in ranges. The values for the first array are stored in B2:B7, and the values for the second array are stored in C2:C7.

Because it's not possible to display more than one value in a single cell, six cells are required to display the resulting array. That explains why you selected six cells before you entered the array formula.

This array formula, of course, returns exactly the same values as these six normal formulas entered into individual cells in D2:D7:

```
=B2*C2  
=B3*C3  
=B4*C4  
=B5*C5  
=B6*C6  
=B7*C7
```

Using a single array formula rather than individual formulas does offer a few advantages:

- ◆ It's a good way of ensuring that all formulas in a range are identical.
- ◆ Using a multicell array formula makes it less likely you will overwrite a formula accidentally. You cannot change one cell in a multicell array formula.
- ◆ Using a multicell array formula will almost certainly prevent novices from tampering with your formulas.

A Single-Cell Array Formula

Now it's time to take a look at a single-cell array formula. Refer again to Figure 14-1. The following array formula occupies a single cell:

```
{=SUM(B2:B7*C2:C7)}
```

You can enter this formula into any cell. But when you enter this formula, make sure you use Ctrl+Shift+Enter (and don't type the curly brackets).

This array formula returns the sum of the total product sales. It's important to understand that this formula does not rely on the information in column D. In fact, you can delete column D and the formula will still work.

This formula works with two arrays, both of which are stored in cells. The first array is stored in B2:B7, and the second array is stored in C2:C7. The formula multiplies the corresponding values in these two arrays and creates a new array (which exists only in memory). The SUM function then operates on this new array and returns the sum of its values.

Creation of an Array Constant

The examples in the previous section used arrays stored in worksheet ranges. The examples in this section demonstrate an important concept: An array does not have to be stored in a range of cells. This type of array, which is stored in memory, is referred to as an *array constant*.

You create an array constant by listing its items and surrounding them with brackets. Here's an example of a five-item vertical array constant:

```
{1,0,1,0,1}
```

The following formula uses the SUM function, with the preceding array constant as its argument. The formula returns the sum of the values in the array (which is 3). Notice that this formula uses an array, but it is not an array formula. Therefore, you do not use Ctrl+Shift+Enter to enter the formula.

```
=SUM({1,0,1,0,1})
```



When you specify an array directly (as shown previously), you must provide the brackets around the array elements. When you enter an array formula, on the other hand, you do not supply the brackets.

At this point, you probably don't see any advantage to using an array constant. The formula that follows, for example, returns the same result as the previous formula:

```
=SUM(1,0,1,0,1)
```

Keep reading, and the advantages will become apparent. Following is a formula that uses two array constants:

```
=SUM({1,2,3,4}*{5,6,7,8})
```

This formula creates a new array (in memory) that consists of the product of the corresponding elements in the two arrays. The new array is:

```
{5,12,21,32}
```

This new array is then used as an argument for the SUM function, which returns the result (70). The formula is equivalent to the following formula, which doesn't use arrays:

```
=SUM(1*5,2*6,3*7,4*8)
```

A formula can work with both an array constant and an array stored in a range. The following formula, for example, returns the sum of the values in A1:D1, each multiplied by the corresponding element in the array constant:

```
=SUM(A1:D1*{1,2,3,4})
```

This formula is equivalent to:

```
=SUM(A1*1,B1*2,C1*3,D1*4)
```

Array Constant Elements

An array constant can contain numbers, text, logical values (TRUE or FALSE), and even error values such as #N/A. Numbers can be in integer, decimal, or scientific format. You must enclose text in double quotation marks (for example, "Tuesday"). You can use different types of values in the same array constant, as in this example:

```
{1,2,3,TRUE,FALSE,TRUE,"Moe","Larry","Curly"}
```

An array constant cannot contain formulas, functions, or other arrays. Numeric values cannot contain dollar signs, commas, parentheses, or percent signs. For example, the following is an invalid array constant:

```
{SQRT(32),$56.32,12.5%}
```

Understanding the Dimensions of an Array

As stated previously, an array can be either one-dimensional or two-dimensional. A one-dimensional array's orientation can be either vertical or horizontal.

One-Dimensional Horizontal Arrays

The elements in a one-dimensional horizontal array are separated by commas. The following example is a one-dimensional horizontal array constant:

```
{1,2,3,4,5}
```

To display this array in a range requires five consecutive cells in a row. To enter this array into a range, select a range of cells that consists of one row and five columns. Then enter `={1,2,3,4,5}` and press **Ctrl+Shift+Enter**.

If you enter this array into a horizontal range that consists of more than five cells, the extra cells will contain #N/A (which denotes unavailable values). If you enter this array into a *vertical* range of cells, only the first item (1) will appear in each cell.

The following example is another horizontal array; it has seven elements and is made up of text strings:

```
{"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"}
```

To enter this array, select seven cells in a row, and type the following (followed by Ctrl+Shift+Enter):

```
={"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"}
```

One-Dimensional Vertical Arrays

The elements in a one-dimensional vertical array are separated by semicolons. The following is a six-element vertical array constant:

```
{10;20;30;40;50;60}
```

Displaying this array in a range requires six cells in a column. To enter this array into a range, select a range of cells that consists of six rows and one column. Then enter the following formula, followed by Ctrl+Shift+Enter:

```
={10;20;30;40;50;60}
```

The following is another example of a vertical array; this one has four elements:

```
{"Widgets";"Sprockets";"Do-Dads";"Thing-A-Majigs"}
```

Two-Dimensional Arrays

A two-dimensional array uses commas to separate its horizontal elements, and semicolons to separate its vertical elements. The following example shows a 3×4 array constant:

```
{1,2,3,4;5,6,7,8;9,10,11,12}
```

To display this array in a range requires 12 cells. To enter this array into a range, select a range of cells that consists of three rows and four columns. Then type the following formula, followed by Ctrl+Shift+Enter:

```
={1,2,3,4;5,6,7,8;9,10,11,12}
```

Figure 14-2 shows how this array appears when entered into a range (in this case, B3:E5).

	A	B	C	D	E	F
1						
2						
3			1	2	3	4
4			5	6	7	8
5			9	10	11	12
6						
7						

Figure 14-2: A 3×4 array, entered into a range of cells

If you enter an array into a range that has more cells than array elements, Excel displays #N/A in the extra cells. Figure 14-3 shows a 3×4 array entered into a 10×5 cell range.

	A	B	C	D	E	F	G
1							
2							
3			1	2	3	4	#N/A
4			5	6	7	8	#N/A
5			9	10	11	12	#N/A
6		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
7		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
8		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
9		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
10		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
11		#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
12							
13							

Figure 14-3: A 3×4 array, entered into a 10×5 cell range

Each row of a two-dimensional array must contain the same number of items. The array that follows, for example, is not valid because the third row contains only three items:

```
{1,2,3,4;5,6,7,8;9,10,11}
```

Excel will not allow you to enter a formula that contains an invalid array.

Naming Array Constants

You can create an array constant, give it a name, and then use this named array in a formula. Technically, a named array is a named formula.



Chapter 3 covers the topic of names and named formulas in detail.

Figure 14-4 shows a named array being created using the Define Name dialog box. The name of the array is *DayNames*, and it refers to the following array constant:

```
{ "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" }
```

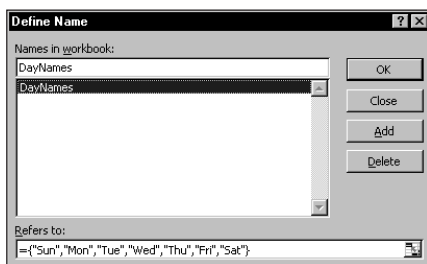


Figure 14-4: Creating a named array constant

Notice that, in the Define Name dialog box, the array is defined using a leading equal sign (=). Without this equal sign, the array is interpreted as a text string rather than an array. Also, you must type the curly brackets when defining a named array constant; Excel does not enter them for you.

After creating this named array, you can use it in a formula. Figure 14-5 shows a worksheet that contains a single array formula entered into the range A1:G1. The formula is:

```
{=DayNames }
```

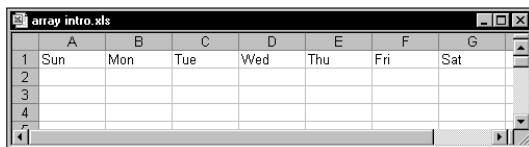


Figure 14-5: Using a named array in an array formula

Because commas separate the array elements, the array has a horizontal orientation. Use semicolons to create a vertical array. Or you can use Excel's TRANSPOSE function to insert a horizontal array into a vertical range of cells (see "Transposing

an Array,” later in this chapter). The following array formula, which is entered into a seven-cell vertical range, uses the TRANSPOSE function:

```
{=TRANSPOSE(DayNames)}
```

You also can access individual elements from the array by using Excel’s INDEX function. The following formula, for example, returns *Wed*, the fourth item in the *DayNames* array:

```
=INDEX(DayNames,4)
```

Working with Array Formulas

This section deals with the mechanics of selecting cells that contain arrays, and entering and editing array formulas. These procedures differ a bit from working with ordinary ranges and formulas.

Entering an Array Formula

When you enter an array formula into a cell or range, you must follow a special procedure so Excel knows that you want an array formula rather than a normal formula. You enter a normal formula into a cell by pressing Enter. You enter an array formula into one or more cells by pressing Ctrl+Shift+Enter.

You can easily identify an array formula, because the formula is enclosed in curly brackets in the formula bar. The following formula, for example, is an array formula:

```
{=SUM(LEN(A1:A5))}
```

Don’t enter the curly brackets when you create an array formula; Excel inserts them for you. If the result of an array formula consists of more than one value, you must select all of the cells in the results range *before* you enter the formula. If you fail to do this, only the first element of the result is returned.

Selecting an Array Formula Range

You can select the cells that contain a multicell array formula manually, by using the normal cell selection procedures. Or you can use either of the following methods:

- ◆ Activate any cell in the array formula range. Select Edit → Go To (or press F5), click the Special button, and then choose the Current Array option. Click OK to close the dialog box.

- ◆ Activate any cell in the array formula range and press Ctrl+/ to select the entire array.

Editing an Array Formula

If an array formula occupies multiple cells, you must edit the entire range as though it is a single cell. The key point to remember is that you can't change just one element of an array formula. If you attempt to do so, Excel displays the messages shown in Figure 14-6.

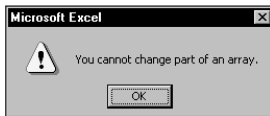


Figure 14-6: Excel's warning message reminds you that you can't edit just one cell of a multicell array formula.

The following rules apply to multicell array formulas. If you try to do any of these things, Excel lets you know about it.

- ◆ You can't change the contents of any individual cell that makes up an array formula.
- ◆ You can't move cells that make up part of an array formula (but you can move an entire array formula).
- ◆ You can't delete cells that form part of an array formula (but you can delete an entire array).
- ◆ You can't insert new cells into an array range. This rule includes inserting rows or columns that would add new cells to an array range.

To edit an array formula, select all the cells in the array range and activate the formula bar as usual (click it or press F2). Excel removes the brackets from the formula while you edit it. Edit the formula and then press Ctrl+Shift+Enter to enter the changes. All of the cells in the array now reflect your editing changes.



If you accidentally press Ctrl+Enter (instead of Ctrl+Shift+Enter) after editing an array formula, the formula will be entered into each selected cell, but it will no longer be an array formula.

Although you can't change any individual cell that makes up a multicell array formula, you can apply formatting to the entire array or to only parts of it.

Expanding or Contracting a Multicell Array Formula

Often, you may need to expand a multicell array formula (to include more cells) or contract it (to include fewer cells). Doing so requires a few steps:

1. Select the entire range that contains the array formula.
2. Press F2 to enter Edit mode.
3. Press Ctrl+Enter. This step enters an identical (non-array) formula into each selected cell.
4. Change your range selection to include additional or fewer cells.
5. Press F2.
6. Press Ctrl+Shift+Enter.

Array Formulas: The Downside

If you've followed along in this chapter, you probably understand some of the advantages of using array formulas. The main advantage, of course, is that an array formula enables you to perform otherwise impossible calculations. As you gain more experience with arrays, you undoubtedly will discover some disadvantages.

Array formulas are one of the least understood features of Excel. Consequently, if you plan to share a workbook with someone who may need to make modifications, you should probably avoid using array formulas. Encountering an array formula when you don't know what it is can be very confusing.

You might also discover that you can easily forget to enter an array formula by pressing Ctrl+Shift+Enter. If you edit an existing array, you still must use these keys to complete the edits. Except for logical errors, this is probably the most common problem that users have with array formulas. If you press Enter by mistake after editing an array formula, just press F2 to get back into Edit mode, and then press Ctrl+Shift+Enter.

Another potential problem with array formulas is that they can slow your worksheet's recalculations, especially if you use very large arrays. On a faster system, this may not be a problem. But, conversely, using an array formula is almost always faster than using a custom VBA function.

Using Multicell Array Formulas

This section contains examples that demonstrate additional features of multicell array formulas (array formulas that are entered into a range of cells). These features include creating arrays from values, performing operations, using functions, transposing arrays, and generating consecutive integers.

Creating an Array from Values in a Range

The following array formula creates an array from a range of cells. Figure 14-7 shows a workbook with some data entered into A1:C4. The range D8:F11 contains a single array formula:

```
{=A1:C4}
```

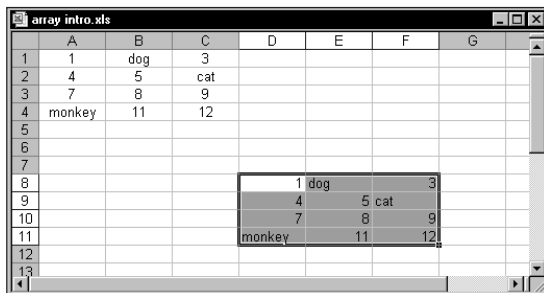


Figure 14-7: Creating an array from a range

The array in D8:F11 is linked to the range A1:C4. Change any value in A1:C4 and the corresponding cell in D8:F11 reflects that change.

Creating an Array Constant from Values in a Range

In the previous example, the array formula in D8:F11 essentially created a link to the cells in A1:C4. It's possible to "sever" this link and create an array constant made up of the values in A1:C4.

To do so, select the cells that contain the array formula (the range D8:F11, in this example). Then press F2 to edit the array formula. Press F9 to convert the cell references to values. Press Ctrl+Shift+Enter to reenter the array formula (which now uses an array constant). The array constant is:

```
{1,"dog",3;4,5,"cat";7,8,9;"monkey",11,12}
```

Figure 14-8 shows how this looks in the formula bar.

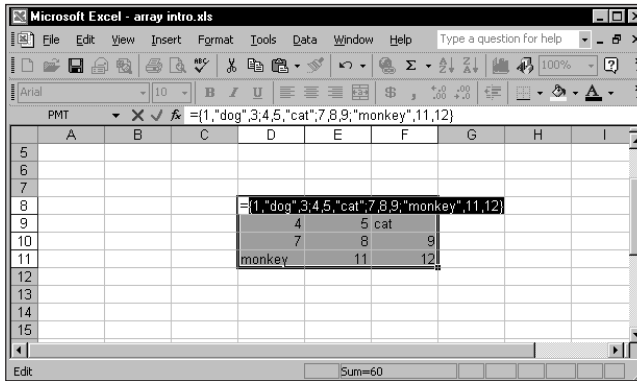


Figure 14-8: After you've pressed F9, the formula bar displays the array constant.

Performing Operations on an Array

So far, most of the examples in this chapter simply entered arrays into ranges. The following array formula creates a rectangular array and multiplies each array element by 2:

```
{={1,2,3,4;5,6,7,8;9,10,11,12}*2}
```

Figure 14-9 shows the result when you enter this formula into a range:

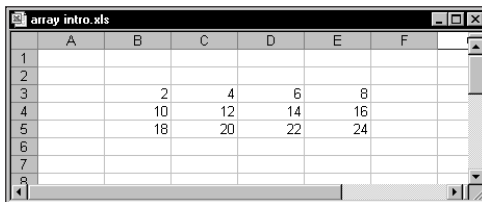


Figure 14-9: Performing a mathematical operation on an array

The following array formula multiplies each array element by itself. Figure 14-10 shows the result when you enter this formula into a range:

```
{={1,2,3,4;5,6,7,8;9,10,11,12}*{1,2,3,4;5,6,7,8;9,10,11,12}}
```


	A	B	C	D	E	F
1						
2						
3		1	4	9	16	
4		25	36	49	64	
5		81	100	121	144	
6						
7						
8						

Figure 14-10: Multiplying each array element by itself

The following array formula is a simpler way of obtaining the same result:

```
{={1,2,3,4;5,6,7,8;9,10,11,12}^2}
```

If the array is stored in a range (such as A1:C4), the array formula returns the square of each value in the range, as follows:

```
{=A1:C4^2}
```

Using Functions with an Array

As you might expect, you also can use functions with an array. The following array formula, which you can enter into a 10-cell vertical range, calculates the square root of each array element in the array constant:

```
{=SQRT({1;2;3;4;5;6;7;8;9;10})}
```

If the array is stored in a range, an array formula such as the one that follows returns the square root of each value in the range:

```
{=SQRT(A1:A10)}
```

Transposing an Array

When you transpose an array, you essentially convert rows to columns and columns to rows. In other words, you can convert a horizontal array to a vertical array (and vice versa). Use Excel's TRANSPOSE function to transpose an array.

Consider the following one-dimensional horizontal array constant:

```
{1,2,3,4,5}
```

You can enter this array into a vertical range of cells by using the TRANSPOSE function. To do so, select a range of five cells that occupy five rows and one column. Then enter the following formula and press Ctrl+Shift+Enter:

```
=TRANSPOSE({1,2,3,4,5})
```

The horizontal array is transposed, and the array elements appear in the vertical range.

Transposing a two-dimensional array works in a similar manner. Figure 14-11 shows a two-dimensional array entered into a range normally, and entered into a range using the TRANSPOSE function. The formula in A1:D3 is:

```
{= {1,2,3,4;5,6,7,8;9,10,11,12}}
```

	A	B	C	D	E
1	1	2	3	4	
2	5	6	7	8	
3	9	10	11	12	
4					
5					
6	1	5	9		
7	2	6	10		
8	3	7	11		
9	4	8	12		
10					
11					
12					

Figure 14-11: Using the TRANSPOSE function to transpose a rectangular array

The formula in A6:C9 is:

```
{=TRANSPOSE({1,2,3,4;5,6,7,8;9,10,11,12})}
```

You can, of course, use the TRANSPOSE function to transpose an array stored in a range. The following formula, for example, uses an array stored in A1:C4 (four rows, three columns). You can enter this array formula into a range that consists of three rows and four columns.

```
{=TRANSPOSE(A1:C4)}
```

Generating an Array of Consecutive Integers

As you will see in Chapter 15, it's often useful to generate an array of consecutive integers for use in an array formula. Excel's ROW function, which returns a row number, is ideal for this. Consider the array formula shown here, entered into a vertical range of 12 cells:

```
{=ROW(1:12)}
```

This formula generates a 12-element array that contains integers from 1 to 12. To demonstrate, select a range that consists of 12 rows and one column, and enter the array formula into the range. You'll find that the range is filled with 12 consecutive integers (see Figure 14-12).

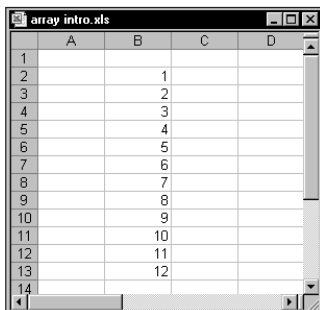


Figure 14-12: Using an array formula to generate consecutive integers

If you want to generate an array of consecutive integers, a formula like the one shown previously is good – but not perfect. To see the problem, insert a new row above the range that contains the array formula. You'll find that Excel adjusts the row references so the array formula now reads:

```
{=ROW(2:13)}
```

The formula that originally generated integers from 1 to 12, now generates integers from 2 to 13.

For a better solution, use this formula:

```
{=ROW(INDIRECT("1:12"))}
```

This formula uses the `INDIRECT` function, which takes a text string as its argument. Excel does not adjust the references contained in the argument for the `INDIRECT` function. Therefore, this array formula *always* returns integers from 1 to 12.



Chapter 15 contains several examples that use the technique for generating consecutive integers.

Using Single-Cell Array Formulas

The examples in the previous section all used a multicell array formula – a single array formula entered into a range of cells. The real power of using arrays becomes apparent when you use single-cell array formulas. This section contains examples of array formulas that occupy a single cell.

Worksheet Functions That Return an Array

Several of Excel's worksheet functions use arrays; you must enter a formula that uses one of these functions into multiple cells as an array formula. These functions are as follows: FORECAST, FREQUENCY, GROWTH, LINEST, LOGEST, MINVERSE, MMULT, and TREND. Consult the online help for more information.

Counting Characters in a Range

Suppose you have a range of cells that contains text entries (see Figure 14-13). If you need to get a count of the total number of characters in that range, the “traditional” method involves creating a formula like the one that follows and copying it down the column:

```
=LEN(A1)
```

	A	B	C	D	E
1	aboriginal				
2	aborigine	Total characters:		112	
3	aborning				
4	abort				
5	abound				
6	about				
7	above				
8	aboveboard				
9	aboveground				
10	abovementioned				
11	abrade				
12	abrasion				
13	abrasive				
14	abreact				
15					
16					

Figure 14-13: A single array formula can count the number of characters in a range of text.

Then, you use a SUM formula to calculate the sum of the values returned by the intermediate formulas.

The following array formula does the job without using any intermediate formulas:

```
{=SUM(LEN(A1:A14))}
```

The array formula uses the LEN function to create a new array (in memory) that consists of the number of characters in each cell of the range. In this case, the new array is:

```
{10,9,8,5,6,5,5,10,11,14,6,8,8,7}
```

The array formula is then reduced to:

```
=SUM({10,9,8,5,6,5,5,10,11,14,6,8,8,7})
```

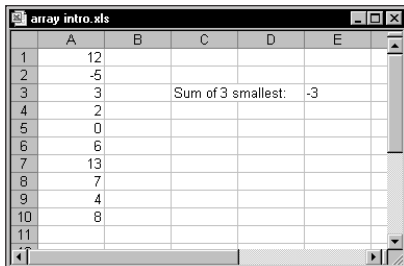
Summing the Three Smallest Values in a Range

The following formula returns the sum of the three smallest values in a range named *Data*:

```
{=SUM(SMALL(Data,{1,2,3}))}
```

The function uses an array constant as the second argument for the SMALL function. This generates a new array, which consists of the three smallest values in the range. This array is then passed to the SUM function, which returns the sum of the values in the new array.

Figure 14-14 shows an example in which the range A1:A10 is named *Data*. The SMALL function is evaluated three times, each time with a different second argument. The first time, the SMALL function has a second argument of 1, and it returns -5. The second time, the second argument for the SMALL function is 2, and it returns 0 (the second smallest value in the range). The third time, the SMALL function has a second argument of 3, and returns the third smallest value of 2.



The screenshot shows an Excel spreadsheet window titled 'array intro.xls'. The spreadsheet has columns A through E and rows 1 through 11. Column A contains the values 12, -5, 3, 2, 0, 6, 13, 7, 4, 8. Cell C3 contains the text 'Sum of 3 smallest:' and cell D3 contains the value '-3'. The spreadsheet is displayed in a standard window with a scroll bar on the right.

	A	B	C	D	E
1	12				
2	-5				
3	3		Sum of 3 smallest:	-3	
4	2				
5	0				
6	6				
7	13				
8	7				
9	4				
10	8				
11					

Figure 14-14: An array formula returns the sum of the three smallest values in A1:A10.

Therefore, the array that's passed to the SUM function is:

```
{-5,0,2}
```

The formula returns the sum of the array (-3).

Counting Text Cells in a Range

The following array formula uses the IF function to examine each cell in a range. It then creates a new array (of the same size and dimensions as the original range) that consists of 1s and 0s, depending on whether the cell contains text. This new

array is then passed to the SUM function, which returns the sum of the items in the array. The result is a count of the number of text cells in the range.

```
{=SUM(IF(ISTEXT(A1:D5),1,0))}
```



This general array formula type (that is, an IF function nested in a SUM function) is very useful for counting. Refer to Chapter 7 for additional examples.

Figure 14-15 shows an example of the preceding formula in cell C8. The array created by the IF function is:

```
{0,1,1,1,1;1,0,0,0,0;1,0,0,0,0;1,0,0,0,0;1,0,0,0,0}
```

	A	B	C	D	E
1		Jan	Feb	Mar	
2	Region 1	7	4	9	
3	Region 2	8	2	8	
4	Region 3	12	1	9	
5	Region 4	14	6	10	
6					
7					
8	No. of text cells:		7		
9					

Figure 14-15: An array formula returns the number of text cells in the range.

Notice that this array contains four rows of three elements (the same dimensions as the range).

A variation on this formula follows:

```
{=SUM(ISTEXT(A1:D5)*1)}
```

This formula eliminates the need for the IF function and takes advantage of the fact that:

TRUE * 1 = 1

and

FALSE * 1 = 0

Eliminating Intermediate Formulas

One of the main benefits of using an array formula is that you can eliminate intermediate formulas in your worksheet. This makes your worksheet more compact, and eliminates the need to display irrelevant calculations. Figure 14-16 shows a worksheet that contains pre-test and post-test scores for students. Column D contains formulas that calculate the changes between the pre-test and the post-test scores. Cell D17 contains a formula, shown here, that calculates the average of the values in column D:

```
=AVERAGE(D2:D15)
```

The screenshot shows an Excel spreadsheet window titled 'array intro.xls'. The spreadsheet contains the following data:

	A	B	C	D
1	Student	Pre-Test	Post-Test	Change
2	Andy	56	67	11
3	Beth	59	74	15
4	Cindy	98	92	-6
5	Duane	78	79	1
6	Eddy	81	100	19
7	Francis	92	94	2
8	Georgia	100	100	0
9	Hilda	92	99	7
10	Isabel	54	69	15
11	Jack	91	92	1
12	Kent	80	88	8
13	Linda	45	68	23
14	Michelle	71	92	21
15	Nancy	94	83	-11
16				
17		Average Change:		7.57
18				

Figure 14-16: Without an array formula, calculating the average change requires intermediate formulas in column D.

With an array formula, you can eliminate column D. The following array formula calculates the average of the changes, but does not require the formulas in column D:

```
{=AVERAGE(C2:C15-B2:B15)}
```

How does it work? The formula uses two arrays, the values of which are stored in two ranges (B2:B15 and C2:C15). The formula creates a *new* array that consists of the differences between each corresponding element in the other arrays. This new array is stored in Excel's memory, not in a range. The AVERAGE function then uses this new array as its argument and returns the result.

The new array consists of the following elements:

```
{11,15,-6,1,19,2,0,7,15,1,8,23,21,-11}
```

The formula, therefore, is reduced to:

```
=AVERAGE({11,15,-6,1,19,2,0,7,15,1,8,23,21,-11})
```

You can use additional array formulas to calculate other measures for the data in this example. For instance, the following array formula returns the largest change (that is, the greatest improvement). This formula returns 23, which represents Linda's test scores.

```
{=MAX(C2:C15-B2:B15)}
```

The following array formula returns the smallest change (that is, the least improvement). This formula returns -11, which represents Nancy's test scores.

```
{=MIN(C2:C15-B2:B15)}
```

Using an Array in Lieu of a Range Reference

If your formula uses a function that requires a range reference, you may be able to replace that range reference with an array constant. This is useful in situations in which the values in the referenced range do not change.



A notable exception to using an array constant in place of a range reference in a function is with the database functions that use a reference to a criteria range (for example, DSUM). Unfortunately, using an array constant instead of a reference to a criteria range does not work.

Figure 14-17 shows a worksheet that uses a lookup table to display a word that corresponds to an integer. For example, looking up a value of 9 returns *Nine* from the lookup table in D1:E10. The formula in cell C1 is:

```
=VLOOKUP(B1,D1:E10,2,FALSE)
```

	A	B	C	D	E	F
1	Number ->	9	Nine	1	One	
2				2	Two	
3				3	Three	
4				4	Four	
5				5	Five	
6				6	Six	
7				7	Seven	
8				8	Eight	
9				9	Nine	
10				10	Ten	
11						
12						

Figure 14-17: You can replace the lookup table in D1:E10 with an array constant.

You can use a two-dimensional array in place of the lookup range. The following formula returns the same result as the previous formula, but it does not require the lookup range in D1:E1:

```
=VLOOKUP(B1,{1,"One";2,"Two";3,"Three";4,"Four";5,"Five";  
6,"Six";7,"Seven";8,"Eight";9,"Nine";10,"Ten"},2,FALSE)
```

Summary

This chapter introduced the concept of *arrays*, collections of items that reside in a range or in Excel's memory. An array formula operates on a range and returns a single value or an array of values.

The next chapter continues this discussion and presents several useful examples that help clarify the concept.

Chapter 15

Performing Magic with Array Formulas

IN THIS CHAPTER

- ◆ More examples of single-cell array formulas
- ◆ More examples of multicell array formulas
- ◆ Returning an array from a custom VBA function

THE PREVIOUS CHAPTER PROVIDED an introduction to arrays and array formulas, and presented some basic examples to whet your appetite. This chapter continues the saga and provides many useful examples that further demonstrate the power of this feature.

I selected the examples in this chapter to provide a good assortment of the various uses for array formulas. Most can be used as-is. You will, of course, need to adjust the range names or references used. Also, you can modify many of the examples easily to work in a slightly different manner.



Each of the examples in this chapter is demonstrated in a file on the companion CD-ROM.

Working with Single-Cell Array Formulas

As I described in the previous chapter, you enter single-cell array formulas into a single cell (not into a range of cells). These array formulas work with arrays contained in a range, or that exist in memory. This section provides some additional examples of such array formulas.

Summing a Range That Contains Errors

You've probably discovered that Excel's SUM function doesn't work if you attempt to sum a range that contains one or more error values (such as #DIV/0! or #N/A). Figure 15-1 shows an example. The SUM formula in cell C9 returns an error value because the range that it sums (C2:C8) contains errors.

	A	B	C	D
1	Total	Number	Per Unit	
2		80	10	8.00
3		120	6	20.00
4		144	12	12.00
5				#DIV/0!
6				#DIV/0!
7		100	20	5.00
8		50	5	10.00
9		TOTAL:		#DIV/0!
10				
11				

Figure 15-1: An array formula can sum a range of values, even if the range contains errors.

The following array formula returns a sum of the values in a range named *Data*, even if the range contains error values:

```
{=SUM(IF(ISERROR(Data),"",Data))}
```

This formula works by creating a new array that contains the original values, but without the errors. The IF function effectively filters out error values by replacing them with an empty string. The SUM function then works on this “filtered” array. This technique also works with other functions, such as MIN and MAX.



You may want to use a function other than ISERROR. The ISERROR function returns TRUE for any error value: #N/A, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!. The ISERR function returns TRUE for any error except #N/A. The ISNA function returns TRUE only if the cell contains #N/A.

Counting the Number of Error Values in a Range

The following array formula is similar to the previous example, but it returns a count of the number of error values in a range named *Data*:

```
{=SUM(IF(ISERROR(Data),1,0))}
```

This formula creates an array that consists of 1s (if the corresponding cell contains an error) and 0s (if the corresponding cell does not contain an error value).

You can simplify the formula a bit by removing the third argument for the IF function. If this argument is not specified, the IF function returns FALSE if the condition is not satisfied (that is, the cell does not contain an error value). The array formula shown here performs exactly like the previous formula, but doesn't use the third argument for the IF function:

```
{=SUM(IF(ISERROR(Data),1))}
```

Actually, you can simplify the formula even more:

```
{=SUM(ISERROR(Data)*1)}
```

This version of the formula relies on the fact that:

TRUE * 1 = 1

and

FALSE * 1 = 0

Summing Based on a Condition

Often, you need to sum values based on one or more conditions. The array formula that follows, for example, returns the sum of the positive values (it excludes negative values) in a range named *Data*:

```
{=SUM(IF(Data>0,Data))}
```

The IF function creates a new array that consists only of positive values and False values. This array is passed to the SUM function, which ignores the False values and returns the sum of the positive values. The *Data* range can consist of any number of rows and columns.

You can also use Excel's SUMIF function for this example. The following formula, which is not an array formula, returns the same result as the previous array formula:

```
=SUMIF(Data,">0")
```

For multiple conditions, however, using SUMIF gets tricky. For example, if you want to sum only values that are greater than 0 and less than or equal to 5, you can use this non-array formula:

```
SUMIF(data,">0",data)-SUMIF(data,">5",data)
```

This formula sums the values that are greater than zero, and then subtracts the sum of the values that are greater than 5. This can be confusing.

Following is an array formula that performs the same calculation:

```
{=SUM((Data>0)*(Data<=5)*Data)}
```

This formula also has a limitation: It will return an error if the *Data* range contains one or more non-numeric cells.



Contrary to what you might expect, you cannot use the AND function in an array formula. The following array formula, while quite logical, doesn't return the correct result:

```
{=SUM(IF(AND(Data>0,Data<=5),Data))}
```

Illogical Behavior from Logical Functions

Excel's AND and OR functions are logical functions that return TRUE or FALSE. Unfortunately, these functions do not perform as expected when used in an array formula.

As shown here, columns A and B contain logical values. The AND function returns TRUE if all of its arguments are TRUE. Column C contains non-array formulas that work as expected. For example, cell C3 contains the following function:

```
=AND(A3, B3)
```

	A	B	C	D	E
1					
2	Condition 1	Condition 2	Non-Array Formulas	Array with AND {=AND(A3:A6, B3:B6)}	Array formula: {=A3:A6*B3:B6}
3	TRUE	TRUE	TRUE	FALSE	1
4	TRUE	FALSE	FALSE	FALSE	0
5	FALSE	TRUE	FALSE	FALSE	0
6	FALSE	FALSE	FALSE	FALSE	0

The range D3:D6 contains this array formula:

```
{=AND(A3:A6, B3:B6)}
```

You might expect this array formula to return the following array:

```
{TRUE, FALSE, FALSE, FALSE}
```

Rather, it returns only a single item: FALSE. In fact, both the AND function and the OR function always return a single result (never an array). Even when using array constants, the AND function still returns only a single value. For example, this array formula does not return an array:

```
{=AND({TRUE,TRUE,FALSE,FALSE},{TRUE,FALSE,TRUE,FALSE})}
```

I don't know if this is by design or if it's a bug. In any case, it certainly is inconsistent with how the other functions operate.

Column E contains another array formula, which follows, that returns an array of 0s and 1s. These 0s and 1s correspond to FALSE and TRUE, respectively.

```
{=A3:A6*B3:B6}
```

In array formulas, you must use this syntax in place of the AND function.

The following array formula, which uses the OR function, does not return an array (as you might expect):

```
=OR(A3:A6,B3:B6)
```

Rather, you can use a formula such as the following, which *does* return an array comprised of logical OR using the corresponding elements in the ranges:

```
{=A3:A6+B3:B6}
```

You can also write an array formula that combines criteria using an OR condition. For example, to sum the values that are less than 0 or greater than 5, use the following array formula:

```
{=SUM(IF((Data<0)+(Data>5),Data))}
```



As with the AND function, you cannot use the OR function in an array formula. The following formula, for example, does not return the correct result:

```
{=SUM(IF(OR(Data<0,Data>5),Data))}
```

For an explanation of the workarounds required for using logical functions in an array formula, refer to the following sidebar, “Illogical Behavior from Logical Functions.”

Summing the *n* Largest Values in a Range

The following array formula returns the sum of the 10 largest values in a range named *Data*:

```
{=SUM(LARGE(Data,ROW(INDIRECT("1:10"))))}
```

The **LARGE** function is evaluated 10 times, each time with a different second argument (1, 2, 3, and so on up to 10). The results of these calculations are stored in a new array, and that array is used as the argument for the **SUM** function.

To sum a different number of values, replace the 10 in the argument for the **INDIRECT** function with another value. To sum the *n smallest* values in a range, use the **SMALL** function instead of the **LARGE** function.

Computing an Average That Excludes Zeros

Figure 15-2 shows a simple worksheet that calculates average sales. The formula in cell B11 is:

```
=AVERAGE(B2:B9)
```

	A	B	C	D	E	F
1	Sales Person	Sales				
2	Abner	23,991				
3	Baker	15,092				
4	Charleston	0				
5	Davis	11,893				
6	Elleman	32,116				
7	Flugelhart	29,089				
8	Gallaway	0				
9	Harrison	33,211				
10						
11		18,174	<-- Average with zeros			
12		24,232	<-- Average without zeros (array formula)			
13						

Figure 15-2: The calculated average includes cells that contain a 0.

This formula, of course, calculates the average of the values in B2:B9. Two of the sales staff had the week off, however, so this average doesn't accurately describe the average sales per representative.



The **AVERAGE** function ignores blank cells, but does not ignore cells that contain 0.

The following array formula returns the average of the range, but excludes the cells containing 0:

```
{=AVERAGE(IF(B2:B9<>0,B2:B9))}
```

This formula creates a new array that consists only of the non-zero values in the range. The `AVERAGE` function then uses this new array as its argument. You also can get the same result with a regular (non-array) formula:

```
=SUM(B2:B9)/COUNTIF(B2:B9,"<>0")
```

This formula uses the `COUNTIF` function to count the number of non-zero values in the range. This value is divided into the sum of the values.

Determining Whether a Particular Value Appears in a Range

To determine whether a particular value appears in a range of cells, you can choose the `Edit → Find` command and do a search of the worksheet. But you also can make this determination by using an array formula.

Figure 15-3 shows a worksheet with a list of names in `A3:E22` (named *NameList*). An array formula in cell `D1` checks the name entered into cell `C1` (named *TheName*). If the name exists in the list of names, the formula displays the text *Found*. Otherwise, it displays *Not Found*.

	A	B	C	D	E
1	Enter a Name -->		Michelle	Found	
2					
3	Al	Daniel	Harold	Lyle	Richard
4	Allen	Dave	Ian	Maggie	Rick
5	Andrew	David	Jack	Margaret	Robert
6	Anthony	Dennis	James	Marilyn	Rod
7	Arthur	Don	Jan	Mark	Roger
8	Barbara	Donald	Jeff	Marvin	Ronald
9	Bernard	Doug	Jeffrey	Mary	Russ
10	Beth	Douglas	Jerry	Matt	Sandra
11	Bill	Ed	Jim	Mel	Scott
12	Bob	Edward	Joe	Merle	Simon
13	Brian	Eric	John	Michael	Stacy
14	Bruce	Fran	Joseph	Michelle	Stephen
15	Carl	Frank	Karl	Mike	Steven
16	Carl	Fred	Kathy	Norman	Stuart
17	Charles	Gary	Keith	Patrick	Susan
18	Chris	George	Kenneth	Paul	Terry
19	Chuck	Glenn	Kevin	Peter	Thomas
20	Clark	Gordon	Larry	Phillip	Timothy
21	Curt	Greg	Leonard	Ray	Vincent
22	Dan	Gregory	Louise	Rebecca	William
23					

Figure 15-3: Using an array formula to determine if a range contains a particular value.

The array formula in cell D1 is:

```
{=IF(OR(TheName=NameList),"Found","Not Found")}
```

This formula compares *TheName* to each cell in the *NameList* range. It builds a new array that consists of logical TRUE or FALSE values. The OR function returns TRUE if any one of the values in the new array is TRUE. The IF function uses this result to determine which message to display.

A simpler form of this formula follows. This formula displays TRUE if the name is found, and returns FALSE otherwise.

```
{=OR(TheName=NameList)}
```

Counting the Number of Differences in Two Ranges

The following array formula compares the corresponding values in two ranges (named *MyData* and *YourData*), and returns the number of differences in the two ranges. If the contents of the two ranges are identical, the formula returns 0.

```
{=SUM(IF(MyData=YourData,0,1))}
```

The two ranges must be the same size and of the same dimensions.

This formula works by creating a new array of the same size as the ranges being compared. The IF function fills this new array with 0s and 1s (0 if a difference is found, 1 if the corresponding cells are the same). The SUM function then returns the sum of the values in the array.

The following formula, which is simpler, is another way of calculating the same result.

```
{=SUM(1*(MyData<>YourData))}
```

This version of the formula relies on the fact that:

TRUE * 1 = 1

and

FALSE * 1 = 0

Returning the Location of the Maximum Value in a Range

The following array formula returns the row number of the maximum value in a single-column range named *Data*:

```
{=MIN(IF(Data=MAX(Data),ROW(Data), ""))}
```

The IF function creates a new array that corresponds to the *Data* range. If the corresponding cell contains the maximum value in *Data*, then the array contains the row number; otherwise, it contains an empty string. The MIN function uses this new array as its second argument and returns the smallest value, which corresponds to the row number of the maximum value in *Data*.

If the *Data* range contains more than one cell that has the maximum value, the row of the first maximum cell is returned.

The following array formula is similar to the previous one, but it returns the actual cell address of the maximum value in the *Data* range. It uses the ADDRESS function, which takes two arguments: a row number and a column number.

```
{=ADDRESS(MIN(IF(Data=MAX(Data),ROW(Data), "")),COLUMN(Data))}
```

Finding the Row of a Value's *n*th Occurrence in a Range

The following array formula returns the row number within a single-column range named *Data* that contains the *n*th occurrence of a cell named *Value*:

```
{=SMALL(IF(Data=Value,ROW(Data), ""),n)}
```

The IF function creates a new array that consists of the row number of values from the *Data* range that are equal to *Value*. Values from the *Data* range that are not equal to *Value* are replaced with an empty string. The SMALL function works on this new array, and returns the *n*th smallest row number.

The formula returns #NUM! if the *Value* is not found or if *n* exceeds the number of the values in the range.

Returning the Longest Text in a Range

The following array formula displays the text string in a range (named *Data*) that has the most characters. If multiple cells contain the longest text string, the first cell is returned.

```
{=INDEX(Data,MATCH(MAX(LEN(Data)),LEN(Data),FALSE),1)}
```

This formula works with two arrays, both of which contain the length of each item in the *Data* range. The MAX function determines the largest value, which corresponds to the longest text item. The MATCH function calculates the offset of the cell that contains the maximum length. The INDEX function returns the contents of the cell containing the most characters. This function works only if the *Data* range consists of a single column.

Determining Whether a Range Contains Valid Values

You might have a list of items that you need to check against another list. For example, you might import a list of part numbers into a range named *MyList*, and you want to ensure that all of the part numbers are valid. You can do this by comparing the items in the imported list to the items in a master list of part numbers (named *Master*).

The following array formula returns TRUE if every item in the range named *MyList* is found in the range named *Master*. Both of these ranges must consist of a single column, but they don't need to contain the same number of rows.

```
{=ISNA(MATCH(TRUE,ISNA(MATCH(MyList,Master,0)),0))}
```

The array formula that follows returns the number of invalid items. In other words, it returns the number of items in *MyList* that do not appear in *Master*.

```
{=SUM(1*ISNA(MATCH(MyList,Master,0)))}
```

To return the first invalid item in *MyList*, use the following array formula:

```
{=INDEX(MyList,MATCH(TRUE,ISNA(MATCH(MyList,Master,0)),0))}
```

Summing the Digits of an Integer

The following array formula calculates the sum of the digits in a positive integer, which is stored in cell A1. For example, if cell A1 contains the value 409, the formula returns 13 (the sum of 4, 0, and 9).

```
{=SUM(MID(A1,ROW(INDIRECT("1:"&LEN(A1))),1)*1)}
```

To understand how this formula works, let's start with the ROW function, shown here:

```
=ROW(INDIRECT("1:"&LEN(A1)))
```

This function returns an array of consecutive integers beginning with 1 and ending with the number of digits in the value in cell A1. For example, if cell A1 contains the value 409, then the LEN function returns 3 and the array generated by the ROW function is:

```
{1,2,3}
```



For more information about using the INDIRECT function to return this array, see Chapter 14.

This array is then used as the second argument for the MID function. The MID part of the formula, simplified a bit and expressed as values, is the following:

```
=MID(409, {1,2,3}, 1)*1
```

This function generates an array with three elements:

```
{4,0,9}
```

By simplifying again and adding the SUM function, the formula looks like this:

```
=SUM({4,0,9})
```

This produces the result of 13.



The values in the array created by the MID function are multiplied by 1 because the MID function returns a string. Multiplying by 1 forces a numeric value result. Alternatively, you can use the VALUE function to force a numeric string to become a numeric value.

Notice that the formula does not work with a negative value because the negative sign is not a numeric value. The following formula solves this problem by using the ABS function to return the absolute value of the number. Figure 15-4 shows a worksheet that uses this formula in cell B2.

```
{=SUM(VALUE(MID(ABS(A2),ROW(INDIRECT("1:"&LEN(ABS(A2))))),1)))}
```

The formula was copied down to calculate the sum of the digits for other values in column A.

	A	B	C
1	Number	Sum of Digits	
2	132	6	
3	9	9	
4	111111	6	
5	980991	36	
6	-980991	36	
7	409	13	
8		0	
9	12	3	
10	123	6	
11			
12			

Figure 15-4: An array formula calculates the sum of the digits in an integer.

Summing Rounded Values

Figure 15-5 shows a simple worksheet that demonstrates a common spreadsheet problem: rounding errors. As you can see, the grand total in cell E5 appears to display an incorrect amount (that is, it's off by a penny). The values in column E use a number format that displays two decimal places. The actual values, however, consist of additional decimal places that do not display due to rounding (as a result of the number format). The net effect of these rounding errors is a seemingly incorrect total. The total, which is actually \$168.320997, displays as \$168.32.

	A	B	C	D	E
1	Description	Quantity	Unit Price	Discount	Total
2	Widgets	6	\$11.69	5.23%	\$66.47
3	Sprockets	8	\$9.74	5.23%	\$73.84
4	Snapholytes	3	\$9.85	5.23%	\$28.00
5	GRAND TOTAL				\$168.32
6					
7					
8					

Figure 15-5: Using an array formula to correct rounding errors

The following array formula creates a new array that consists of values in column E, rounded to two decimal places:

```
{=SUM(ROUND(E2:E4,2))}
```

This formula returns \$168.31.

You also can eliminate these types of rounding errors by using the ROUND function in the formula that calculates each row total in column E. This technique does not require an array formula.



Refer to Chapter 10 for more information about Excel's functions that are relevant to rounding.

Summing Every *n*th Value in a Range

Suppose you have a range of values and you want to compute the sum of every third value in the list – the first, the fourth, the seventh, and so on. One solution is to hard code the cell addresses in a formula. But a better solution is to use an array formula.

Refer to the data in Figure 15-6. The values are stored in a range named *Data*, and the value of *n* is in cell E6 (named *n*).

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	
1		Data								
2		1		3	= nth value					
3		2		70	= Result returned by a single array formula					
4		3								
5		4								
6		5								
7		6								
8		7								
9		8								
10		9								
11		10								
12		11								
13		12								
14		13								
15		14								
16		15								
17		16								
18		17								
19		18								
20		19								
21										

Figure 15-6: An array formula returns the sum of every *n*th value in the range.

The following array formula returns the sum of every *n*th value in the range:

```
{SUM(IF(MOD(ROW(INDIRECT("1:"&COUNT(Data)))-1,n)=0,Data,""))}
```

This formula generates an array of consecutive integers, and the MOD function uses this array as its first argument. The second argument for the MOD function is the value of *n*. The MOD function creates another array that consists of the remainders when each row number is divided by *n*. When the array item is 0 (that is, the row is evenly divisible by *n*), the corresponding item in the *Data* range will be included in the sum.

You'll find that this formula fails when n is 0 (that is, sums no items). The modified array formula that follows uses an IF function to handle this case:

```
{=IF(n=0,0,SUM(IF(MOD(ROW(INDIRECT("1:"&COUNT(data)))-1,n)=0,data,"")))}
```

This formula works only when the *Data* range consists of a single column of values. It does not work for a multicolumn range, or for a single row of values.

To make the formula work with a horizontal range, you need to transpose the array of integers generated by the ROW function. The modified array formula that follows works only with a horizontal *Data* range:

```
{=IF(n=0,0,SUM(IF(MOD(TRANSPOSE(ROW(INDIRECT("1:"&COUNT(Data))))-1,n)=0,Data,"")))}
```

Removing Non-Numeric Characters from a String

The following array formula extracts a number from a string that contains text. For example, consider the string *ABC145Z*. The formula returns the numeric part, 145.

```
{=MID(A1,MATCH(0,(ISERROR(MID(A1,ROW(INDIRECT("1:"&LEN(A1))),1)*1)*1),0),LEN(A1)-SUM((ISERROR(MID(A1,ROW(INDIRECT("1:"&LEN(A1))),1)*1)*1)))}
```

This formula works only with a single embedded number. For example, it fails with a string such as *X45Z99*.

Determining the Closest Value in a Range

The array formula that follows returns the value in a range named *Data* that is closest to a another value (named *Target*):

```
{=INDEX(Data,MATCH(SMALL(ABS(Target-Data),1),ABS(Target-Data),0))}
```

If two values in the *Data* range are equidistant from the *Target* value, the formula returns the first one in the list. Figure 15-7 shows an example of this formula. In this case, the *Target* value is 45. The array formula in cell D3 returns 48 – the value closest to 45.

Returning the Last Value in a Column

Suppose you have a worksheet that you update frequently by adding new data to columns. You might need a way to reference the last value in column A (the value most recently entered). If column A contains no empty cells, the solution is relatively simple, and doesn't require an array formula:

```
=OFFSET(A1,COUNTA(A:A)-1,0)
```

	A	B	C	D	E
1	Data				
2	-12		Target Value -->	45	
3	-4		Closest Match:	48	
4	4				
5	12				
6	20				
7	32				
8	40				
9	48				
10	56				
11	72				
12	80				
13	88				
14	96				
15	97				
16	105				
17	137				
18	145				
19	165				
20	173				
21					

Figure 15-7: An array formula returns the closest match.

This formula uses the COUNTA function to count the number of nonempty cells in column A. This value (minus 1) is used as the second argument for the OFFSET function. For example, if the last value is in row 100, COUNTA returns 100. The OFFSET function returns the value in the cell 99 rows down from cell A1, in the same column.

If column A has one or more empty cells interspersed, which is frequently the case, the preceding formula won't work because the COUNTA function doesn't count the empty cells.

The following array formula returns the contents of the last nonempty cell in the first 500 rows of column A:

```
{=INDEX(A1:A500,MAX(ROW(A1:A500)*(A1:A500<>"")))}
```

You can, of course, modify the formula to work with a column other than column A. To use a different column, change the four column references from A to whatever column you need. If the last nonempty cell occurs in a row beyond row 500, you need to change the two instances of "500" to a larger number. The fewer rows referenced in the formula, the faster the calculation speed.



You cannot use this formula, as written, in the same column with which it's working. Attempting to do so generates a circular reference. You can, however, modify it. For example, to use the function in cell A1, change the references so they begin with row 2.

Returning the Last Value in a Row

The following array formula is similar to the previous formula, but it returns the last nonempty cell in a row (in this case, row 1):

```
{=INDEX(1:1,MAX(COLUMN(1:1)*(1:1<>"")))}
```

To use this formula for a different row, change the 1:1 reference to correspond to the row.

Ranking Data with an Array Formula

Often, computing the rank orders for the values in a range of data is helpful. If you have a worksheet containing the annual sales figures for 20 salespeople, for example, you may want to know how each person ranks, from highest to lowest.

If you've used Excel's RANK function, you may have noticed that the ranks produced by this function don't handle ties the way that you may like. For example, if two values are tied for third place, the RANK function gives both of them a rank of 3. You may prefer to assign each an average (or midpoint) of the ranks – in other words, a rank of 3.5 for both values tied for third place.

Figure 15-8 shows a worksheet that uses two methods to rank a column of values (named *Sales*). The first method (column C) uses Excel's RANK function. Column D uses array formulas to compute the ranks.

	A	B	C	D	E	F	G
1	Salesperson	Sales	Excel's Rank Function	Ranks With Array Formula			
2	Adams	123,000	6	6			
3	Bigelow	98,000	9	10			
4	Fredericks	98,000	9	10	Assigned middle rank		
5	Georgio	98,000	9	10			
6	Jensen	25,000	12	12			
7	Juarez	101,000	8	8			
8	Klein	305,000	1	1			
9	Lynch	145,000	3	3.5			
10	Meyne	145,000	3	3.5	Assigned average rank		
11	Robertson	121,000	7	7			
12	Slokum	124,000	5	5			
13	Wu	150,000	2	2			
14							

Figure 15-8: Ranking data with Excel's RANK function and with array formulas

The following is the array formula in cell D2:

```
{=SUM(1*(B2<=Sales))-(SUM(1*(B2=Sales))-1)/2}
```

This formula is copied to the cells below it.



Each ranking is computed with a separate array formula, not with an array formula entered into multiple cells.

Each array function works by computing the number of higher values and subtracting one half of the number of equal values minus 1.

Creating a Dynamic Crosstab Table

A crosstab table tabulates or summarizes data across two dimensions. Take a look at the data in Figure 15-9. This worksheet shows a simple expense account listing. Each item consists of the date, the expense category, and the amount spent. Each column of data is a named range, indicated in the first row.

	A	B	C	D	E	F	G	H	
1	Date	Category	Amount						
2	4-Jan	Food	23.50			Transp	Food	Lodging	
3	4-Jan	Transp	15.00			4-Jan	160.50	49.57	65.95
4	4-Jan	Food	9.12			5-Jan	20.00	27.80	89.00
5	4-Jan	Food	16.95			6-Jan	0.00	101.96	75.30
6	4-Jan	Transp	145.50			7-Jan	11.50	25.00	112.00
7	4-Jan	Lodging	65.95						
8	5-Jan	Transp	20.00						
9	5-Jan	Food	7.80						
10	5-Jan	Food	20.00						
11	5-Jan	Lodging	89.00						
12	6-Jan	Food	9.00						
13	6-Jan	Food	3.50						
14	6-Jan	Food	11.02						
15	6-Jan	Food	78.44						
16	6-Jan	Lodging	75.30						
17	7-Jan	Transp	11.50						
18	7-Jan	Food	15.50						
19	7-Jan	Food	9.50						
20	7-Jan	Lodging	112.00						
21									

Figure 15-9: You can use array formulas to summarize data such as this in a dynamic crosstab table.

Array formulas summarize this information into a handy table that shows the total expenses – by category – for each day. Cell F3 contains the following array formula, which is copied to the remaining 14 cells in the table:

```
{=SUM(( $E3=Date )*( F$2=Category )*Amount )}
```

These array formulas display the totals for each day, by category.

The formula sums the values in the *Amount* range, but does so only if the row and column names in the summary table match the corresponding entries in the *Date* and *Category* ranges. It does so by multiplying two Boolean values by the *Amount*. If both Boolean values are True, the result is the Amount. If one or both of the Boolean values is False, the result is 0.

You can customize this technique to hold any number of different categories and any number of dates. You can eliminate the dates, in fact, and substitute people's names, departments, regions, and so on.



You also can use Excel's pivot table feature to summarize data in this way. However, pivot tables do not update automatically when the data changes, so the array formula method described here has at least one advantage.

Working with Multicell Array Formulas

The previous chapter introduced array formulas entered into multicell ranges. In this section, I present a few more array multicell formulas. Most of these formulas return some or all of the values in a range, but rearranged in some way.

Returning Only Positive Values from a Range

The following array formula works with a single-column vertical range (named *Data*). The array formula is entered into a range that's the same size as *Data*, and returns only the positive values in the *Data* range (0s and negative numbers are ignored).

```
{=INDEX(Data,SMALL(IF(Data>0,ROW(INDIRECT("1:"&ROWS(Data))))),
ROW(INDIRECT("1:"&ROWS(Data))))}
```

As you can see in Figure 15-10, this formula works but not perfectly. The *Data* range is A2:A21, and the array formula is entered into C2:C21. However, the array formula displays #NUM! error values for cells that don't contain a value.

This more complex array formula avoids the error value display:

```
{=IF(ISERR(SMALL(IF(Data>0,ROW(INDIRECT("1:"&ROWS(Data))))),
ROW(INDIRECT("1:"&ROWS(Data))))), "", INDEX(Data,SMALL(IF
(Data>0,ROW(INDIRECT("1:"&ROWS(Data))))),ROW(INDIRECT
("1:"&ROWS(Data))))))}
```

	A	B	C
1	Data		Positive Vals
2	33		33
3	-33		44
4	44		4
5	4		43
6	-5		99
7	0		5
8	43		6
9	-1		7
10	-2		8
11	-3		9
12	-33		10
13	99		11
14	5		12
15	6		#NUM!
16	7		#NUM!
17	8		#NUM!
18	9		#NUM!
19	10		#NUM!
20	11		#NUM!
21	12		#NUM!

Figure 15-10: Using an array formula to return only the positive values in a range

Returning Nonblank Cells from a Range

The following formula is a variation on the formula in the previous section. This array formula works with a single-column vertical range named *Data*. The array formula is entered into a range of the same size as *Data*, and returns only the nonblank cell in the *Data* range.

```
{=IF(ISERR(SMALL(IF(Data<>"",ROW(INDIRECT("1:"&ROWS(Data)))),
ROW(INDIRECT("1:"&ROWS(Data)))),""),INDEX(Data,SMALL(IF(Data
<>"",ROW(INDIRECT("1:"&ROWS(Data))),ROW(INDIRECT("1:"&ROWS
(Data))))))}
```

Reversing the Order of the Cells in a Range

The following array formula works with a single-column vertical range (named *Data*). The array formula, which is entered into a range of the same size as *Data*, returns the values in *Data*, but in reverse order.

```
{=IF(INDEX(Data,ROWS(data)-ROW(INDIRECT("1:"&ROWS(Data)))+1)
="","",INDEX(Data,ROWS(Data)-ROW(INDIRECT("1:"&ROWS(Data))
+1))}
```

Figure 15-11 shows this formula in action. The range A2:A20 is named *Data*, and the array formula is entered into the range C2:C20.

	A	B	C
1	Data Entry Range	Reversed	
2	first	10	
3	second	9	
4	third	8	
5	fourth	7th	
6	5th	6th	
7	6th	5th	
8	7th	fourth	
9	8	third	
10	9	second	
11	10	first	
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			

Figure 15-11: A multicell array formula reverses the order of the values in the range.

Sorting a Range of Values Dynamically

Suppose your worksheet contains a single-column vertical range named *Data*. The following array formula, entered into a range with the same number of rows as *Data*, returns the values in *Data*, sorted from highest to lowest. This formula works only with numeric values, not with text.

```
{=LARGE(Data,ROW(INDIRECT("1:"&ROWS(Data))))}
```

To sort the values in *Data* from lowest to highest, use this array formula:

```
{=SMALL(Data,ROW(INDIRECT("1:"&ROWS(Data))))}
```

This formula can be useful if you need to have your data entry sorted immediately. Start by defining the range name *Data* as your data entry range. Then enter the array formula into another range with the same number of rows as *Data*.

You'll find that the array formula returns #NUM! for cells that don't have a value. This can be annoying if you're entering data. The modified version, which follows, is more complex, but it eliminates the display of the error value:

```
{=IF(ISERR(LARGE(Data,ROW(INDIRECT("1:"&ROWS(Data))))), "",  
LARGE(Data,ROW(INDIRECT("1:"&ROWS(Data))))}
```

Returning a List of Unique Items in a Range

If you have a single-column range named *Data*, the following array formula returns a list of the unique items in the range:

```
{=INDEX(Data,SMALL(IF(MATCH(Data,Data,0)=ROW(INDIRECT("1:"&ROWS(Data)
))),
MATCH(Data,Data,0),""),ROW(INDIRECT("1:"&ROWS(Data))))}
```

This formula does not work if the *Data* range contains any blank cells. The unfilled cells of the array formula display #NUM!. Figure 15-12 shows an example. Range A2:A20 is named *Data*, and the array formula is entered into range C2:C20.

	A	B	C	D
1	Data		Unique Items	
2	Dog		Dog	
3	Dog		Cat	
4	Dog		Monkey	
5	Dog		Elephant	
6	Cat		Pigeon	
7	Cat		Donkey	
8	Cat		#NUM!	
9	Cat		#NUM!	
10	Monkey		#NUM!	
11	Cat		#NUM!	
12	Elephant		#NUM!	
13	Elephant		#NUM!	
14	Elephant		#NUM!	
15	Pigeon		#NUM!	
16	Pigeon		#NUM!	
17	Pigeon		#NUM!	
18	Donkey		#NUM!	
19	Dog		#NUM!	
20	Monkey		#NUM!	
21				

Figure 15-12: Using an array formula to return unique items from a list

Displaying a Calendar in a Range

Figure 15-13 shows a calendar displayed in a range of cells. The worksheet has two defined names: *m* (for the month) and *y* (for the year). A single array formula, entered into 42 cells, displays the corresponding calendar. The following array formula is entered into the range B6:H11:

```
{=IF(MONTH(DATE(y,m,1))<>MONTH(DATE(y,m,1)-(WEEKDAY(
DATE(y,m,1))-1)+{0;7;14;21;28;35}+
{0,1,2,3,4,5,6}),"",DATE(y,m,1)-(WEEKDAY(
DATE(y,m,1))-1)+{0;7;14;21;28;35}+{0,1,2,3,4,5,6})}
```

The array formula actually returns date values, but the cells are formatted to display only the day portion of the date. Also, notice that the array formula uses array constants. You can simplify the array formula quite a bit by removing the IF function.

```
{=DATE(y,m,1)-(WEEKDAY(
DATE(y,m,1))-1)+{0;7;14;21;28;35}+
{0,1,2,3,4,5,6}}
```



Figure 15-13: Displaying a calendar using a single array formula



See Chapter 14 for more information about array constants.

This version of the formula displays the days from the preceding month and the next month. The IF function in the original formula checks each date to make sure it's in the current month. If not, the IF function returns an empty string.

Returning an Array from a Custom VBA Function

The chapter's final example demonstrates one course of action you can take if you can't figure out a particular array formula. If Excel doesn't provide the tools you need, you need to create your own.

For example, I struggled for several hours in an attempt to create an array formula that returns a sorted list of text entries. Although you can create an array formula that returns a sorted list of *values* (see "Sorting a Range of Values Dynamically," earlier in this chapter), doing the same for text entries is much more challenging.

The following formula works, but only if the *Data* range does not contain any duplicate entries.

```
{=INDEX(Data,MATCH(ROW(INDIRECT("1:"&COUNTA(Data))),
COUNTIF(Data,"<="&Data),0))}
```

Therefore, I created a custom VBA function called SORTED, which I list here:

```
Function SORTED(rng, Optional ascending) As Variant
    Dim SortedData() As Variant
    Dim CellCount As Long
    Dim Temp As Variant, i As Long, j As Long
    CellCount = rng.Count
    ReDim SortedData(1 To CellCount)

    ' Check optional argument
    If IsMissing(ascending) Then ascending = True

    ' Exit with an error if not a single column
    If rng.Columns.Count > 1 Then
        SORTED = CVErr(xlErrValue)
        Exit Function
    End If

    ' Transfer data to SortedData
    For i = 1 To CellCount
        SortedData(i) = rng(i)
        If TypeName(SortedData(i)) = "Empty" _
            Then SortedData(i) = ""
    Next i
    On Error Resume Next

    ' Sort the SortedData array
    For i = 1 To CellCount
        For j = i + 1 To CellCount
            If SortedData(j) <> "" Then
                If ascending Then
                    If SortedData(i) > SortedData(j) Then
                        Temp = SortedData(j)
                        SortedData(j) = SortedData(i)
                        SortedData(i) = Temp
                    End If
                Else
                    If SortedData(i) < SortedData(j) Then
                        Temp = SortedData(j)
                        SortedData(j) = SortedData(i)
                        SortedData(i) = Temp
                    End If
                End If
            End If
        Next j
    Next i
End Function
```



```
        End If
    End If
Next j
Next i

' Transpose it
    SORTED = Application.Transpose(SortedData)
End Function
```



Refer to Part V for information about creating custom VBA functions.

The SORTED function takes two arguments: a range reference and an optional second argument that specifies the sort order. The default sort order is ascending order. If you specify FALSE as the second argument, the range is returned sorted in descending order.

Once the SORTED function procedure is entered into a VBA module, you can use the SORTED function in your formulas. The following array formula, for example, returns the contents of a single-column range named *Data*, but sorted in ascending order. You enter this formula into a range the same size as the *Data* range.

```
{=SORTED(Data)}
```

This array formula returns the contents of the *Data* range, but sorted in descending order:

```
{=SORTED(Data,False)}
```

As you can see, using a custom function results in a much more compact formula. Custom functions, however, are usually much slower than formulas that use Excel's built-in functions.

Figure 15-14 shows an example of this function used in an array formula. Range A2:A17 is named *Data*, and the array formula is entered into range C2:C17.

	A	B	C	D
1	Data Entry		Sorted Data	
2	Arias		Arias	
3	Davis		Colangelo	
4	Trammell		Darr	
5	Jackson		Davis	
6	Magadan		Gonzalez	
7	Mendez		Gwynn	
8	Kotsay		Henderson	
9	Colangelo		Jackson	
10			Klesko	
11	Nevin		Kotsay	
12	Henderson		Magadan	
13	Klesko		Mendez	
14	Perez		Nevin	
15	Gwynn		Perez	
16	Gonzalez		Trammell	
17	Darr			
18				
19				

Figure 15-14: Using a custom worksheet function in an array formula

Summary

This chapter provided many examples of useful array formulas. You can use these formulas as is, or adapt them to your needs. It also presented a custom worksheet function that returns an array.

The next chapter presents intentional circular references.

Part V

Miscellaneous Formula Techniques

CHAPTER 16

Intentional Circular References

CHAPTER 17

Charting Techniques

CHAPTER 18

Pivot Tables

CHAPTER 19

Conditional Formatting and Data Validation

CHAPTER 20

Creating Megaformulas

CHAPTER 21

Tools and Methods for Debugging Formulas

Chapter 16

Intentional Circular References

IN THIS CHAPTER

- ◆ General information regarding how Excel handles circular references
- ◆ Why you might want to use an intentional circular reference
- ◆ How Excel determines calculation and iteration settings
- ◆ Examples of formulas that use intentional circular references
- ◆ Potential problems when using intentional circular references

WHEN MOST SPREADSHEET users hear the term *circular reference*, they immediately think of an error condition. Generally, a circular reference represents an accident – something that you need to correct. Sometimes, however, a circular reference can be a good thing. This chapter presents some examples that demonstrate intentional circular references.

What Are Circular References?

When entering formulas in a worksheet, you occasionally may see a message from Excel, such as the one shown in Figure 16-1. This demonstrates Excel's way of telling you that the formula you just entered will result in a *circular reference*. A circular reference occurs when a formula refers to its own cell, either directly or indirectly. For example, you create a circular reference if you enter the following formula into cell A10 because the formula refers to the cell that contains the formula:

```
=SUM(A1:A10)
```

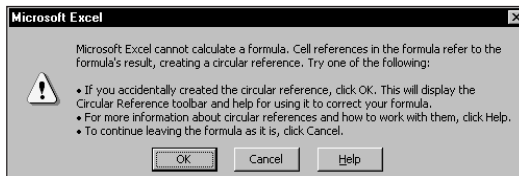


Figure 16-1: Excel's way of telling you that your formula contains a circular reference

Every time the formula in A10 is calculated, it must be recalculated because A10 has changed. In theory, the calculation could continue forever while the value in cell A10 tried to reach infinity.

Correcting an Accidental Circular Reference

When you see the circular reference message after entering a formula, Excel gives you three options:

- ◆ Click OK to attempt to locate the circular reference (Excel's Circular Reference toolbar displays). This also has the annoying side effect of displaying a help screen whether you need it or not.
- ◆ Click Cancel to enter the formula as is.
- ◆ Click Help to read about circular references in the online help.

Most circular reference errors are caused by simple typographical errors or incorrect range specifications. For example, when creating a SUM formula in cell A10, you might accidentally specify an argument of A1:A10 instead of A1:A9.

If you know the source of the problem, click Cancel. Excel displays a message in the status bar to remind you that a circular reference exists. In this case, the message reads "Circular: A10." If you activate a different workbook or worksheet, the message simply displays "Circular" (without the cell reference). You can then edit the formula and fix the problem.

If you get the circular message error, but you don't know what formula caused the problem, you can click OK. When you do so, Excel displays the Help topic on circular references and the Circular Reference toolbar (see Figure 16-2). On the Circular Reference toolbar, click the first cell in the Navigate Circular Reference drop-down list box, and then examine the cell's formula. If you cannot determine whether that cell caused the circular reference, click the next cell in the Navigate Circular Reference box. Continue to review the formulas until the status bar no longer displays Circular.

About Circular References

For a practical, real-life demonstration of a circular reference, refer to the sidebar, "More about Circular References," later in this chapter.

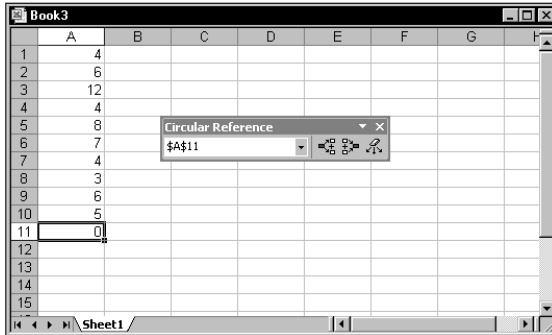


Figure 16-2: The Circular Reference toolbar



Excel won't display its Circular Reference dialog box if you have the Iteration setting turned on. You can check this in the Options dialog box (in the Calculation tab). I discuss more about this setting later.

Understanding Indirect Circular References

Usually, a circular reference appears quite obvious and, therefore, easy to identify and correct. Sometimes, however, circular references are indirect. In other words, one formula may refer to another formula that refers to a formula that refers back to the original formula. In some cases, you need to conduct a bit of detective work to figure out the problem.



For more information about tracking down a circular reference, refer to Chapter 21.

Intentional Circular References

As mentioned previously, you can use a circular reference to your advantage in some situations. A circular reference, if set up properly, can serve as the functional equivalent of a Do-Loop construct used in a programming language such as VBA. An intentional circular reference introduces recursion or iteration into a problem. Each intermediate “answer” from a circular reference calculation functions in the subsequent calculation. Eventually, the solution converges to the final value.

By default, Excel does not permit iterative calculations. You must explicitly tell Excel that you want it to perform iterative calculations in your workbook. You do this on the Calculation tab of the Options dialog box (see Figure 16-3).

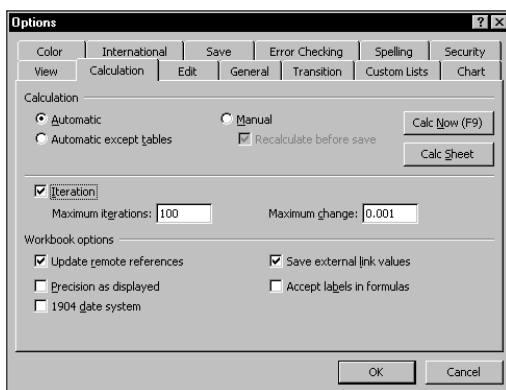


Figure 16-3: To calculate a circular reference, you must check the Iteration check box.

Figure 16-4 shows a simple example of a worksheet that uses an intentional circular reference. A company has a policy of contributing five percent of its net profit to charity. The contribution itself, however, is considered an expense and is therefore subtracted from the net profit figure. This produces a circular reference.

	A	B	C	D	E	F	G
1	Gross Income	250,000					
2	Expenses	189,500					
3	Contributions	2,881		5% of Net Profit			
4	Net Profit	57,619		<i>Gross Income - Expenses - Contributions</i>			
5							
6							
7							

Figure 16-4: The company also deducts the five percent contribution of net profits as an expense, creating an intentional circular reference.



You cannot resolve the circular reference unless you turn on the Iteration setting.

The text in column A corresponds to the named cells in column B, and cell C3 is named *Pct*. The *Contributions* cell (B3) contains the following formula:

```
=Pct*Net_Profit
```

The *Net_Profit* cell (B4) contains the following formula:

```
=Gross_Income-Expenses-Contributions
```

These formulas produce a resolvable circular reference. Excel keeps calculating until the formula results converge on a solution.



A reader of the first edition of this book pointed out another way to approach this problem without using a circular reference. Use the following formula to calculate the *Net_Profit* cell:

```
=(Gross_Income-Expenses)/(1+Pct)
```

Then calculate the *Contributions* cell using this formula:

```
=Pct*Net_Profit
```



You can access the workbook shown in Figure 16-4 on the companion CD-ROM. For your convenience, the worksheet includes a button that, when clicked, displays the Calculation tab of the Options dialog box. This makes it easy to experiment with various iteration settings. In addition, the CD-ROM contains a file that demonstrates how to perform this calculation without using a circular reference.

The Calculation tab of the Options dialog box contains three controls relevant to circular references:

- ◆ Iteration check box: If unchecked, Excel does not perform iterative calculations, and Excel displays a warning dialog box if you create a formula that has a circular reference. When creating an intentional circular reference, you must check this check box.

- ◆ **Maximum iterations:** Determines the maximum number of iterations that Excel will perform. This value cannot exceed 32,767.
- ◆ **Maximum change:** Determines when iteration stops. For example, if this setting is .01, iterations stops when a calculation produces a result that differs by less than 1 percent of the previous value.



Calculation continues until Excel reaches the number of iterations specified in the Maximum iterations box, or until a recalculation changes all cells by less than the amount you set in the Maximum change box (whichever is reached first). Depending on your application, you may need to adjust the settings in the Maximum iterations field or the Maximum change field. For a more accurate solution, make the Maximum change field smaller. If the result doesn't converge after 100 iterations, you can increase the Maximum iterations field.

To get a feel for how this works, open the example workbook presented in the previous section. Then:

1. Access the Calculation tab in the Options dialog box and make sure the Iteration check box is checked.
2. Set the Maximum iterations setting to 1.
3. Set the Maximum change setting to .001.
4. Enter a different value into the *Gross_Income* cell (cell B1).
5. Press F9 to calculate the sheet.

Because the Maximum iteration setting is 1, pressing F9 performs just one iteration. You'll find that the *Contributions* cell has not converged. Press F9 a few more times, and you'll see the result converge on the solution. When the solution is found, pressing F9 has no noticeable effect. If the Maximum iterations setting reflects a large value, the solution appears almost immediately (unless it involves some slow calculations).

How Excel Determines Calculation and Iteration Settings

You should understand that all open workbooks use the same calculation and iteration settings. For example, if you have two workbooks open, you cannot have one

of them set to automatic calculation and the other set to manual calculation. Although you can save a workbook with particular settings (for example, manual calculation with no iterations), those settings can change if you open another workbook.

Excel follows these general rules to determine which calculation and iteration settings to use:

- ◆ The first workbook opened uses the calculation mode saved with that workbook. If you open other workbooks, they use the same calculation mode.

For example, suppose you have two workbooks: Book1 and Book2. Book1 has its Iteration setting turned off (the default setting), and Book2 (which uses intentional circular references) has its Iteration setting turned on. If you open Book1 and then Book2, both workbooks will have the iteration setting turned off. If you open Book2 and then Book1, both workbooks will have their iteration setting turned on.

- ◆ Changing the calculation mode for one workbook changes the mode for all workbooks.

If you have both Book1 and Book2 open, changing the calculation mode or Iteration setting of either workbook affects both workbooks.

- ◆ All worksheets in a workbook use the same mode of calculation.
- ◆ If you have all workbooks closed and you create a new workbook, the new workbook uses the same calculation mode as the last closed workbook. One exception: if you create the workbook from a template. If so, the workbook uses the calculation mode specified in the template.
- ◆ If the mode of calculation in a workbook changes, and you save the file, the current mode of calculation saves with the workbook.

Circular Reference Examples

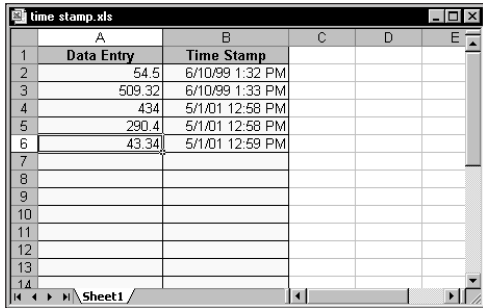
Following are a few more examples of using intentional circular references. They demonstrate creating circular references for time stamping a cell, calculating an all-time-high value, solving a recursive equation, and solving simultaneous equations.



For these examples to work properly, you must have the Iteration setting in effect. Select Tools → Options, and click the Calculation tab. Make sure the Iteration check box is checked.

Time Stamping a Cell Entry

Figure 16-5 shows a worksheet designed such that entries in column A are “time stamped” in column B. The formulas in column B monitor the corresponding cell in column A. When you insert an entry in column A, the formula enters the current date and time.



	A	B	C	D	E
1	Data Entry	Time Stamp			
2	54.5	6/10/99 1:32 PM			
3	509.32	6/10/99 1:33 PM			
4	434	5/1/01 12:58 PM			
5	290.4	5/1/01 12:58 PM			
6	43.34	5/1/01 12:59 PM			
7					
8					
9					
10					
11					
12					
13					
14					

Figure 16-5: Using circular reference formulas to time stamp entries in column A



The workbook shown in Figure 16-5 also appears on the companion CD-ROM.

The formula in cell B2, which is copied down to other cells in column B, is:

```
=IF(ISBLANK(A2), "", IF(B2="", NOW(), B2))
```

This formula uses an IF function to check cell A2. If the cell is empty, the formula returns an empty string. If A2 is not empty, the formula checks the value in cell B2 (that is, a self-reference). If B2 is empty, the formula returns the date and time. Using the second IF statement ensures that the NOW function does not recalculate.

Calculating an All-Time-High Value

Figure 16-6 shows a worksheet that displays the sales made by sales representatives. This sheet updates every month: New sales figures replace the values in column B.

	A	B	C	D	E	F
1	April Sales		123,323 <-- All-Time High!			
2						
3	Sales Rep	Amount Sold				
4	Jackson	49,032				
5	Smith	56,892				
6	Lopez	32,445				
7	Peterson	55,678				
8	Rayez	59,898				
9						
10						

Figure 16-6: Using a circular reference formula to keep track of the highest value ever entered in column B

The formula in cell C1 keeps track of the all-time-high sales – the largest value *ever* entered into column B. This formula, which uses a circular reference, appears as follows:

```
=MAX(B:B,C1)
```

The formula uses the MAX function to return the maximum value in column B, or in cell C1. In this example, the formula displays 98,223. This reflects a value from a previous month (in other words, a value not currently in column B).



The companion CD-ROM contains the workbook shown in Figure 16-6.

Generating Unique Random Integers

You can take advantage of a circular reference to generate unique random integers in a range. The worksheet in Figure 16-7 generates 15 random integers between 1 and 30 in column A. The integers are generated such that they produce unique numbers (that is, not duplicated). You may want to use this technique to generate random lottery number picks.

Column B contains formulas that count the number of times a particular number appears in the range A1:A15. For example, the formula in cell B1 follows. This formula displays the number of times the value in cell A1 appears in the range A1:A15:

```
=COUNTIF($A$1:$A$15,A1)
```

	A	B	C	D	E	F
1	5	1		SOLUTION FOUND		
2	12	1				
3	19	1				
4	25	1				
5	14	1				
6	18	1				
7	23	1				
8	8	1				
9	17	1				
10	9	1				
11	4	1				
12	21	1				
13	16	1				
14	22	1				
15	29	1				
16						

Figure 16-7: Using circular reference formulas to generate unique random integers in column A

Each formula in column A contains a circular reference. The formula examines the sum of the cells in column B. If this sum does not equal 15, a new random integer generates. When the sum of the cells in column B equals 15, the values in column A are all unique. The formula in cell A1 appears like this:

```
=IF(SUM($B$1:$B$15)<>15,INT(RAND()*30+1),A1)
```

Cell D1, which follows, contains a formula that displays the status. If the sum of the cells in column B does not equal 15, the formula displays the text *CALC AGAIN* (press F9 to perform more iterations). When column B contains all 1s, the formula displays *SOLUTION FOUND*.

```
=IF(SUM(B1:B15)<>15,"CALC AGAIN","SOLUTION FOUND")
```

To generate a new set of random integers, select any cell in column B. Then press F2 to edit the cell, and press Enter to reenter it. The number of calculations required depends on:

- ◆ The Iteration setting on the Calculation tab of the Options dialog box. If you specify a higher number of iterations, you have a better chance of finding 15 unique values.
- ◆ The number of values requested, compared to the number of possible values. This example seeks 15 unique integers from a pool of 30. Fewer calculations are required if, for example, you request 15 unique values from a pool of 100.

Solving a Recursive Equation

A recursive equation refers to an equation in which a variable appears on both sides of the equal sign. The following equations represent examples of recursive equations:

$$\begin{aligned}x &= 1/(x+1) \\x &= \text{COS}(x) \\x &= \text{SQRT}(X+5) \\x &= 2^{(1/x)} \\x &= 5 + (1/x)\end{aligned}$$

To solve a recursive equation, make sure that you turn on the Iteration setting. Then convert the equation into a self-referencing formula. To solve the first equation, enter the following formula into cell A1:

$$=1/(A1+1)$$

The formula converges at 0.618033988749895, the value of x that satisfies the equation.

Sometimes, this technique doesn't work. For example, consider the following recursive equation:

$$x = 5 + (1/x)$$

If you enter the formula that follows into cell A1, you'll find that it returns a #DIV/0! error because the iterations begin with 0 (and dividing by 0 results in an error):

$$=5+(1/A1)$$

To solve this type of equation, you need to use two cells. The following step-by-step instructions demonstrate:

1. Enter any non-zero value in cell A1.
2. Enter the following formula in cell A2:

$$=5+(1/A1)$$

3. Enter the following formula in cell A1:

$$=A2$$

Both cells A1 and A2 display 5.19258235429625, the value of x that satisfies the equation. Note that, in Step 1, entering a non-zero value essentially provides a non-zero *seed* for the recursion. After you replace this value with the formula (in

Step 3), the initial value in cell A1 still operates as the starting value for the formula in cell A2.



The seed cell must reside to the left or above the formula because of the way Excel performs calculations.

Figure 16-8 shows a worksheet that calculates several recursive equations. Note that the equations in rows 5 and 6 require a seed value. The formulas in column E use the values in column C to provide a check of the results. For example, the formula in cell E2 is:

=1/(C2+1)

	A	B	C	D	E	F
1	Equation	Seed	Circular Ref Formula		Check	
2	$x = 1/(x+1)$		0.618033989		0.61803399	
3	$x = \cos(x)$		0.739085133		0.73908513	
4	$x = \text{SQRT}(x+5)$		2.791287847		2.79128785	
5	$x = 2^{(1/x)}$	1.559610469	1.559610469		1.55961047	
6	$x = 5 + (1/x)$	5.192582404	5.192582404		5.1925824	
7						
8						

Figure 16-8: This workbook uses circular references to calculate several recursive equations.



You can access the workbook shown in Figure 16-8 on the companion CD-ROM.

Solving Simultaneous Equations Using a Circular Reference

In some cases, you can use circular references to solve simultaneous equations. Consider the two simultaneous equations listed here:

$$\begin{aligned} 3x + 4y &= 8 \\ 3x + 8y &= 20 \end{aligned}$$

You need to find the value of x and the value of y that satisfies both equations. First, rewrite the equations to express them in terms of x and y . The following represents the first equation, expressed in terms of x :

$$x = (8 - 4y)/3$$

The following equation represents the second equation, expressed in terms of y :

$$y = (20 - 3x)/8$$

As shown in Figure 16-9, cell B5 is named X and cell B6 is named Y . The formulas in these cells mirror the previous equations. The formula in B5 (X) appears like this:

$$=(8-(4*Y))/3$$

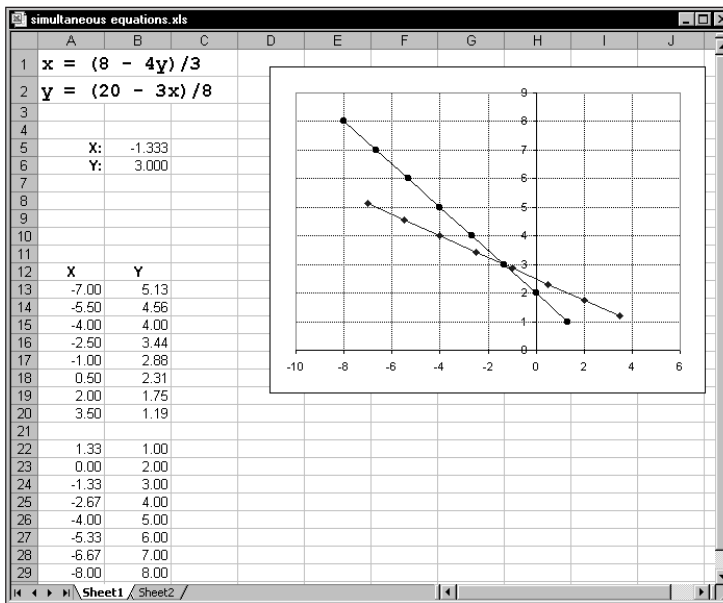


Figure 16-9: This worksheet solves two simultaneous equations.

The formula in cell B6 (Y) is:

$$=(20-(3*X))/8$$

The figure also shows a chart that plots the two equations. The intersection of the two lines represents the values of X and Y that solve the equations.

More about Circular References

For a practical, real-life demonstration of a circular reference, refer to the sidebar, "About Circular References," earlier in this chapter.

Note the circular reference. The *X* cell refers to the *Y* cell, and the *Y* cell refers to the *X* cell. These cells converge to display the solution:

$$X = -1.333$$

$$Y = 3.000$$

Using intentional circular references to solve simultaneous equations is more of an interesting demonstration than a practical approach. You'll find that some iterative calculations never converge. In other words, successive recalculations will never *hone in* on a solution. For example, consider the simultaneous equations that follow. A solution *does* exist, but you cannot use circular references to find it.

$$x = 4 - y/2$$

$$y = 3 + 2x$$



The use of matrices presents the best approach for solving simultaneous equations with Excel. See Chapter 10 for examples.



The companion CD-ROM contains a workbook with two sets of simultaneous equations. You can solve one set by using intentional circular references; you cannot solve the other set using this technique.

Potential Problems with Intentional Circular References

Although intentional circular references can be useful, using this feature has some potential problems. Perhaps the best advice is to use this feature with caution, and make sure you understand how it works.

To take advantage of an intentional circular reference, you must have the Iteration setting in effect. When the Iteration setting is in effect, Excel does not warn you of circular references. Therefore, you run the risk of creating an *accidental* circular reference without even knowing about it.

The number of iterations specified in the Maximum iteration field applies to all formulas in the workbook, not just those that use circular references. If your workbook contains many complex formulas, these additional iterations can slow things down considerably. Therefore, when you use intentional circular references, keep your worksheets very simple.

You may need to distribute a workbook that uses intentional circular references to other users. If Excel's Iteration setting is not active when you open the workbook, Excel displays the circular reference error message, which probably confuses all but the most sophisticated users.

Summary

This chapter provided an overview of how Excel handles circular references. Although most circular references indicate an error, there exist some benefits to writing formulas that use intentional circular references. To take advantage of a circular reference, you must have the Iteration setting in effect.

The next chapter demonstrates how formulas can expand your chart-making capabilities.

Chapter 17

Charting Techniques

IN THIS CHAPTER

- ◆ Creating charts from any number of worksheets or different workbooks
- ◆ Plotting functions with one and two variables
- ◆ Creating awesome designs with formulas
- ◆ Working with linear and nonlinear trendlines
- ◆ Useful charting tricks for working with charts

EXCEL SUPPORTS MORE THAN 100 different chart types, and you have almost complete control over nearly every aspect of each chart. This chapter, which assumes that you're familiar with Excel's charting feature, demonstrates some useful charting techniques – most of which involve formulas.

Representing Data in Charts

Basically, a *chart* presents a table of numbers visually. Displaying data in a well-conceived chart can make the data more understandable. Because a chart presents a picture, charts are particularly useful for understanding a lengthy series of numbers and their interrelationships. Making a chart can help you to spot trends and patterns that you otherwise could not identify when examining a range of numbers.

You create charts from numbers that appear in a worksheet. You can enter these numbers directly, or you can derive them as the result of formulas. Normally, the data used by a chart resides in a single worksheet, within one file, but that's not a strict requirement. A single chart can use data from any number of worksheets, or even from different workbooks.

Understanding the SERIES Formula

A chart consists of one or more data series, and each data series appears as a line, column, bar, and so on. A chart has a SERIES formula for each series in the chart. When you select a data series in a chart, its SERIES formula appears in the formula

bar. This is not a “real” formula. In other words, you can’t use it in a cell and you can’t use worksheet functions within the SERIES formula. You can, however, edit the arguments in the SERIES formula. A SERIES formula has the following syntax:

```
=SERIES(name, category_labels, values, order)
```

The arguments you can use in the SERIES formula include:

- ◆ *name*: (Optional) The name used in the legend. If the chart has only one series, the name argument is used as the title.
- ◆ *category_labels*: (Optional) The range that contains the labels for the category axis. If omitted, Excel uses consecutive integers beginning with 1.
- ◆ *values*: (Required) The range that contains the values.
- ◆ *order*: (Required) An integer that specifies the plotting order of the series (relevant only if the chart has more than one series).

Range references in a SERIES formula are always absolute, and they always include the sheet name. For example:

```
=SERIES(Sheet1!$B$1,,Sheet1!$B$2:$B$7,1)
```

A range reference can consist of a noncontiguous range. If so, each range is separated by a comma and the argument is enclosed in parentheses. In the following SERIES formula, the values range consists of B2:B3 and B5:B7:

```
=SERIES(,,(Sheet1!$B$2:$B$3,Sheet1!$B$5:$B$7),1)
```

Although a SERIES formula can refer to data in other worksheets, the data for a series must reside on a single sheet. The following SERIES formula, for example, is not valid because the data series references two different worksheets:

```
=SERIES(,,(Sheet1!$B$2,Sheet2!$B$2),1)
```

USING NAMES IN A SERIES FORMULA

You can substitute range names for the range references in a SERIES formula. When you do so, Excel changes the reference in the SERIES formula to include the workbook name. For example, the SERIES formula shown here uses a range named *MyData* (located in a workbook named *budget.xls*). Excel added the workbook name and exclamation point.

```
=SERIES(Sheet1!$B$1,,budget.xls!MyData,1)
```

Chart-Making Tips

Here I present a number of chart-making tips that you might find helpful:

- ◆ To create a chart with a single keystroke, select the data you want to chart and press F11. The result is a new chart sheet that contains a chart of the default chart type.
- ◆ You can size the chart in a chart sheet according to the window size by using the View → Sized with Window command. When you enable this setting, the chart adjusts itself when you resize the workbook window (it always fits perfectly in the window). In this mode, the chart that you're working on may or may not correspond to how it looks when printed.
- ◆ If you have many charts of the same type to create, changing the default chart format to the chart type with which you're working is much more efficient than separately formatting each chart. Then you can create all of your charts without having to select the chart type. To change the default chart type, select Chart → Chart Type and choose the new default chart type. Then click the Set as default chart type button. You can also save it as a user-defined custom chart type so that you can reuse it later. To do so, click the Custom Types tab and click the Add button.
- ◆ To print an embedded chart on a separate page, select the chart and choose File → Print (or click the Print button). Excel prints the chart on a page by itself and does *not* print the worksheet.
- ◆ If you don't want a particular embedded chart to appear on your printout, right-click the chart and choose Format Chart Area from the shortcut menu. Click the Properties tab in the Format Chart Area dialog box and remove the check mark from the Print object check box.
- ◆ Sometimes, using a mouse to select a particular chart element is tricky. You may find it easier to use the keyboard to select a chart element. When a chart is activated, press the up arrow or down arrow to cycle through all parts in the chart. When a data series is selected, press the right arrow or left arrow to select individual points in the series.
- ◆ When you select a chart element, you'll find that many of the toolbar buttons that you normally use for worksheet formatting also work with the selected chart element. For example, if you select the chart's Plot Area, you can change its color by using the Fill Color tool on the Formatting toolbar. If you select an element that contains text, you can use the Font Color tool to change the color of the text.

Continued

Chart-Making Tips (Continued)

- ◆ Prior to Excel 97, clicking an embedded chart selected the chart object. You could then adjust its properties. To activate the chart, you actually had to double-click it. Beginning with Excel 97, clicking an embedded chart activates the chart contained inside the chart object. You can adjust the chart object's properties by using the Properties tab of the Format Chart dialog box. To select the chart object itself, press Ctrl while you click the chart. You may want to select the chart object to change its name by using the Name box.
- ◆ You can delete all data series from a chart. If you do so, the chart appears empty. It retains its settings, however. Therefore, you can add a data series to an empty chart and it again looks like a chart.
- ◆ For more control over positioning your chart, press Ctrl while you click the chart. Then use the arrow keys to move the chart one pixel at a time.
- ◆ To create a line that continues through a point that has no information, type the formula `=NA()` in the blank cells in your range.

Using names in a series formula provides a significant advantage: If you change the range reference for the name, the chart automatically reflects the new data. In the preceding SERIES formula, for example, assume the range named *MyData* refers to A1:A20. The chart displays the 20 values in that range. You can then use the Insert → Name → Define command to redefine *MyData* as a different range, say A1:A30. The chart then displays the 30 data points defined by *MyData* (no chart editing is necessary).

As I noted previously, a SERIES formula cannot use worksheet functions. You *can*, however, create named formulas (which use functions) and use these named formulas in your SERIES formula. As you see later in this chapter, this technique enables you to perform charting tricks that seem impossible.

UNLINKING A CHART SERIES FROM ITS DATA RANGE

Normally, an Excel chart uses data stored in a range. Change the data in the range, and the chart updates automatically. In some cases, you may want to “unlink” the chart from its data ranges and produce a *static chart*—a chart that never changes. For example, if you plot data generated by various what-if scenarios, you may want to save a chart that represents some baseline so you can compare it with other scenarios. There are two ways to create such a chart:

- ◆ *Paste it as a picture*: Activate the chart and then press Shift and choose Edit → Copy Picture (the Paste Picture command is available only if you press Shift when you select the Edit menu). Then press the Shift key and select Edit → Paste Picture. The result is a picture of the copied chart.

- ◆ *Convert the range references to arrays:* Click a chart series and then click the formula bar to activate the SERIES formula. Press F9 to convert the ranges to arrays. Repeat this for each series in the chart. This technique (as opposed to creating a picture) enables you to continue to edit the chart. This technique will not work for large amounts of data because there is a limit to the length of a SERIES formula.

Creating Links to Cells

You can add cell links to various elements of a chart. Adding cell links can make your charts more dynamic. You can set dynamic links for chart titles, data labels, additional descriptive text, and pictures.

ADDING TITLE LINKS

The labels in a chart (Chart Title, Category Axis Title, and Value Axis Title) are normally not linked to any cell. In other words, they contain static text that changes only when you edit them manually. You can, however, create a link so a title refers to a worksheet cell.

To create a linked title, first make sure the chart contains the chart element title that you want. You can use the Chart Options dialog box to add titles to a chart that doesn't already have them (select Chart → Chart Options to display this dialog box). Next, select the title and click in the formula bar. Type an equal sign and then click the cell that contains the title text. The result is a formula that contains the sheet reference and the cell reference as an absolute reference (for example, =Sheet1!\$A\$1). Press Enter to attach the formula to the chart title. Figure 17-1 shows a chart in which the Chart Title is linked to cell A1.

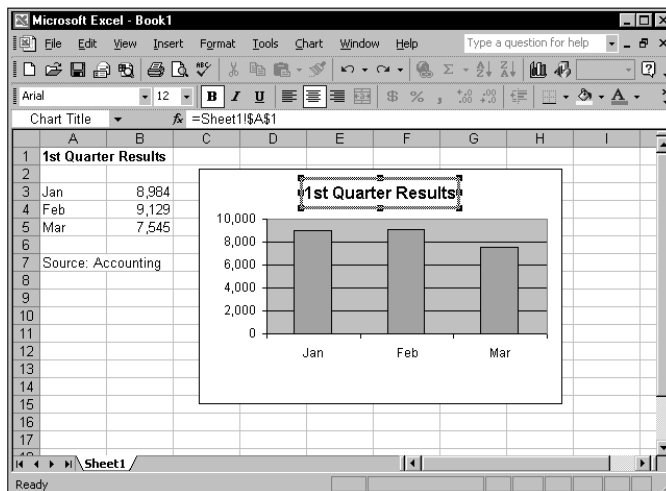


Figure 17-1: The Chart Title linked to cell A1

ADDING LINKS TO DATA LABELS

You probably know that Excel enables you to label each data point in a chart. You do this on the Data Labels tab in the Format Data Series dialog box. Unfortunately, this feature isn't very flexible. For example, you can't specify a range that contains the labels. You can, however, edit individual data labels. To do so, click once on any data label to select them all, then click a second time to select the single data label. Once a single data label is selected, you can add any text you like. Or, you can specify a link to a cell by clicking the formula bar and entering a reference formula (such as =Sheet1!\$A\$1).



The Power Utility Pak includes a handy utility that makes it easy to add data labels to your charts by specifying a worksheet range — an often-requested feature that Microsoft refuses to add. A trial version of the Power Utility Pak is available on the companion CD-ROM.

ADDING TEXT LINKS

You might want your chart to display some other text (such as a descriptive note) that's stored in a cell. Doing so is easy. First, activate the chart. Then click in the formula bar, type an equal sign, and click the cell that contains the text. Press Enter. Excel creates a Text Box in the center of your chart (see Figure 17-2). You can drag this Text Box to its desired location and apply any type of formatting you like.

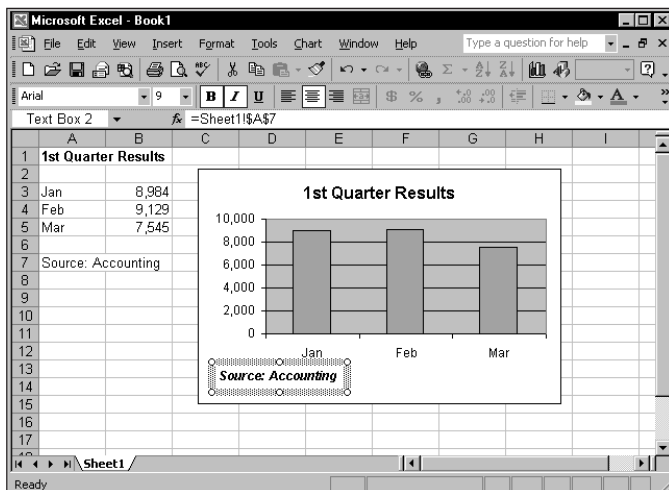


Figure 17-2: A Text Box linked to a cell

To add an unlinked Text Box, just select the chart, type the text in the formula bar, and press Enter.



Adding objects, such as Text Boxes, to a chart can be very tricky. For example, you may find that subsequent operations to the chart (such as removing axes or the legend) may cause the object to disappear from view. This is a long-time bug that Microsoft refuses to address. For best results, add the Text Box after you've made all other modifications to the chart.

ADDING PICTURE LINKS

Excel has a feature that enables you to display a data table inside of a chart. You can select this option in Step 3 of the Chart Wizard, or add a data table to an existing chart by using the Data Table tab of the Chart Options dialog box. The data table option displays a table that shows the values used in a chart. This can be a handy feature, but it's not very flexible. For example, you have limited formatting options, and you have no control over the position of the data table (it always appears below the chart). A linked picture of a range presents an alternative to the data table (see Figure 17-3 for an example).

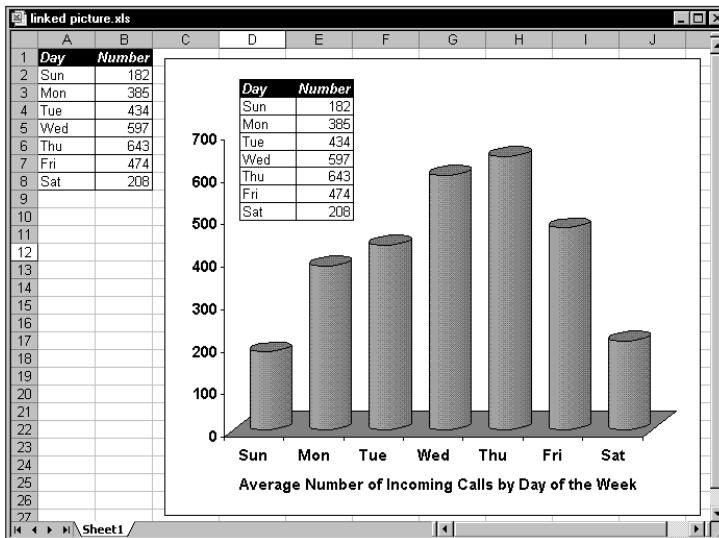


Figure 17-3: This chart contains a linked picture of the A1:B8 range.



A workbook that demonstrates the use of a linked picture in a chart is available on the companion CD-ROM.

To create a linked picture in a chart, first create the chart as you normally would. Then perform the following steps:

1. Select the range that you would like to include in the chart.
2. Select Edit → Copy.
3. Activate the chart.
4. Press Shift, and then select Edit → Paste Picture. This pastes an *unlinked* picture of the range.
5. To create the link, select the picture and then type a reference to the range in the formula bar. You can do this easily by typing an equal sign and then reselecting the range.

The picture now contains a live link to the range. If you change the values or cell formatting, the changes will be reflected in the linked picture. This technique also works with chart sheets.

Charting Progress toward a Goal

You're probably familiar with a "thermometer" type display that shows the percentage of a task that's completed. It's relatively easy to create such a display in Excel. The trick involves creating a chart that uses a single cell (which holds a percentage value) as a data series.

Figure 17-4 shows a worksheet set up to track daily progress toward a goal: 1,000 new customers in a 15-day period. Cell B18 contains the goal value, and cell B19 contains a simple sum formula:

```
=SUM(B2:B16)
```

Cell B21 contains a formula that calculates the percent of goal:

```
=B19/B18
```

As you enter new data in column B, the formulas display the current results.

To create the chart, select cell B21 and click the Chart Wizard button. Notice the blank row before cell B21. Without this blank row, Excel uses the entire data block for the chart, not just the single cell. Since B21 is isolated from the other data, the Chart Wizard uses only the single cell. In Step 1 of the Chart Wizard dialog, specify a Column chart with the first subtype (Clustered Column). Click Next twice and

make some additional adjustments on the Step 3 page: add a Chart Title (Title tab), remove Category (x) axis (Axes tab), remove the legend (Legend tab), and specify Show value (Data Labels tab). Click Finish to create the chart.

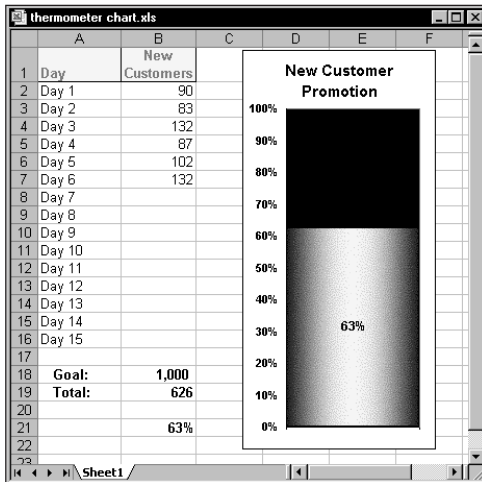


Figure 17-4: This chart displays progress toward a goal.

Then make some additional customizations. Double-click the column to display the Format Data Series dialog. Click the Options tab, and set the Gap width to 0 (this makes the column occupy the entire width of the plot area). You also may want to change the pattern used in the column. Do this in the Patterns tab. The example uses a gradient fill effect. Next, double-click the vertical axis to bring up the Format Axis dialog. In the Scale tab, set the Minimum to 0 and the Maximum to 1.

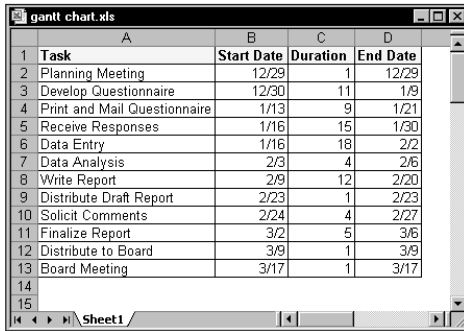
You can make other cosmetic changes as you like. For example, you may want to change the chart's width to make it look more like a thermometer, as well as adjust fonts, colors, and so on.



The workbook containing the progress chart also appears on the companion CD-ROM.

Creating a Gantt Chart

Gantt charts represent the time required to perform each task in a project. Figure 17-5 shows data used to create the simple Gantt chart shown in Figure 17-6. Creating a Gantt chart isn't difficult when using Excel, but it does require some set-up work.



	A	B	C	D
1	Task	Start Date	Duration	End Date
2	Planning Meeting	12/29	1	12/29
3	Develop Questionnaire	12/30	11	1/9
4	Print and Mail Questionnaire	1/13	9	1/21
5	Receive Responses	1/16	15	1/30
6	Data Entry	1/16	18	2/2
7	Data Analysis	2/3	4	2/6
8	Write Report	2/9	12	2/20
9	Distribute Draft Report	2/23	1	2/23
10	Solicit Comments	2/24	4	2/27
11	Finalize Report	3/2	5	3/6
12	Distribute to Board	3/9	1	3/9
13	Board Meeting	3/17	1	3/17
14				
15				

Figure 17-5: Data used in the Gantt chart

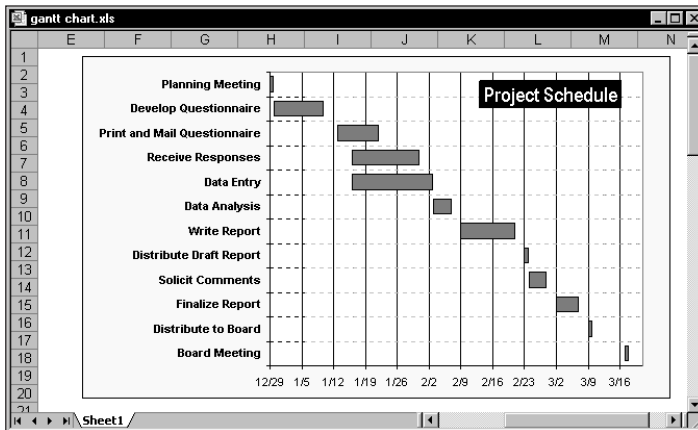


Figure 17-6: You can create a Gantt chart from a bar chart.



You can access a workbook that demonstrates setting up a Gantt chart on the companion CD-ROM.

Follow these steps to create this chart:

1. Enter the data as shown in Figure 17-5. The formula in cell D2, which was copied to the rows below it, is:

$$=B2+C2-1$$
2. Use the Chart Wizard to create a stacked bar chart from the range A2:C13. Use the second subtype, labeled *Stacked Bar*.

3. In Step 2 of the Chart Wizard, select the Columns option. Also, notice that Excel incorrectly uses the first two columns as the Category axis labels.
4. In Step 2 of the Chart Wizard, click the Series tab and add a new data series. Then set the chart's series to the following:
Series 1: B2:B13
Series 2: C2:C13
Category (x) axis labels: A2:A13
5. In Step 3 of the Chart Wizard, remove the legend and then click Finish to create an embedded chart.
6. Adjust the height of the chart so that all the axis labels are visible. You can also accomplish this by using a smaller font size.
7. Access the Format Axis dialog box for the horizontal axis. Adjust the horizontal axis Minimum and Maximum scale values to correspond to the earliest and latest dates in the data (note that you can enter a date into the Minimum or Maximum edit box). You also may want to change the date format for the axis labels.
8. Access the Format Axis dialog box for the vertical axis, and click the Scale tab. Select the option labeled Categories in Reverse Order, and also set the option labeled Value (Y) Axis Crosses at Maximum.
9. Select the first data series and access the Format Data Series dialog box. On the Patterns tab, set Border to None and Area to None. This makes the first data series invisible.
10. Apply other formatting as desired.

Creating a Comparative Histogram

With a bit of creativity, you can create charts that you thought impossible with Excel. For example, Figure 17-7 shows a comparative histogram chart. Such a chart, sometimes known as a *population pyramid*, often displays population data.

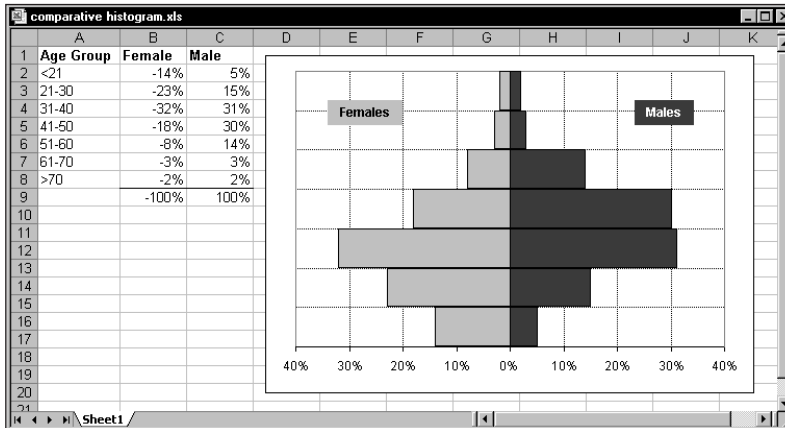


Figure 17-7: Producing this comparative histogram chart requires a few tricks.



The companion CD-ROM contains a workbook that demonstrates a comparative histogram chart.

Follow these steps to create the chart:

1. Enter the data as shown in Figure 17-7. Notice that the values for females are entered as negative numbers.
2. Select A1:C8 and create a 2D bar chart. Use the subtype labeled *Clustered Bar*.
3. Apply the following custom number format to the horizontal axis:
0%;0%;0%
This custom format eliminates the negative signs in the percentages.
4. Select the vertical axis and access the Format Axis dialog box. Click the Patterns tab and remove all tick marks. Set the Tick mark labels option to Low. This keeps the axis in the center of the chart, but displays the axis labels at the left side.
5. Select either of the data series and then access the Format Data Series dialog box. Click the Options tab and set the Overlap to 100 and the Gap width to 0.
6. Delete the legend.
7. Add two Text Boxes to the chart (Females and Males), to substitute for the legend.
8. Apply other formatting as desired.

Creating a Box Plot

A box plot (sometimes known as a quartile plot) is often used to summarize data. Figure 17-8 shows a box plot created for four groups of data. The raw data appears in columns A through D. The range G2:J7, used in the chart, contains formulas that summarize the data. Table 17-1 shows the formulas in column G (which were copied to the three columns to the right).

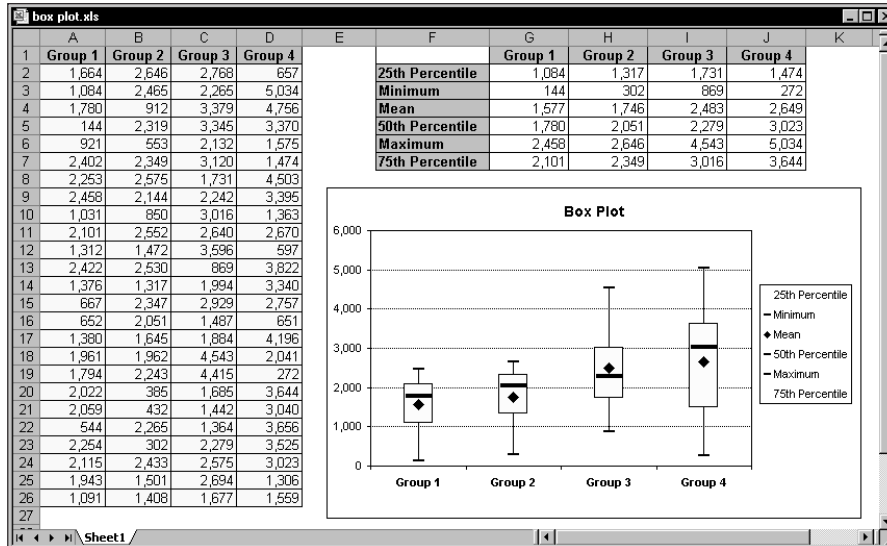


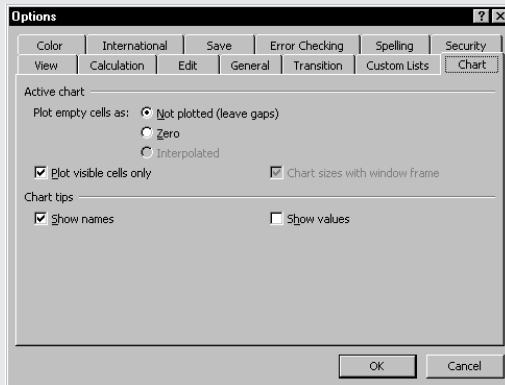
Figure 17-8: This box plot summarizes the data in columns A through D.

TABLE 17-1 FORMULAS USED TO CREATE A BOX PLOT

Cell	Calculation	Formula
G2	25th Percentile	=QUARTILE(A2:A26,1)
G3	Minimum	=MIN(A2:A26)
G4	Mean	=AVERAGE(A2:A26)
G5	50th Percentile	=QUARTILE(A2:A26,2)
G6	Maximum	=MAX(A2:A26)
G7	75th Percentile	=QUARTILE(A2:A26,3)

Handling Missing Data

Sometimes, data that you chart may lack one or more data points. Excel offers several ways to handle the missing data. You don't specify these options in the Format Data Series dialog box or even in the Chart Options dialog box. Rather, you must select the chart, choose Tools → Options, and then click the Chart tab, shown here.



This setting applies only to the active chart. You must have an active chart when you open the Options dialog box. Otherwise, the option is grayed. This is an excellent example of a setting that shows up in an unexpected dialog box.

The options that you set apply to the entire active chart, and you can't set a different option for different series in the same chart. The following are the options in the Chart panel for the active chart:

- ◆ Not plotted (leave gaps): Missing data gets ignored, causing the data series to have a gap
- ◆ Zero: Missing data is treated as zero
- ◆ Interpolated: Missing data is calculated by using data on either side of the missing point(s). This option is available only for line charts.

Follow these steps to create the box plot:

1. Select the range F1:J7.
2. Click the Chart Wizard button.
3. In Step 1 of the Chart Wizard, select a Line chart type and the fourth chart subtype (Line with markers). Click Next.

4. In Step 2 of the Chart Wizard, select the Rows option. Click Finish to create the chart.
5. Activate the first data series (25th Percentile), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to None. Click the Options tab and place a check mark next to High-low lines and Up-down bars. Adjust the colors if desired.
6. Activate the second data series (Minimum), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to a horizontal bar. Adjust the colors if desired.
7. Activate the third data series (Mean), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to a diamond shape. Adjust the colors if desired.
8. Activate the fourth data series (50th Percentile), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to a horizontal bar. Adjust the colors if desired.
9. Activate the fifth data series (Maximum), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to a horizontal bar. Adjust the colors if desired.
10. Activate the sixth data series (75th Percentile), open the Format Data Series dialog box, and click the Patterns tab. Set the Line option to None. Set the Marker Style to None. Adjust the colors if desired.



After performing all of these steps, you may want to create a custom chart type to simplify the creation of additional box plots. Activate the chart, and select Chart → Chart Type. Click the Custom Types tab and choose the User-defined option. Click the Add button and specify a name and description for your chart.

Plotting Every nth Data Point

Normally, Excel doesn't plot data that resides in a hidden row or column. You can sometimes use this to your advantage, because it's an easy way to control what data appears in the chart.

Suppose you have a lot of data in a column, and you want to plot only every tenth data point. One way to accomplish this is to use AutoFilter in conjunction with a formula. Figure 17-9 shows a worksheet with AutoFilter in effect. The chart plots only the data in the visible (filtered) rows and ignores the values in the hidden rows.

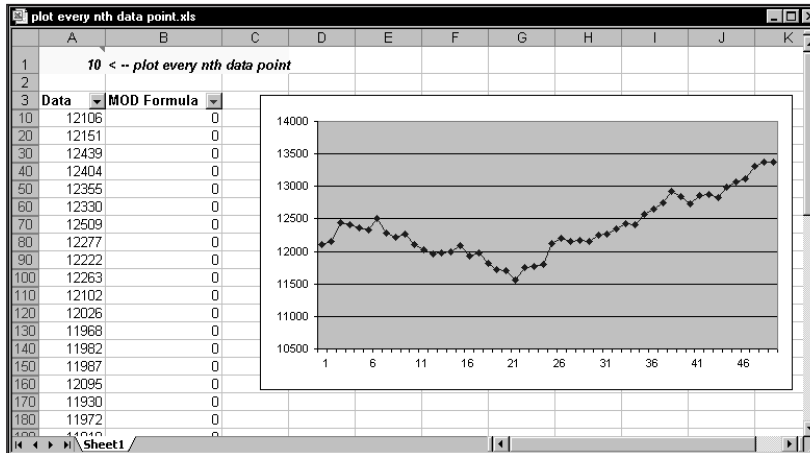


Figure 17-9: This chart plots every *n*th data point (specified in A1) by ignoring data in the rows hidden by AutoFiltering.



The workbook shown in Figure 17-9 also appears on the companion CD-ROM.

Cell A1 contains the value 10. The value in this cell determines which rows to hide. Column B contains identical formulas that use the value in cell A1. For example, the formula in cell B3 is:

```
=MOD(ROW(), $A$1)
```

This formula uses the MOD function to calculate the remainder when the row number (returned by the ROW function) is divided by the value in A1. As a result, every *n*th cell (the value in cell A1 determines *n*) contains 0. Then use the Data → Filter → AutoFilter command to turn on AutoFiltering. Set up the AutoFilter to display only the rows that contain a 0 in column B. Note that if you change the value in cell A1, you need to respecify the AutoFilter criteria for column B (the rows will not hide automatically).

The preceding formula uses the row number to determine which cells are visible. If you would prefer that the chart always includes the first data point, use the following formula, which refers to the cell (A4) that contains the first data point:

```
=MOD(ROW() - ROW($A$4), $A$1)
```



In some cases, you may not like the idea that hidden data is not displayed in your chart. To override this, activate the chart and select the Tools → Options command. In the Options dialog box, click the Chart tab and remove the check mark from the check box labeled Plot visible cells only.

Updating a Data Series Automatically

It's not difficult to change the data range used by a chart, but in some cases you may prefer a chart that updates automatically when you enter new data. If you have a chart that displays daily sales, for example, you probably need to change the chart's data range each day you add new data. This section describes a way to force Excel to update the chart's data range whenever you add new data to your worksheet.



A workbook that demonstrates automatically updating a data series appears on the companion CD-ROM.

To force Excel to update your chart automatically when you add new data, follow these steps:

1. Create the worksheet shown in Figure 17-10.
2. Select Insert → Name → Define to bring up the Define Name dialog box. In the Names in workbook field, enter **Date**. In the Refers to field, enter this formula:

```
=OFFSET(Sheet1!$A$2,0,0,COUNTA(Sheet1!$A:$A)-1)
```

3. Click Add. Notice that the OFFSET function refers to the first data point (cell A2) and uses the COUNTA function to get the number of data points in the column. Because column A has a heading in row 1, the formula subtracts 1 from the number.

4. Type **Sales** in the Names in workbook field. Enter this formula in the Refers to field:

```
=OFFSET(Sheet1!$B$2,0,0,COUNTA(Sheet1!$B:$B)-1)
```

5. Click Add and then OK to close the dialog box.
6. Activate the chart and select the data series.

- Replace the range references with the names that you defined in Steps 2 and 4. The formula should read:

```
=SERIES(,Sheet1!Date,Sheet1!Sales,1)
```

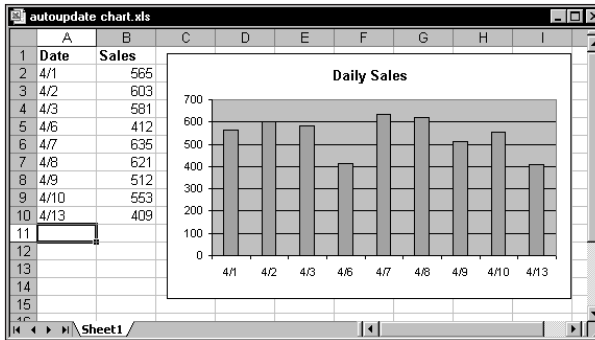


Figure 17-10: This chart updates automatically whenever you add new data to columns A and B.

After you perform these steps, the chart updates automatically when you add data to columns A and B.



To use this technique for your own data, make sure that the first argument for the OFFSET function refers to the first data point, and that the argument for COUNTA refers to the entire column of data. Also, if the columns used for the data contain any other entries, COUNTA returns an incorrect value.

Plotting the Last n Data Points

You can use a technique that makes your chart show only the most recent data points in a column. For example, you can create a chart that always displays the most recent 12 months of data (see Figure 17-11).

The instructions that follow describe how to create the chart in this figure.

- Create a worksheet like the one shown in Figure 17-11.
- Select Insert → Name → Define to bring up the Define Name dialog box. In the Names in workbook field, enter Dates. In the Refers to field, enter this formula:

```
=OFFSET(Sheet1!$A$1,COUNTA(Sheet1!$A:$A)-12,0,12,0)
```

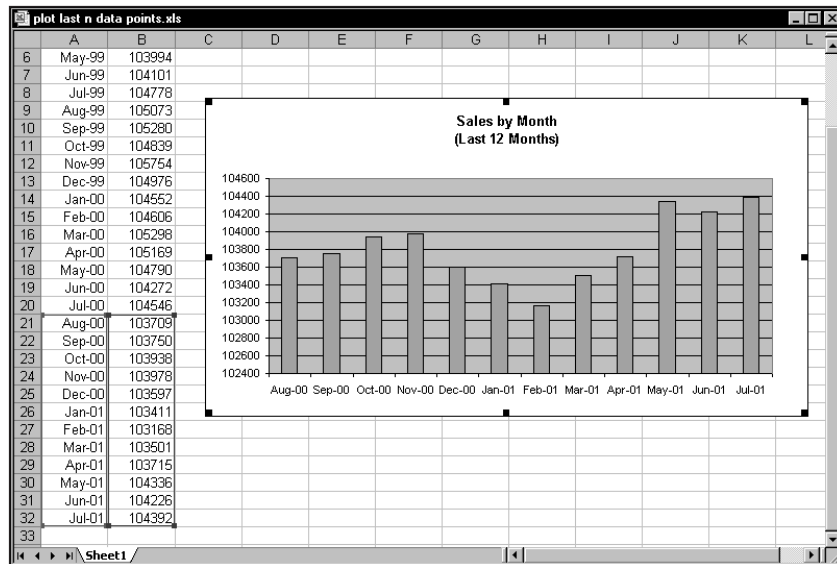


Figure 17-11: This chart displays the 12 most recent data points.

3. Click Add. Notice that the OFFSET function refers to cell A1 (not the cell with the first month).
4. Type Sales in the Names in workbook field. Enter this formula in the Refers to field:


```
=OFFSET(Sheet1!$B$1,COUNTA(Sheet1!$B:$B)-12,0,12,1)
```
5. Click Add and then click OK to close the dialog box.
6. Activate the chart and select the data series.
7. Replace the range references with the names that you defined in Steps 2 and 4. The formula should read:

```
=SERIES(,Sheet1!Dates,Sheet1!Sales,1)
```



To plot a different number of data points, adjust the formulas entered in Steps 2 and 4. Replace both occurrences of 12 with your new value.

Plotting Data Interactively

This section describes two techniques that you can use to get maximum value out of a single chart. As you'll see, the user determines data plotted by the chart – either by activating a row or selecting from a drop-down list.

Plotting Based on the Active Row

Figure 17-12 shows a chart that displays the data in the row that contains the cell pointer. When you move the cell pointer, press F9 and the chart displays the data from that row.

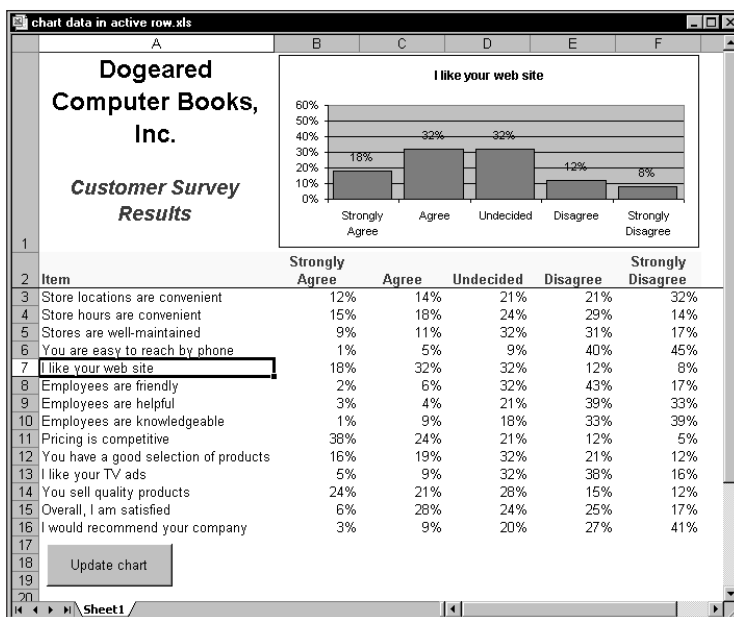


Figure 17-12: Pressing F9 displays the data in the row that contains the cell pointer.

The chart uses two named formulas, each with a mixed reference (the column part is absolute, but the row part is relative). The following names assume that cell A3 was active when the names were created. *ChartTitle* is defined as:

```
=OFFSET($A3,0,0)
```

ChartData is defined as:

```
=OFFSET($A3,0,1,1,5)
```

The SERIES formula for the chart's data series uses these named formulas. The SERIES formula looks like this:

```
=SERIES(Sheet1!ChartTitle,Sheet1!$B$2:$F$2,Sheet1!ChartData,1)
```

When the worksheet is recalculated, the named formulas get updated based on the active cell.



You can access the workbook shown in Figure 17-12 on the companion CD-ROM.

The worksheet contains a button that executes a simple VBA macro that determines if the cell pointer appears in a row that contains data (in other words, rows 3 through 16). If so, the sheet is calculated. If not, nothing happens. The macro listing follows:

```
Sub UpdateChart()
    If ActiveCell.Row > 2 And ActiveCell.Row < 17 Then _
        ActiveSheet.Calculate
End Sub
```

Selecting Data from a Combo Box

Figure 17-13 shows a chart that displays data as specified by a drop-down control (known as a Combo Box). The chart uses the data in B1:E2, but the month selected in the Combo Box determines the contents of these cells. Range A6:D17 contains the monthly data, and formulas in B2:E2 display the data using the value in cell A2. For example, when cell A4 contains the value 4, the chart displays data for April (the fourth month).

The formula in cell B2 is:

```
=INDEX(A6:A17,$A$2)
```

This formula was copied to C2:E2.

The key here is to get the Combo Box to display the month names and place the selected month index into cell A2. To create the Combo Box:

1. Select View → Toolbars → Forms to display the Forms toolbar.
2. On the Forms toolbar, click the control labeled Combo Box, and drag it into the worksheet to create the control.

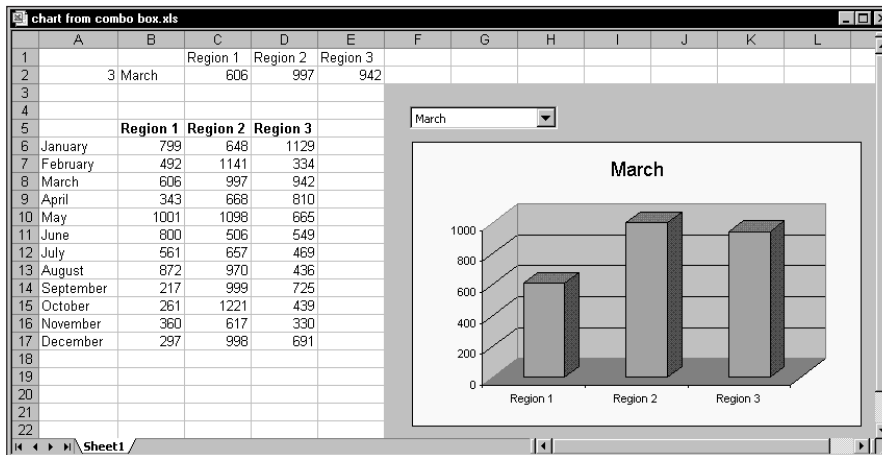


Figure 17-13: Selecting data to plot using a Combo Box

3. Double-click the Combo Box to display the Format Control dialog box.
4. In the Format Control dialog box, click the Control tab.
5. Specify A6:A17 as the Input range, and A2 as the Cell link.

You'll find that the Combo Box displays the month names and puts the index number of the selected month into cell A2. The formulas in row 2 display the appropriate data, which displays in the chart.



The workbook containing the Combo Box example appears on the companion CD-ROM.

Plotting Functions with One Variable

Excel's charting tools can plot various mathematical and trigonometric functions. For example, Figure 17-14 shows a plot of the SIN function, for values of x (expressed in radians) from -5 to $+5$ in increments of 0.5 . The function is expressed as:

$$y = \text{SIN}(x)$$

The chart is an XY Scatter chart, with the x values stored in column A and the y values in column B. Each pair of x and y values appear as a data point in the chart, and the points connect with a line.

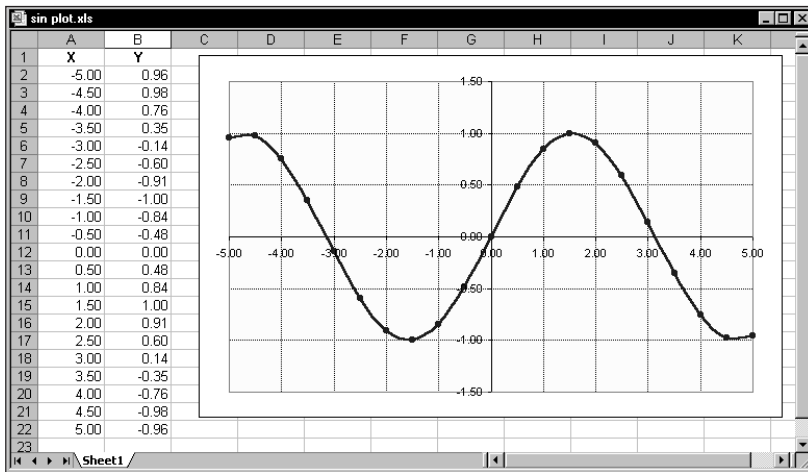


Figure 17-14: This chart plots the $\text{SIN}(x)$.

CREATING PLOTS

The key to creating plots of functions, of course, lies in coming up with two data ranges: one for the x values and one for the y values. The y values will be generated by formulas.



When plotting functions, make sure you select the XY chart type. If you use any other chart type, Excel always uses equal increments on the x -axis.

A BETTER WAY TO PLOT FUNCTIONS

You can use a technique developed by Stephen Bullen, an Excel expert extraordinaire. This method plots functions or formulas automatically, without actually generating any values in the worksheet! This is one of the most impressive Excel applications I've seen, and I'm grateful to Stephen for allowing me to use it in this book.



The companion CD-ROM features a workbook that demonstrates Stephen Bullen's plotting technique.

Figure 17-15 shows an example that plots the following function for 25 x values ranging from -5 to $+5$:

$$y = (x^3) * (x^2)$$

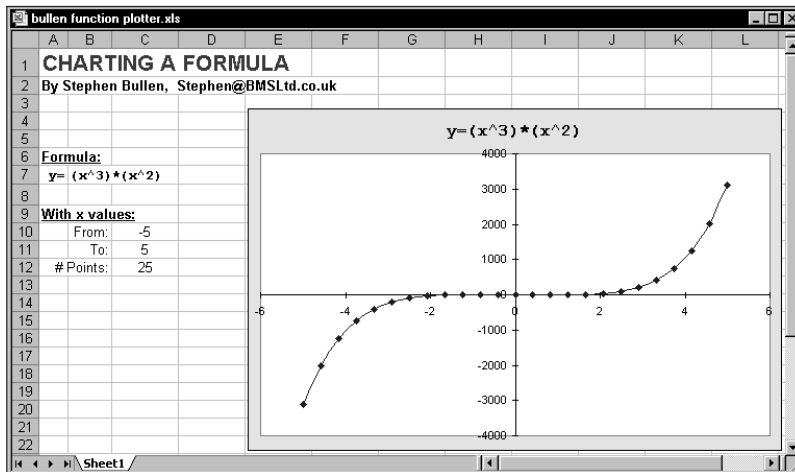


Figure 17-15: Plotting functions using a technique developed by Stephen Bullen

Stephen's technique uses two named formulas: *X* and *Y*. The SERIES function in the chart, which uses these defined names, looks like this:

```
=SERIES(,Sheet1!X,Sheet1!Y,1)
```

To plot a function using this worksheet:

1. Enter the formula *as text* into cell B7.
2. Enter the beginning value for *x* into cell C10.
3. Enter the ending value for *x* into cell C11.
4. Specify the number of points to plot in cell C12.

So how does it work? Let's start by analyzing the *X* formula, which generates the *x* values for the chart series:

```
=$C$10+(ROW(OFFSET($B$1,0,0,$C$12,1))-1)*($C$11-$C$10)/($C$12-1)
```



Use Excel's Insert → Name → Define command to examine the named formulas in this workbook.

This formula uses the OFFSET function to generate an array of *n* values, where the value in cell C12 determines *n*. The array begins with the value in C10 and ends

with the value in C11. The following expression calculates the increment between successive x values (calculated by subtracting the ending value from the beginning value, and dividing by the number of points minus 1):

```
=(C$11-C$10)/(C$12-1)
```

As an example, when x begins with -3 , ends with 3 , and contains five data points, the following array is created:

```
{-3, -1.5, 0, 1.5, 3}
```

The Y formula uses the EVALUATE function to create an array of y values for the chart:

```
=EVALUATE($B$7&"+"*0")
```

EVALUATE is an XLM macro function, and it cannot be used in a worksheet formula. This function essentially evaluates a string expression and returns a result. Although you cannot use the EVALUATE function in a worksheet formula, you *can* use it in a name.

Suppose B7 contains the string $SIN(x)$, and the first x value is -3 . The Y formula for the first data point is:

```
=EVALUATE("SIN(x)"&"+"*0")
```

Simplified, the function's argument is: $SIN(X)+X*0$



The $+"*0$ portion of the formula forces the result to be numeric.

The EVALUATE function grabs the first value (-3) from the array generated by the X formula, evaluates the following expression, and returns the result as the first y value:

```
SIN(-3)-3*0
```

The workbook also contains a formula in cell G8, which is hidden by the chart. The chart's title is linked to this cell, which contains the following formula:

```
=A7&B7&TEXT(NOW(),"")
```

This formula concatenates cells A7 and B7, and uses the NOW function to force an update of the chart if either of these cells is changed.

Plotting Functions with Two Variables

The preceding section described how to plot functions that use a single variable. For example, you can plot the following function for various values of x :

$$y = x^2$$

You also can plot functions that use two variables. For example, the following function calculates a value of z for various values of two variables (x and y):

$$z = \text{SIN}(x) * \text{COS}(y)$$

Figure 17-16 shows a surface chart that plots the value of z for x values ranging from -3.0 to 0 , and for y values ranging from 2.0 to 5.0 . Both x and y use an increment of 0.15 .

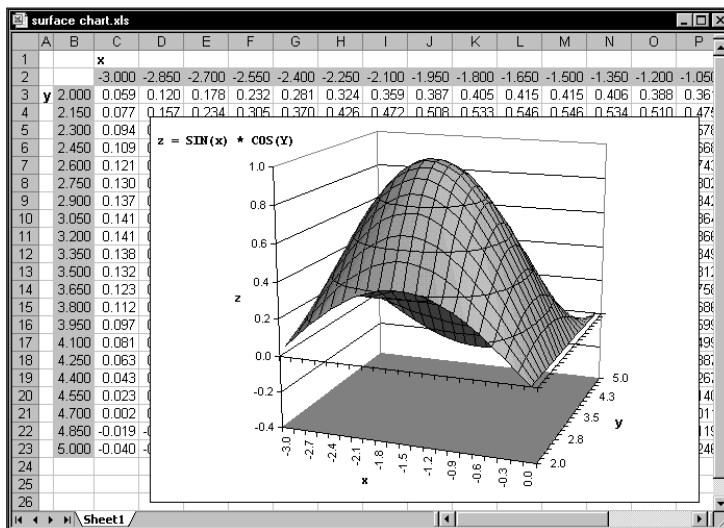


Figure 17-16: Using a surface chart to plot a function with two variables

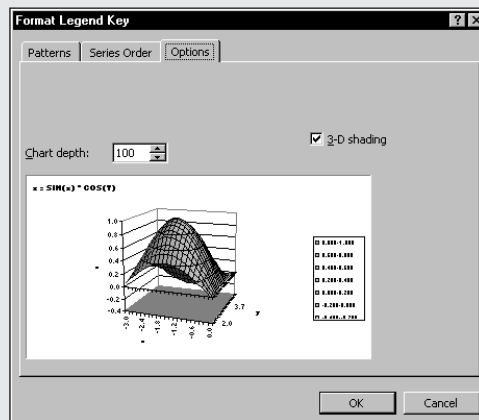
If you work with surface charts, you may notice that this chart type has some serious limitations. Ideally, you want to create an “XYZ surface chart,” in which you supply various values for X , Y , and Z . Unfortunately, Excel does not support this type of chart.

A surface chart in Excel essentially shows a 3-D view of what looks like a rubber sheet stretched over a 3-D column chart. The example in Figure 17-16 contains 21 data series (corresponding to values of y), each of which contains 21 data points (corresponding to values of x).

“Secret” Formatting Tips for Surface Charts

You may discover that Excel does not permit you to select an individual data series in a surface chart. Because of this, you cannot perform the types of formatting normally available in the Format Data Series dialog box.

You can apply some types of formatting to a Surface chart, but Excel makes you jump through a few hoops to get to the proper dialog box — Format Legend Key (see the accompanying figure). To get to this dialog box, make sure the Surface chart displays a legend. Then click the legend to select it and then click any legend key (a colored square to the left of the legend entry). Double-click the selected legend key and you'll get the Format Legend Key dialog box.



- ◆ Use the Patterns tab to change the color of the selected legend key; this also changes the color of the corresponding data series. If you would like your Surface chart to display using a single color, you need to change each legend key.
- ◆ Use the Options tab to change the depth of the chart. You can change the chart's depth by changing this setting while *any* legend key is selected.
- ◆ You can also apply 3-D shading in the Options tab. Again, this setting applies to the entire chart, not just the data series that corresponds to the selected legend entry.

To create a meaningful 3-D surface chart, you need to start with a 2-D range with the upper left cell empty. The top row should contain increasing or decreasing values of x with a constant difference between each x value. The left column should contain increasing or decreasing values of y with a constant difference between y values. The z values fill in the remaining cells corresponding to the respective x - y pair. Select the entire range as the source data for the chart.

Creating Awesome Designs

Figure 17-17 shows an example of an XY chart that displays “hypocycloid” curves using random values. This type of curve is the same as that generated by Hasbro’s popular SpiroGraph toy, which you may remember from childhood.

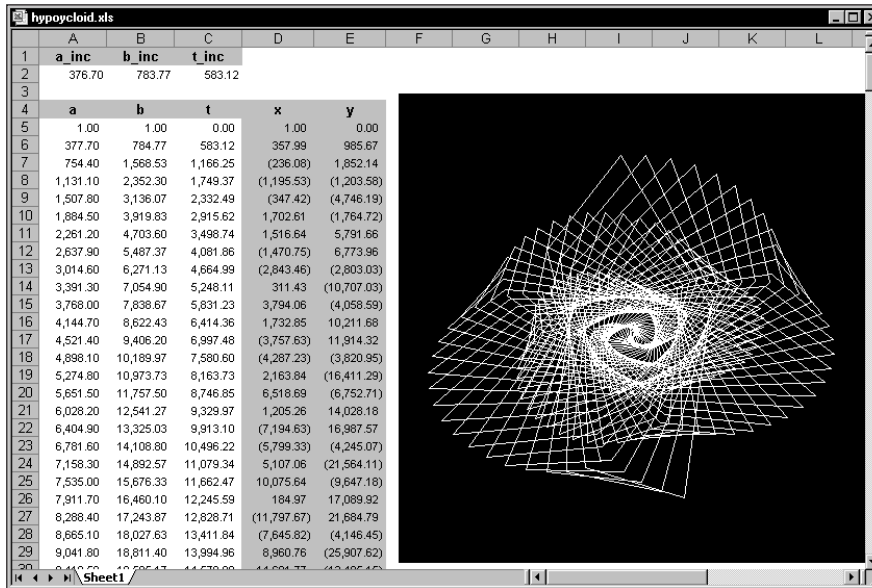


Figure 17-17: A hypocycloid curve



The companion CD-ROM contains a workbook with a more sophisticated example of the technique shown in Figure 17-17.

The chart uses data in columns D and E (the x and y ranges). These columns contain formulas that rely on data in columns A through C. The first column (column D) consists of the following formula:

$$=(A5-B5)*COS(C5)+B5*COS((A5/B5-1)*C5)$$

The formula in the second column (column E) is as follows:

$$=(A5-B5)*SIN(C5)-B5*SIN((A5/B5-1)*C5)$$

Pressing F9 recalculates the worksheet, which generates new increment values (random) for columns A through C, and creates a new display in the chart. The variety (and beauty) of charts generated using these formulas may amaze you.

Working with Trendlines

With some charts, you may want to plot a trendline that describes the data. A *trendline* points out general trends in your data. In some cases, you can forecast future data with trendlines. A single series can have more than one trendline.



In general, only XY Scatter charts should use a trendline. If you use a different chart type (such as Column or Line), the *x* values are assumed to be a series of integers that begin with 1.

Excel makes adding a trendline to a chart quite simple. Although you might expect this option to appear in the Format Data Series dialog box, it doesn't. You must go to the Add Trendline dialog box, shown in Figure 17-18, which you access by selecting Chart → Add Trendline. This command is available only when a data series is selected.

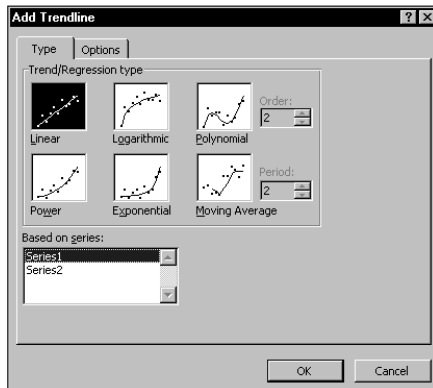


Figure 17-18: The Add Trendline dialog box offers several types of automatic trendlines.

The type of trendline that you choose depends on your data. Linear trends are the most common type, but you can describe some data more effectively with another type. When you click the Options tab in the Add Trendline dialog box, Excel displays the options shown in Figure 17-19.

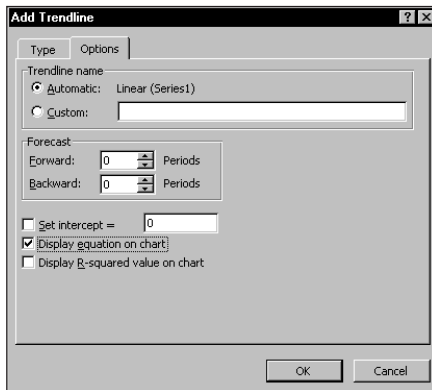


Figure 17-19: The Options tab in the Add Trendline dialog box

The Options tab enables you to specify a name to appear in the legend and the number of periods that you want to forecast. Additional options enable you to set the intercept value, specify that the equation used for the trendline should appear on the chart, and choose whether the R-squared value appears on the chart.

When Excel inserts a trendline, it may look like a new data series, but it's not. It's a new chart element with a name, such as Series 1 Trendline 1. And, of course, it does not have a corresponding SERIES formula. You can double-click a trendline to change its formatting or its options.

Linear Trendlines

Figure 17-20 shows two charts. The chart on the left depicts a data series without a trendline. As you can see, the data seems to be “linear” over time. The chart on the right is the same chart, but with a linear trendline that shows the trend in the data.

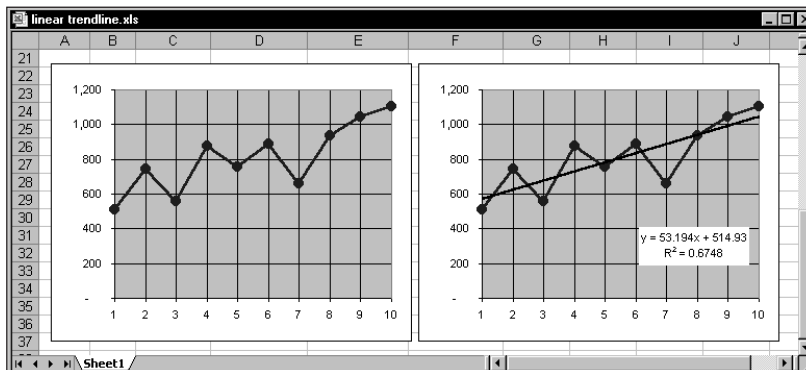


Figure 17-20: Before (left chart) and after (right chart) adding a linear trendline to a chart



The workbook shown in Figure 17-21 also appears on the companion CD-ROM.

The second chart also uses the options to display the equation and the R-squared value. In this example, the equation is:

$$y = 53.194x + 514.93$$

The R-squared value is 0.6748.



To display more or fewer decimal places in the equation and R-squared value, select the box and click the Increase Decimal or Decrease Decimal button on the Formatting toolbar.

What do these numbers mean? You can describe a straight line with an equation of the form:

$$y = mx + b$$

For each value of x (in this case, column B), you can calculate the predicted value of y (the value on the trendline) by using this equation. The variable m represents the slope of the line and b represents the y -intercept. For example, the month of February has an x value of 2 and a y value of 743. The predicted value for February, obtained using the following formula, is 621.318:

$$=(53.194*2)+514.93$$

The R-squared value, sometimes referred to as the *coefficient of determination*, ranges in value from 0 to 1. This value indicates how closely the estimated values for the trendline correspond to your actual data. A trendline is most reliable when its R-squared value is at or near 1.

CALCULATING THE SLOPE AND Y-INTERCEPT

As you know, Excel can display the equation for the trendline in a chart. This equation shows the slope (m) and y -intercept (b) of the best-fit trendline. You can calculate the value of the slope and y -intercept yourself, using the LINEST function in a formula.

Figure 17-21 shows 10 data points (x values in column B, y values in column C).

	A	B	C	D	E	F	G	H
1	Month	X	Actual Y				m	b
2	Jan	1	512				53.19394	514.9333
3	Feb	2	743					
4	Mar	3	559					
5	Apr	4	875					
6	May	5	755					
7	Jun	6	890					
8	Jul	7	663					
9	Aug	8	934					
10	Sep	9	1,042					
11	Oct	10	1,102					
12	Nov	11						
13	Dec	12						

Figure 17-21: Using the LINEST function to calculate slope and y-intercept

The formula that follows is an array formula that displays its result (the slope and y-intercept) in two cells:

```
{=LINEST(C2:C11,B2:B11) }
```

To enter this formula, start by selecting two cells (in this example, G2:H2). Then type the formula (without the brackets), and press Ctrl+Shift+Enter. Cell G2 displays the slope; cell H2 displays the y-intercept.

CALCULATING PREDICTED VALUES

Once you know the values for the slope and y-intercept, you can calculate the predicted y value for each x. Figure 17-22 shows the result. Cell E2 contains the following formula, which is copied down the column:

```
=(B2*$G$2)+$H$2
```

	A	B	C	D	E	F	G	H
1	Month	X	Actual Y		Predicted Y		m	b
2	Jan	1	512		568.1273		53.19394	514.9333
3	Feb	2	743		621.3212			
4	Mar	3	559		674.5152			
5	Apr	4	875		727.7091			
6	May	5	755		780.9030			
7	Jun	6	890		834.0970			
8	Jul	7	663		887.2909			
9	Aug	8	934		940.4848			
10	Sep	9	1,042		993.6788			
11	Oct	10	1,102		1,046.8727			
12	Nov	11			1,100.0667			
13	Dec	12			1,153.2606			

Figure 17-22: Column D contains formulas that calculate the predicted values for y.

The calculated values in column E represent the values used to plot the linear trendline. You can calculate predicted values of y without first computing the slope

and y -intercept. You do so with an array formula that uses the TREND function. Select D2:D11, type the following formula (without the brackets), and press Ctrl+Shift+Enter:

```
{=TREND(C2:C11,B2:B11)}
```

LINEAR FORECASTING

When your chart contains a trendline, you can instruct Excel to forecast and plot additional values. You do this on the Options tab in the Format Trendline dialog box (or the Options tab in the Add Trendline dialog box). Just specify the number of periods to forecast. Figure 17-23 shows a chart that forecasts results for two subsequent periods.

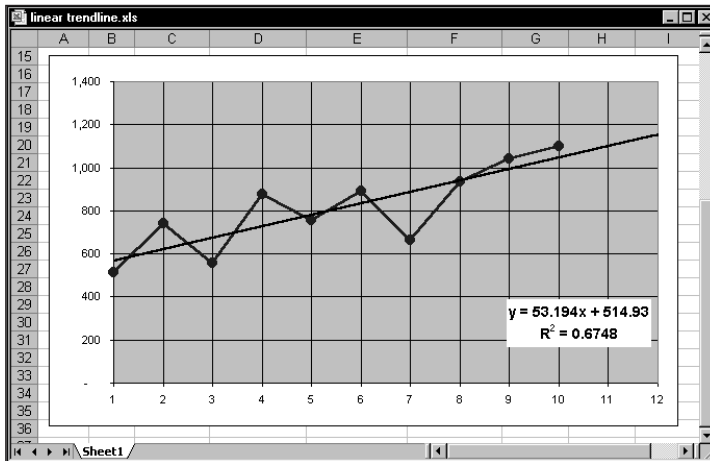


Figure 17-23: Using a trendline to forecast values for two additional periods of time

If you know the values of the slope and y -intercept (see “Calculating the Slope and Y-Intercept,” earlier in the chapter), you can calculate forecasts for other values of x . For example, to calculate the value of y when $x = 11$ (November), use the following formula:

```
=(53.194*11)+514.93
```

You can also forecast values by using the FORECAST function. The following formula, for example, forecasts the value for November (that is, $x = 11$) using known x and known y values:

```
=FORECAST(11,C2:C11,B2:B11)
```

CALCULATING R-SQUARED

The accuracy of forecasted values depends on how well the linear trendline fits your actual data. The value of R-squared represents the degree of fit. R-squared values closer to 1 indicate a better fit—and more accurate predictions. In other words, you can interpret R-squared as the proportion of the variance in y attributable to the variance in x .

As described previously, you can instruct Excel to display the R-squared value in the chart. Or, you can calculate it directly in your worksheet using the RSQ function. The following formula calculates R-squared for x values in B1:B11 and y values for C1:C11.

```
=RSQ(B2:B11,C2:C11)
```



The value of R-squared calculated by the RSQ function is valid only for a linear trendline.

Nonlinear Trendlines

Curve fitting refers to the process of making projections beyond a data range (extrapolation) or for making estimates between acquired data points (interpolation). Besides linear trendlines, an Excel chart can display trendlines of the following types:

- ◆ **Logarithmic:** Used when the rate of change in the data increases or decreases quickly, and then flattens out.
- ◆ **Power:** Used when the data consists of measurements that increase at a specific rate. The data cannot contain zero or negative values.
- ◆ **Exponential:** Used when data values rise or fall at increasingly higher rates. The data cannot contain zero or negative values.
- ◆ **Polynomial:** Used when data fluctuates. You can specify the order of the polynomial (from 2 to 6) depending on the number of fluctuations in the data.



The Type tab in the Trendline dialog box offers the option of Moving average, which really isn't a trendline. This option, however, can be useful for smoothing out "noisy" data. The Moving average option enables you to specify the number of data points to include in each average. For example, if you select 5, Excel averages every group of five data points.

Earlier in this chapter, I described how to calculate the slope and y -intercept for the linear equation that describes a linear trendline. Nonlinear trendlines also have equations, as described in the sections that follow.



The companion CD-ROM contains a workbook with the nonlinear trendline examples described in this section.

LOGARITHMIC TRENDLINE

The equation for a logarithmic trendline is:

$$y = (c * \text{LN}(x)) - b$$

Figure 17-24 shows a chart with a logarithmic trendline added. A single array formula in E2:F2 calculates the values for c and b . The formula is:

```
{=LINEST(C2:C11, LN(B2:B11))}
```

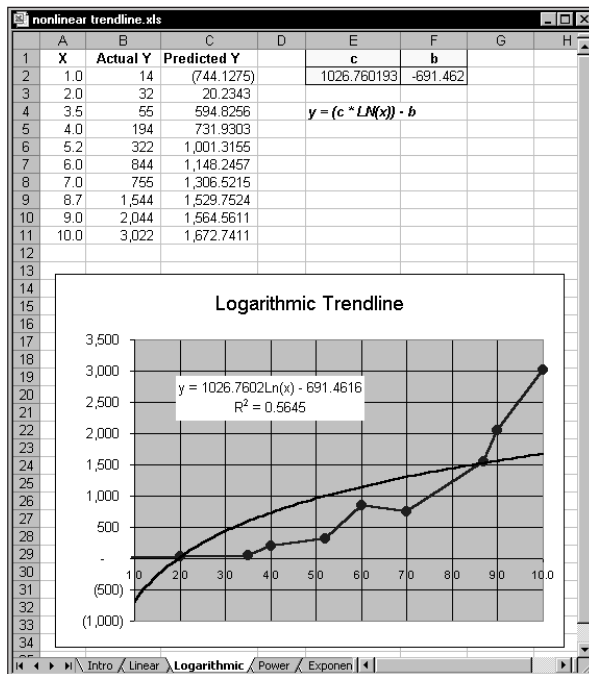


Figure 17-24: A chart displaying a logarithmic trendline

Column C shows the predicted y values for each value of x , using the calculated values for b and c . For example, the formula in cell C2 is:

$$=(\$E\$2*\text{LN}(A2))+\$F\$2$$

As you can see, a logarithmic trendline does not provide a good fit for this data. The R-square value is low, and the trendline does not match the data.

POWER TRENDLINE

The equation for a power trendline looks like this:

$$y = c * x^b$$

Figure 17-25 shows a chart with a power trendline added. The first element in a two-cell array formula in E2:F2 calculates the values for b . The formula is:

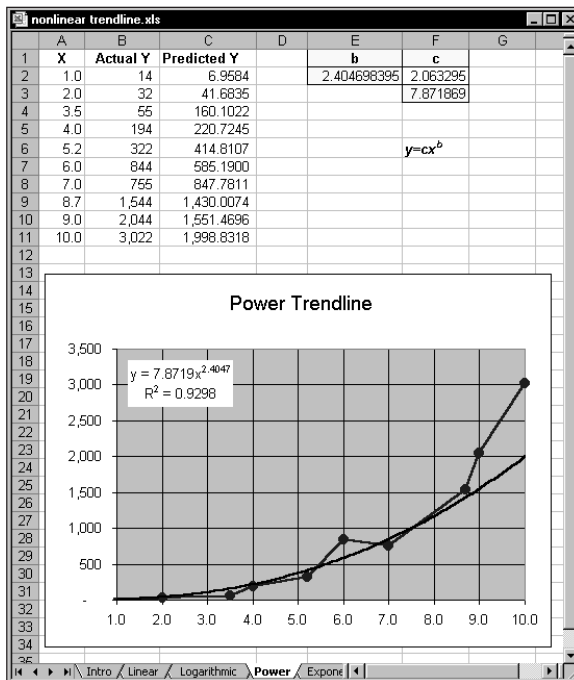
$$=\text{LINEST}(\text{LN}(B2:B11), \text{LN}(A2:A11), , \text{TRUE})$$


Figure 17-25: A chart displaying a power trendline

The following formula, in cell F3, calculates the value for c :

$$=\text{EXP}(F2)$$

Column C shows the predicted y values for each value of x , using the calculated values for b and c . For example, the formula in cell C2 is:

```
=F$3*(A2^$E$2)
```

EXPONENTIAL TRENDLINE

The equation for an exponential trendline looks like this:

$$y = c * \text{EXP}(b * x)$$

Figure 17-26 shows a chart with an exponential trendline added. The first element in a two-cell array formula in F2:G2 calculates the values for b . The formula is:

```
{=LINEST(LN(B2:B11),A2:A11)}
```

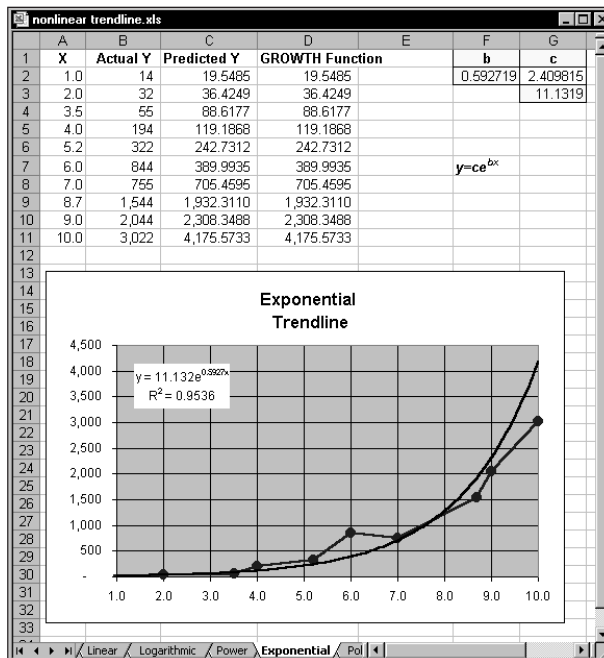


Figure 17-26: A chart displaying an exponential trendline

The following formula, in cell G3, calculates the value for c :

```
=EXP(G2)
```

Column C shows the predicted y values for each value of x , using the calculated values for b and c . For example, the formula in cell C2 is:

```
=G$3*EXP($F$2*A2)
```

Column D uses the GROWTH function in an array formula to generate predicted y values. The array formula, entered in D2:D10, appears like this:

```
{=GROWTH(B2:B11,A2:A11)}
```

POLYNOMIAL TRENDLINE

When you request a polynomial trendline, you also need to specify the order of the polynomial (ranging from 2 through 6). The equation for a polynomial trendline depends on the order. The following equation, for example, is for a third-order polynomial trendline:

$$y = (c_3 * x^3) + (c_2 * x^2) + (c_1 * x^1) + b$$

Notice that there are three c coefficients (one for each order).

Figure 17-27 shows a chart with a third-order polynomial trendline added. A four-element array formula entered in F2:I2 calculates the values for each of three c coefficients and the b coefficient. The formula is:

```
{=LINEST(B2:B11,A2:A11^{1,2,3})}
```

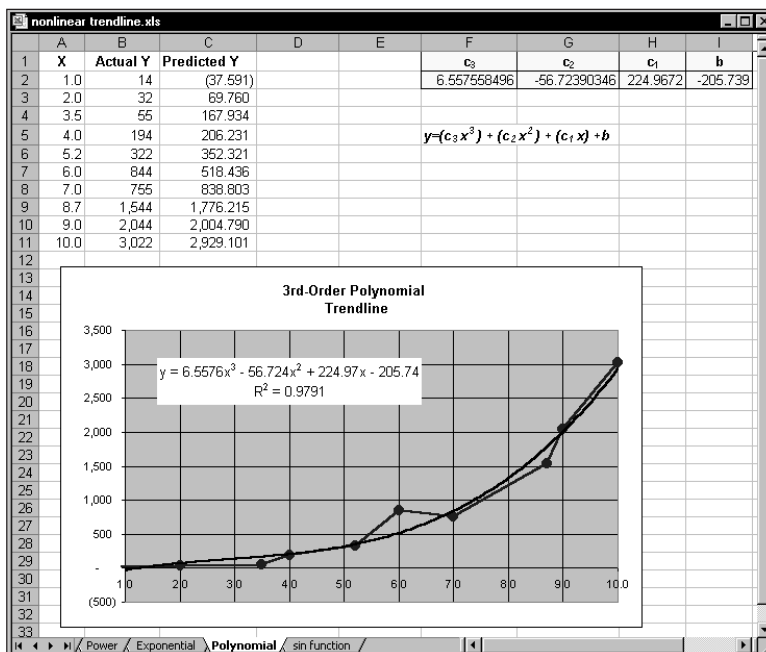


Figure 17-27: A chart displaying a polynomial trendline

Column C shows the predicted y values for each value of x , using the calculated values for b and the three c coefficients. For example, the formula in cell C2 is:

$$=(\$F\$2*A2^3)+(\$G\$2*A2^2)+(\$H\$2*A2)+\$I\$2$$

Useful Chart Tricks

This section contains a number of useful charting tricks that I've accumulated over the years. These tricks include storing multiple charts on a chart sheet, viewing an embedded chart in a window, changing worksheet values by dragging data points in a chart, and animating charts.

Storing Multiple Charts on a Chart Sheet

Most Excel users would agree that a chart sheet holds a single chart. Most of the time, that's a true statement. However, it's certainly possible to store multiple charts on a single chart sheet. In fact, Excel enables you to do this directly. If you activate an embedded chart and then select Chart → Location, Excel displays its Chart Location dialog box. If you select the As new sheet option and specify an existing chart sheet as the location, you see the dialog box shown in Figure 17-28. Click OK and the chart appears on top of the chart in the chart sheet.

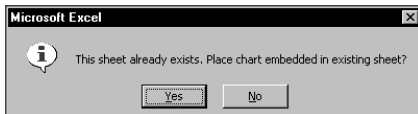


Figure 17-28: Excel enables you to relocate an embedded chart to an existing chart sheet.

Generally, you'll want to add embedded charts to an *empty* chart sheet. To create an empty chart sheet, select a single blank cell and press F11. Or, you can select the chart area in a chart sheet and press Del.

By storing multiple charts on a chart sheet, you can take advantage of the View → Sized with Window command to automatically scale the charts to the window size and dimensions. Figure 17-29 shows an example of a chart sheet that contains six embedded charts.



This workbook is available on the companion CD-ROM.

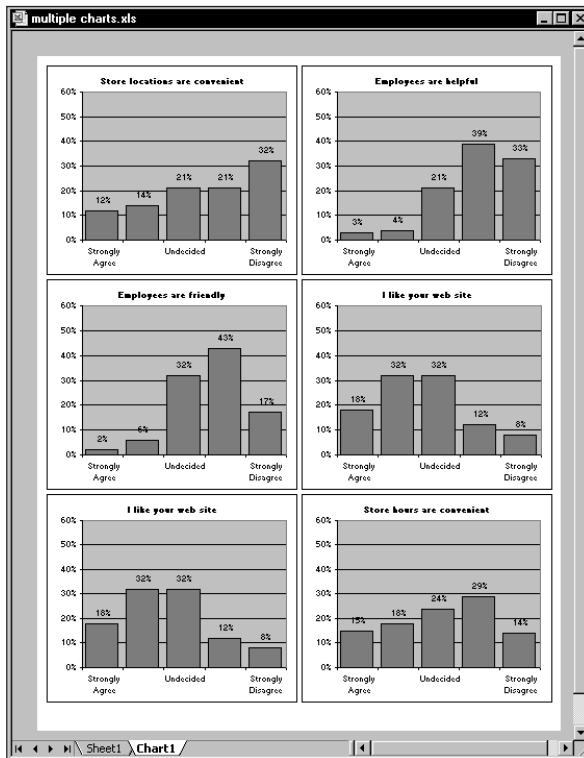


Figure 17-29: This chart sheet contains six embedded charts.

Viewing an Embedded Chart in a Window

When you activate an embedded chart, the chart actually is contained in a window that is normally *invisible*. To see an embedded chart in its own window, right-click the embedded chart and select Chart Window from the shortcut menu. The embedded chart remains on the worksheet, but the chart also appears in its own floating window. You can move and resize this window (but you can't maximize it). If you move the window, you'll notice that the embedded chart still displays in its original location. Activating any other window makes the embedded chart window invisible again.

Changing a Worksheet Value by Dragging a Data Point

Excel provides an interesting chart-making feature that also can prove somewhat dangerous. This feature enables you to change the value in a worksheet by dragging the data markers on two-dimensional line charts, bar charts, column charts, XY charts, and bubble charts.

Here's how it works. Select an individual data point in a chart series (not the entire series) and then drag the point in the direction in which you want to adjust the value. As you drag the data marker, the corresponding value in the worksheet changes to correspond to the data point's new position on the chart.

If the value of a data point that you move is the result of a formula, Excel displays its Goal Seek dialog box. Use this dialog box to specify the cell that Excel should adjust to make the formula produce the result that you pointed out on the chart. This technique is useful if you know what a chart should look like and you want to determine the values that will produce the chart.



Obviously, this feature can be dangerous, because you inadvertently can change values that you shouldn't — so exercise caution.

Using Animated Charts

Most people don't realize it, but Excel is capable of performing simple animations using shapes and charts (animations require macros). Consider the XY chart shown in Figure 17-30.

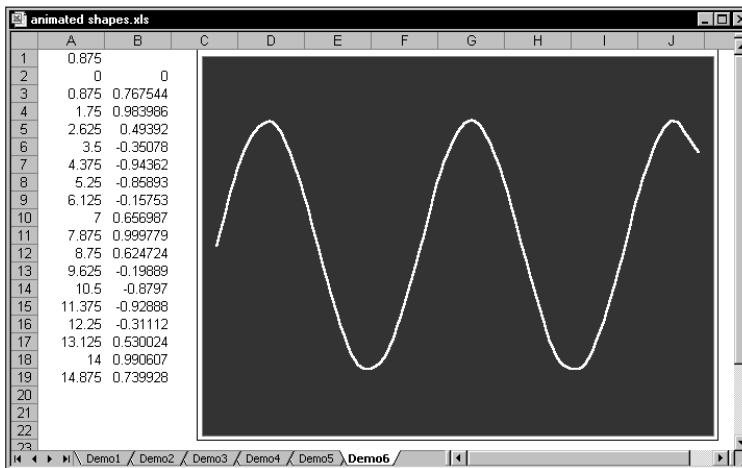


Figure 17-30: A simple VBA procedure turns this chart into an interesting animation.

The x values (column A) depend on the value in cell A1. The value in each row represents the previous row's value, plus the value in A1. Column B contains formulas that calculate the SIN of the corresponding value in column A. The following simple procedure produces an interesting animation. It simply changes the value in cell A1, which causes the values in the x and y ranges to change.

```

Sub AnimateChart()
    Range("A1") = 0
    For i = 1 To 150
        Range("A1") = Range("A1") + 0.035
    Next i
    Range("A1") = 0
End Sub

```



The companion CD-ROM contains a workbook that features this animated chart, plus several other animation examples.

Creating a “Gauge” Chart

Figure 17-31 shows what appears to be a new chart type that resembles a gauge. Actually, it’s a standard pie chart, but with one hidden slice. The hidden slice occupies 50 percent of the chart, and it was hidden by setting its fill color to transparent and specifying no border.

The pie chart uses the values in range A1:A3. Cell A1 contains the value 1, and this represents the hidden slice. Cell A2 contains the value that will appear in the gauge. Cell A3 contains this simple formula:

=1-A2

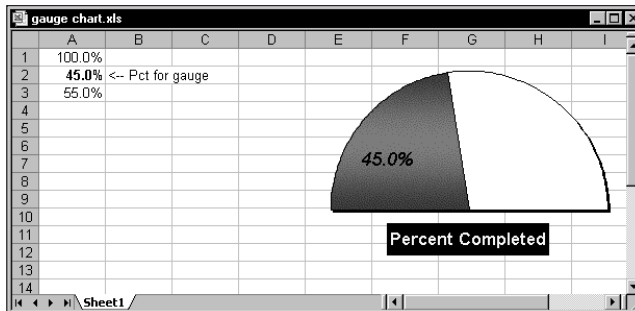


Figure 17-31: Hiding one slice of a pie chart creates a gauge chart.

Creating a “Clock” Chart

Figure 17-32 shows an XY chart formatted to look like a clock. It not only *looks* like a clock, but it also functions like a clock. There is really no reason why anyone would need to display a clock such as this on a worksheet, but creating the workbook was challenging, and you may find it instructive.

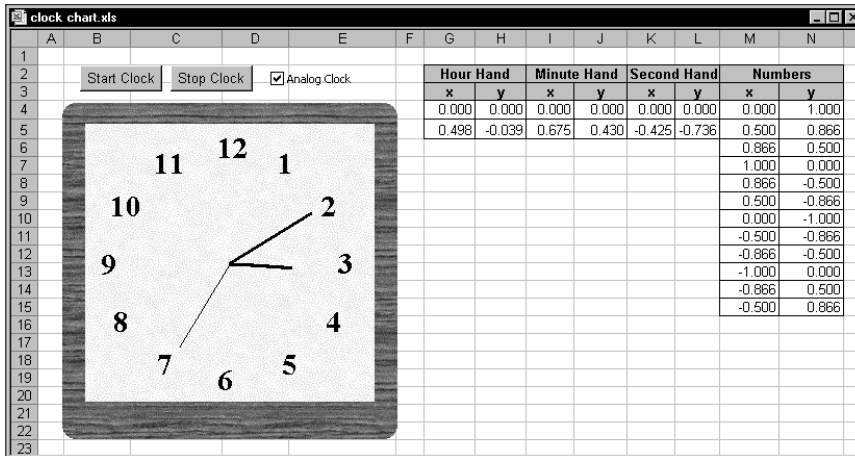


Figure 17-32: This fully functional clock is actually an XY chart in disguise.

The chart uses four data series: one for the hour hand, one for the minute hand, one for the second hand, and one for the numbers. The last data series draws a circle with 12 points. The numbers consist of manually entered data labels. (See the sidebar, “Plotting a Circle.”)

The formulas listed in Table 17-2 calculate the data series for the clock hands (the range G4:L4 contains zero values, not formulas).

TABLE 17-2 FORMULAS USED TO GENERATE A CLOCK CHART

Cell	Description	Formula
G5	Origin of hour hand	$=0.5*\text{SIN}((\text{HOUR}(\text{NOW}())+(\text{MINUTE}(\text{NOW}())/60))\text{*}(2*\text{PI}()/12))$
H5	End of hour hand	$=0.5*\text{COS}((\text{HOUR}(\text{NOW}())+(\text{MINUTE}(\text{NOW}())/60))\text{*}(2*\text{PI}()/12))$

Continued

TABLE 17-2 FORMULAS USED TO GENERATE A CLOCK CHART (Continued)

Cell	Description	Formula
I5	Origin of minute hand	$=0.8*\text{SIN}((\text{MINUTE}(\text{NOW}())+(\text{SECOND}(\text{NOW}())/60))*2*\text{PI}()/60)$
J5	End of minute hand	$=0.8*\text{COS}((\text{MINUTE}(\text{NOW}())+(\text{SECOND}(\text{NOW}())/60))*2*\text{PI}()/60)$
K5	Origin of second hand	$=0.85*\text{SIN}(\text{SECOND}(\text{NOW}))*2*\text{PI}()/60)$
L5	End of second hand	$=0.85*\text{COS}(\text{SECOND}(\text{NOW}))*2*\text{PI}()/60)$

This workbook uses a simple VBA procedure, which recalculates the worksheet every second.

In addition to the clock chart, the workbook contains a text box that displays the time using the NOW() function, as shown in Figure 17-33. Normally hidden, you can display this text box by deselecting the Analog clock check box. A simple VBA procedure attached to the check box hides and unhides the chart, depending on the status of the check box.

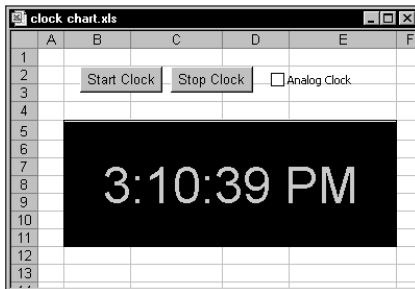


Figure 17-33: Displaying a digital clock in a worksheet is much easier, but not as fun to create.



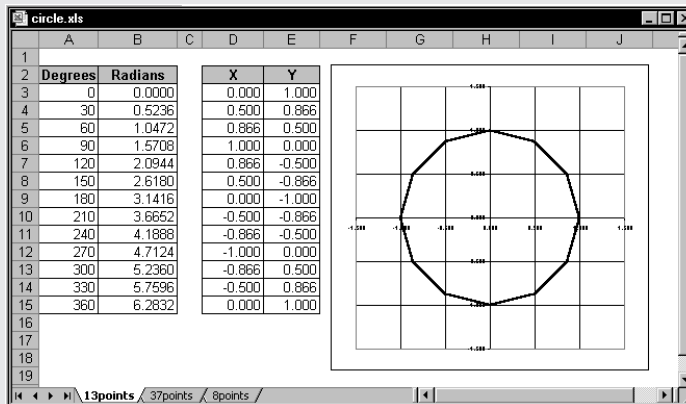
The workbook with the animated clock example appears on the companion CD-ROM. The CD also includes a different version of this file that uses VBA procedures instead of formulas.

When you examine the workbook, keep the following points in mind:

- ◆ The ChartObject, named *ClockChart*, covers up a range named *DigitalClock*—used to display the time digitally.
- ◆ The two buttons on the worksheet are from the Forms toolbar, and each has a VBA procedure assigned to it (StartClock and StopClock).
- ◆ The CheckBox control (named *cbClockType*) on the worksheet is from the Forms toolbar, not from the Control Toolbox toolbar. Clicking the object executes a procedure named *cbClockType_Click*, which simply toggles the Visible property of the ChartObject. When invisible, the digital clock is revealed.
- ◆ The chart is an XY chart with four data series. These series represent the hour hand, the minute hand, the second hand, and the 12 numbers.

Plotting a Circle

You can create an XY chart that draws a perfect circle. To do so, you need two ranges, one for the x values and another for the y values. The number of data points in the series determines the *smoothness* of the circle. Or, you simply select the Smoothed line option in the Format Series dialog box (Patterns tab) for the data series.



The example shown (available on the companion CD-ROM) uses 13 points to create the circle. If you work in degrees, generate a series of values such as the ones shown in column A. The series starts with 0 and has 30-degree increments. If you work in radians (column B), the first series starts with 0 and increments by $\pi/6$.

Continued

Plotting a Circle *(Continued)*

The ranges used in the chart appear in columns D and E. If you work in degrees, the formula in D3 is:

```
=SIN(RADIANS(A3))
```

The formula in E3 is:

```
=COS(RADIANS(A3))
```

If you work in radians, use this formula in D3:

```
=SIN(A3)
```

And use this formula in E3:

```
=COS(A3)
```

The formulas in D3 and E3 simply copy down to subsequent rows.

To plot a circle with more data points, you need to adjust the increment value in columns A and B (or C and D if working in radians). The final value should be the same as those shown in row 15. In degrees, the increment is 360 divided by the number of data points minus 1. In radians, the increment is π divided by the number of data points minus 1, divided by 2.

- ◆ The UpdateClock procedure executes when you click the Start Clock button. This procedure determines which clock is visible and performs the appropriate updating.
- ◆ The UpdateClock procedure uses the OnTime method of the Application object. This method enables you to execute a procedure at a specific time. Before the UpdateClock procedure ends, it sets up a new OnTime event that occurs in one second. In other words, the UpdateClock procedure is called every second.
- ◆ The UpdateClock procedure uses some basic trigonometry to determine the angles at which to display the hands on the clock.

Drawing with an XY Chart

The final example has absolutely no practical value, but you may find it interesting (and maybe even a bit entertaining). The worksheet consists of an embedded XY chart, along with a number of controls. (These controls, from the Forms toolbar, are not ActiveX controls.)



The workbook demonstrating drawing with an XY chart appears on the companion CD-ROM.

Clicking one of the arrow buttons draws a line in the chart, the size of which is determined by the step value, set with one of the Spin controls. With a little practice (and patience) you can create simple sketches. Figure 17-34 shows an example.

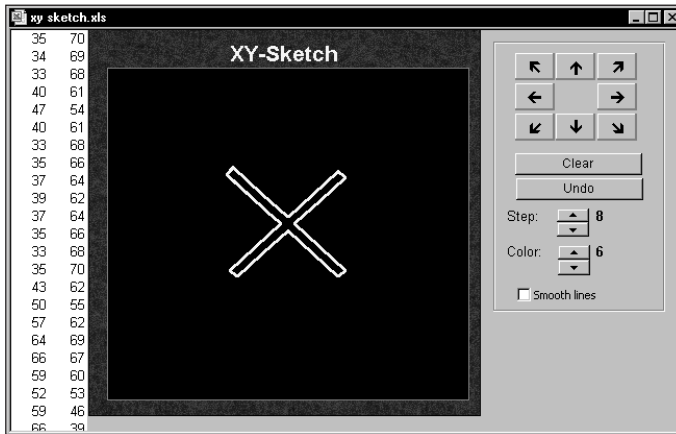


Figure 17-34: This drawing is actually an embedded XY chart.

Clicking an arrow button executes a macro that adds two values to a range: an *x* value and a *y* value. It then redefines two range names (*XRange* and *YRange*) used in the chart's *SERIES* formula. Particularly handy is the multilevel *Undo* button. Clicking this button simply erases the last two values in the range, and then redefines the range names. Additional accoutrements include the capability to change the color of the lines, and the capability to display smoothed lines.

Summary

This chapter presented details on the *SERIES* formula used in charts, and presented several examples of nonstandard charts that you can produce with Excel. The chapter also discussed various types of trendlines and provided techniques for plotting functions. It presented a variety of useful chart tips and techniques that you can adapt for use with your charts.

The next chapter covers formula techniques with pivot tables.

Chapter 18

Pivot Tables

IN THIS CHAPTER

- ◆ An introduction to pivot tables
- ◆ How to create a pivot table from a database
- ◆ How to group items in a pivot table
- ◆ How to create a calculated field or a calculated item in a pivot table

EXCEL'S PIVOT TABLE FEATURE probably represents the most technologically sophisticated component in Excel. This chapter may seem a bit out of place in this book. After all, a pivot table does its job without using formulas. That's exactly the point. If you haven't yet discovered the power of pivot tables, this chapter will demonstrate how using a pivot table can serve as an excellent alternative to creating many complex formulas.

About Pivot Tables

A *pivot table* is essentially a dynamic summary report generated from a database. The database can reside in a worksheet or in an external file. A pivot table can help transform endless rows and columns of numbers into a meaningful presentation of the data.

For example, a pivot table can create frequency distributions and cross-tabulations of several different data dimensions. In addition, you can display subtotals and any level of detail that you want. Perhaps the most innovative aspect of a pivot table lies in its interactivity. After you create a pivot table, you can rearrange the information in almost any way imaginable, and even insert special formulas that perform new calculations. You even can create post hoc groupings of summary items (for example, combine Northern Region totals with Western Region totals).

As far as I can tell, the term *pivot table* is unique to Excel. The name stems from the fact that you can rotate (that is, pivot) the table's row and column headings around the core data area to give you different views of your summarized data.

One minor drawback to using a pivot table is that, unlike a formula-based summary report, a pivot table does not update automatically when you change the source data. This does not pose a serious problem, however, since a single click of the Refresh toolbar button forces a pivot table to use the latest data.

A Pivot Table Example

The best way to understand the concept of a pivot table is to see one. Start with Figure 18-1, which shows the data used in creating the pivot table in this chapter.

	A	B	C	D	E	F
1	Date	Amount	AcctType	OpenedBy	Branch	Customer
2	9/1/2000	340	Checking	New Accts	Central	Existing
3	9/1/2000	15,759	CD	Teller	Westside	Existing
4	9/1/2000	15,276	CD	New Accts	North County	Existing
5	9/1/2000	12,000	CD	New Accts	Westside	Existing
6	9/1/2000	5,000	CD	New Accts	North County	Existing
7	9/1/2000	7,000	Savings	New Accts	North County	New
8	9/1/2000	5,000	Savings	New Accts	Westside	Existing
9	9/1/2000	4,623	Savings	New Accts	North County	Existing
10	9/1/2000	5,879	Checking	New Accts	Central	Existing
11	9/1/2000	3,171	Checking	New Accts	Westside	Existing
12	9/1/2000	4,000	Savings	New Accts	Central	Existing
13	9/1/2000	5,000	Checking	New Accts	Central	Existing
14	9/1/2000	16,000	CD	New Accts	Central	New
15	9/1/2000	50,000	Savings	New Accts	Central	Existing
16	9/1/2000	13,636	CD	New Accts	North County	Existing
17	9/4/2000	50,000	CD	New Accts	North County	New
18	9/4/2000	15,000	CD	New Accts	Westside	New
19	9/4/2000	13,000	CD	New Accts	North County	New
20	9/4/2000	13,000	CD	New Accts	Central	Existing
21	9/4/2000	3,000	Checking	New Accts	Central	Existing
22	9/4/2000	2,878	Savings	New Accts	North County	Existing
23	9/4/2000	13,519	CD	New Accts	Central	New
24	9/4/2000	4,000	Checking	New Accts	Central	Existing
25	9/4/2000	3,075	Checking	New Accts	Westside	Existing
26	9/4/2000	4,000	Checking	New Accts	North County	Existing
27	9/4/2000	6,000	Savings	New Accts	Central	Existing
28	9/4/2000	65,000	Savings	New Accts	Westside	Existing
29	9/4/2000	240	Checking	New Accts	Central	Existing

Figure 18-1: This database is used to create a pivot table.

This database consists of daily new-account information for a three-branch bank. The database contains 350 records, and tracks the following:

- ◆ The date that each account was opened
- ◆ The opening amount
- ◆ The account type (CD, checking, savings, or IRA)
- ◆ Who opened the account (a teller or a new-account representative)
- ◆ The branch at which it was opened (Central, Westside, or North County)
- ◆ Whether a new customer or an existing customer opened the account



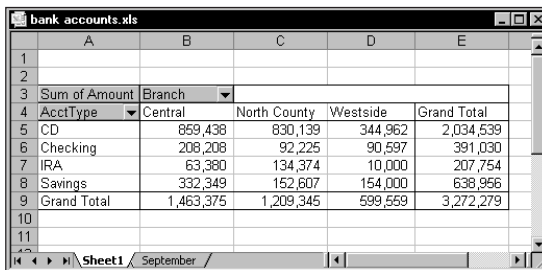
The workbook shown in Figure 18-1 also appears on the companion CD-ROM.

The bank accounts database contains a lot of information. But in its current form, the data does not reveal much. To make the data more useful, you need to summarize it. Summarizing a database is essentially the process of answering questions about the data. Following are a few questions that may be of interest to the bank's management:

- ◆ What is the total deposit amount for each branch, broken down by account type?
- ◆ How many accounts were opened at each branch, broken down by account type?
- ◆ What's the dollar distribution of the different account types?
- ◆ What types of accounts do tellers open most often?
- ◆ How does the Central branch compare to the other two branches?
- ◆ Which branch opens the most accounts for new customers?

You could, of course, write formulas to answer these questions. Often, however, a pivot table is a better choice. Creating a pivot table takes only a few seconds and doesn't require a single formula.

Figure 18-2 shows a pivot table created from the database displayed in Figure 18-1. This pivot table shows the amount of new deposits, broken down by branch and account type. This particular summary represents one of dozens of summaries that you can produce from this data.



Sum of Amount	Branch			
AcctType	Central	North County	Westside	Grand Total
CD	859,438	830,139	344,962	2,034,539
Checking	208,208	92,225	90,597	391,030
IRA	63,380	134,374	10,000	207,754
Savings	332,349	152,607	154,000	638,956
Grand Total	1,463,375	1,209,345	599,559	3,272,279

Figure 18-2: A simple pivot table

Figure 18-3 shows another pivot table generated from the bank data. This pivot table uses a page field for the Customer item. In this case, the pivot table displays the data only for existing customers (the user could also select New or All from page field list). Notice the changes in the orientation of the table; branches appear in rows and account types appear in columns. This is another example of the flexibility of a pivot table.

Pivot Table Terminology

Understanding the terminology associated with pivot tables is the first step in mastering this feature. Refer to the accompanying figure to get your bearings.

Sum of Amount		Customer		
Branch	AcctType	Existing	New	Grand Total
Central	CD	671,289	123,149	794,438
	Checking	156,884	49,228	206,112
	IRA	27,000	-	27,000
	Savings	239,347	70,600	309,947
Central Total		1,094,520	242,977	1,337,497
North County	CD	646,184	152,500	798,684
	Checking	55,880	20,070	75,950
	IRA	35,554	7,000	42,554
	Savings	87,136	39,607	126,743
North County Total		824,754	219,177	1,043,931
Westside	CD	143,766	71,437	215,203
	Checking	52,978	7,419	60,397
	IRA	10,000	-	10,000
	Savings	153,500	500	154,000
Westside Total		360,244	79,356	439,600
Grand Total		2,279,518	541,510	2,821,028

- ◆ **Column field:** A field that has a column orientation in the pivot table. Each item in the field occupies a column. In the figure, Customer represents a column field that contains two items (Existing and New). You can have nested column fields.
- ◆ **Data area:** The cells in a pivot table that contain the summary data. Excel offers several ways to summarize the data (sum, average, count, and so on). In the figure, the Data area includes C5:E20.
- ◆ **Grand totals:** A row or column that displays totals for all cells in a row or column in a pivot table. You can specify that grand totals be calculated for rows, columns, or both (or neither). The pivot table in the figure shows grand totals for both rows and columns.
- ◆ **Group:** A collection of items treated as a single item. You can group items manually or automatically (group dates into months, for example). The pivot table in the figure does not have any defined groups.
- ◆ **Item:** An element in a field that appears as a row or column header in a pivot table. In the figure, Existing and New are items for the Customer field. The Branch field has three items: Central, North County, and Westside. AcctType has four items: CD, Checking, IRA, and Savings.

- ◆ Page field: A field that has a page orientation in the pivot table – similar to a slice of a three-dimensional cube. You can display only one item (or all items) in a page field at one time. In the figure, OpenedBy represents a page field that displays the New Accts item; the pivot table shows data only for New Accts.
- ◆ Refresh: To recalculate the pivot table after making changes to the source data.
- ◆ Row field: A field that has a row orientation in the pivot table. Each item in the field occupies a row. You can have nested row fields. In the figure, Branch and AcctType both represent row fields.
- ◆ Source data: The data used to create a pivot table. It can reside in a worksheet or an external database.
- ◆ Subtotals: A row or column that displays subtotals for detail cells in a row or column in a pivot table. The pivot table in the figure displays subtotals for each branch.

Branch	CD	Checking	IRA	Savings	Grand Total
Central	736,289	158,980	63,380	261,749	1,220,398
North County	677,639	72,155	125,374	113,000	988,168
Westside	273,525	83,178	10,000	153,500	520,203
Grand Total	1,687,453	314,313	198,754	528,249	2,728,769

Figure 18-3: A pivot table that uses a page field

Data Appropriate for a Pivot Table

Not all data can be used to create a pivot table. The data that you summarize must be in the form of a database. You can store the database in either a worksheet (sometimes known as a list) or an external database file. Although Excel can generate a pivot table from any database, not all databases benefit.

Generally speaking, fields in a database table can consist of two types:

- ◆ Data: Contains a value or data to be summarized. In Figure 18-1, the Amount field is a data field.
- ◆ Category: Describes the data. In Figure 18-1, the Date, AcctType, OpenedBy, Branch, and Customer fields are category fields because they describe the data in the Amount field.

A single database table can have any number of data fields and category fields. When you create a pivot table, you usually want to summarize one or more of the data fields. Conversely, the values in the category fields appear in the pivot table as rows, columns, or pages.

Exceptions exist, however, and you may find Excel's pivot table feature useful even for databases that don't contain actual numerical data fields. The database in Figure 18-4, for example, doesn't contain any numerical data, but you can create a useful pivot table that counts the items in fields rather than sums them.

You can summarize information in a pivot table by using methods other than summing. For example, the pivot table that you see in Figure 18-5 cross-tabulates the Month Born field by the Sex field; the intersecting cells show the count for each combination of month and gender.

	A	B	C
1	Employee	Month Born	Sex
2	Anthony Taylor	July	Male
3	Charles S. Billings	February	Male
4	Christine Poundsworth	January	Female
5	Clark Bickerson	February	Male
6	Douglas Williams	March	Male
7	Janet Silberstein	April	Female
8	James Millen	May	Male
9	Jeffrey P. Jones	June	Male
10	Joan Morrison	July	Female
11	John T. Foster	August	Male
12	Kurt Kamichoff	January	Male
13	Michael Hayden	February	Male
14	Phyllis Todd	March	Female
15	Richard E. Card	April	Male
16	Rick Fogerty	May	Male
17	Robert H. Miller	June	Male
18	Stephen C. Carter	July	Male
19	Steven H. Katz	August	Male
20	Thomas F. Abbott	February	Male

Figure 18-4: This database doesn't have any numerical fields, but you can use it to generate a pivot table.

	A	B	C	D	E
1					
2					
3	Count of Employee	Sex			
4	Month Born	Female	Male	Grand Total	
5	January	1	3	4	
6	February	0	6	6	
7	March	3	1	4	
8	April	1	3	4	
9	May	1	4	5	
10	June	1	4	5	
11	July	1	5	6	
12	August	1	4	5	
13	September	0	5	5	
14	October	1	4	5	
15	November	1	4	5	
16	December	1	5	6	
17	Grand Total	12	48	60	
18					
19					

Figure 18-5: This pivot table summarizes non-numeric fields by displaying a count rather than a sum.

Creating a Pivot Table

You create a pivot table using a series of steps presented in the PivotTable and PivotChart Wizard. You access this wizard by choosing Data → PivotTable and PivotChart Report. Then, carry out the steps outlined here.



This discussion assumes you use Excel 2000 or later. The procedure differs slightly in earlier versions of Excel.

Step 1: Specifying the Data Location

When you choose Data → PivotTable and PivotChart Report, you'll see the dialog box shown in Figure 18-6.



Figure 18-6: The first of three PivotTable and PivotChart Wizard dialog boxes

In this step, you identify the data source. Excel is quite flexible in the data that you can use for a pivot table. (See the sidebar, “Pivot Table Data Sources.”) This example uses a worksheet database.



You see different dialog boxes while you work through the wizard, depending on the location of the data that you want to analyze. The following sections present the wizard's dialog boxes for data located in an Excel list or database.

Pivot Table Data Sources

The data used in a pivot table can come from a variety of sources, including Excel databases or lists, data sources external to Excel, multiple tabled ranges, and other pivot tables. I describe these sources here.

Excel List or Database

Usually, the data that you analyze is stored in a worksheet database (also known as a list). Databases stored in a worksheet have a limit of 65,535 records and 256 fields. Working with a database of this size isn't efficient, however (and memory may not even permit it). The first row in the database should contain field names. No other rules exist. The data can consist of values, text, or formulas.

External Data Source

If you use the data in an external database for a pivot table, use Query (a separate application) to retrieve the data. You can use dBASE files, SQL Server data, or other data that your system is set up to access. Step 2 of the PivotTable and PivotChart Wizard prompts you for the data source. Note that in Excel 2000 or later, you also can create a pivot table from an OLAP (OnLine Analytical Processing) database.

Multiple Consolidation Ranges

You also can create a pivot table from multiple tables. This procedure is equivalent to consolidating the information in tables. When you create a pivot table to consolidate information in tables, you have the added advantage of using all of the pivot table tools while working with the consolidated data.

Another Pivot Table

Excel enables you to create a pivot table from an existing pivot table. Actually, this is a bit of a misnomer. The pivot table that you create is based on the *data* that the first pivot table uses (not the pivot table itself). If the active workbook has no pivot tables, this option is grayed — meaning you can't choose it. If you need to create more than one pivot table from the same set of data, the procedure is more efficient (in terms of memory usage) if you create the first pivot table and then use that pivot table as the source for subsequent pivot tables.

Step 2: Specifying the Data

To move on to the next step of the wizard, click the Next button. Step 2 of the PivotTable and PivotChart Wizard prompts you for the data. Remember, the dialog box varies depending on your choice in the first dialog box; Figure 18-7 shows the dialog box that appears when you select an Excel list or database in Step 1.

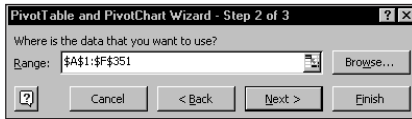


Figure 18-7: In Step 2, you specify the data range.

If you place the cell pointer anywhere within the worksheet database when you select Data → PivotTable Report, Excel identifies the database range automatically in Step 2 of the PivotTable and PivotChart Wizard.

You can use the Browse button to open a different worksheet and select a range. To move on to Step 3, click the Next button.



If the source range for a pivot table is named *Database*, you can use Excel's built-in Data Form to add new data to the range. The named range will extend automatically to include the new records.

Step 3: Completing the Pivot Table

Figure 18-8 shows the dialog box for the final step of the PivotTable and PivotChart Wizard. In this step, you specify the location for the pivot table.



Figure 18-8: In Step 3, you specify the pivot table's location.

If you select the New worksheet option, Excel inserts a new worksheet for the pivot table. If you select the Existing worksheet option, the pivot table appears on the current worksheet (you can specify the starting cell location).

At this point, you can click the Options button to select some options that determine how the table appears. (Refer to the sidebar “Pivot Table Options.”) You can set these options at any time after you create the pivot table, so you do not need to do so before creating the pivot table.

You can set up the actual layout of the pivot table by using either of two techniques:

- ◆ By clicking the Layout button in Step 3 of the PivotTable and PivotChart Wizard. You then can use a dialog box to lay out the pivot table.
- ◆ By clicking the Finish button to create a blank pivot table. You then can use the PivotTable Field List toolbar to lay out the pivot table.

I describe both of these options in the following subsections.

USING A DIALOG BOX TO LAY OUT A PIVOT TABLE

When you click the Layout button of the wizard's last dialog box, you get the dialog box shown in Figure 18-9. The fields in the database appear as buttons along the right side of the dialog box. Simply drag the buttons to the appropriate area of the pivot table diagram (which appears in the center of the dialog box).

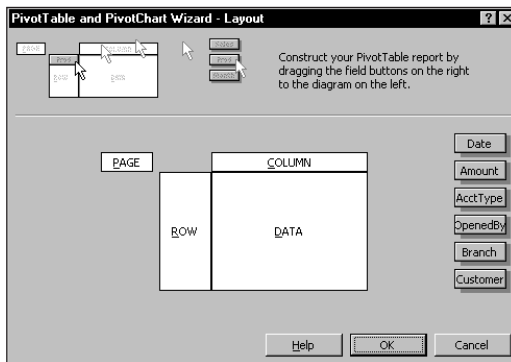


Figure 18-9: Specify the table layout



For versions prior to Excel 2000, this dialog box appears as Step 3 of the wizard. For these versions, this is the only way to lay out a pivot table.

The pivot table diagram has four areas:

- ◆ Page: Values in the field appear as page items in the pivot table.
- ◆ Row: Values in the field appear as row items in the pivot table.
- ◆ Data: The field is summarized in the pivot table.
- ◆ Column: Values in the field appear as column items in the pivot table.

You can drag as many field buttons as you want to any of these locations, and you don't have to use all the fields. Any fields that you don't use simply don't appear in the pivot table.

When you drag a field button to the Data area, the PivotTable and PivotChart Wizard applies the Sum function if the field contains numeric values; it applies the Count function if the field contains non-numeric values.

While you set up the pivot table, you can double-click a field button to customize it. You can specify, for example, to summarize a particular field as a Count or other function. You also can specify which items in a field to hide or omit. If you drag a field button to an incorrect location, just drag it off the table diagram to get rid of it. Note that you can customize fields at any time after you create the pivot table; I demonstrate this later in the chapter.

Figure 18-10 shows how the dialog box looks after dragging some field buttons to the pivot table diagram. This pivot table displays the sum of the Amount field, broken down by AcctType (as rows) and Customer (as columns). In addition, the Branch field appears as a page field. Click OK to redisplay the PivotTable and PivotChart Wizard – Step 3 of the dialog box.

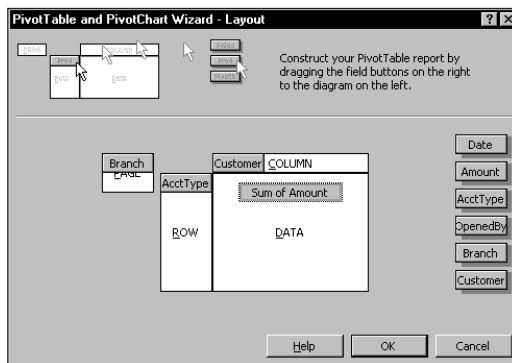


Figure 18-10: The table layout after dragging field buttons to the pivot table diagram

USING THE PIVOTTABLE FIELD LIST TOOLBAR TO LAY OUT A PIVOT TABLE

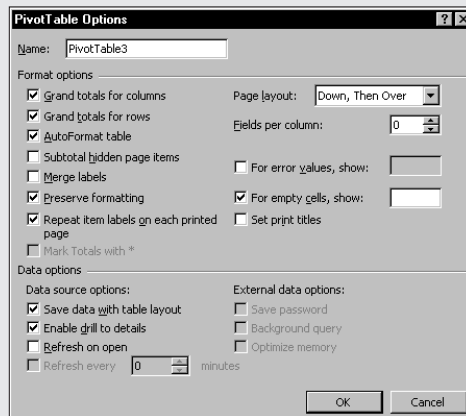
You may prefer to lay out your pivot table directly in the worksheet, using the PivotTable Field List toolbar. The technique closely resembles the one just described, because you still drag and drop fields. But in this case, you drag fields from the toolbar into the worksheet.



You cannot use this technique with versions prior to Excel 2000. Also, note that Excel 2000 doesn't have a PivotTable Field List toolbar. Rather, the fields are displayed as buttons on the PivotTable toolbar.

Pivot Table Options

Excel provides plenty of options that determine how your pivot table looks and works. To access these options, click the Options button in the final step of the PivotTable and PivotChart Wizard to display the PivotTable Options dialog box. You also can access this dialog box after you create the pivot table. Right-click any cell in the pivot table and then select Table Options from the shortcut menu. The accompanying figure shows the PivotTable Options dialog box. Following, I list its choices:



- ◆ **Name:** You can provide a name for the pivot table. Excel provides default names in the form of PivotTable1, PivotTable2, and so on.
- ◆ **Grand totals for columns:** Check this box if you want Excel to calculate grand totals for items displayed in columns.
- ◆ **Grand totals for rows:** Check this box if you want Excel to calculate grand totals for items displayed in rows.
- ◆ **AutoFormat table:** Check this box if you want Excel to apply one of its AutoFormats to the pivot table. Excel uses the AutoFormat even if you rearrange the table layout.
- ◆ **Subtotal hidden page items:** Check this box if you want Excel to include hidden items in the page fields in the subtotals.
- ◆ **Merge labels:** Check this box if you want Excel to merge the cells for outer row and column labels. Doing so may make the table more readable.
- ◆ **Preserve formatting:** Check this box if you want Excel, when it updates the pivot table, to keep any of the formatting that you applied.

- ◆ Repeat item labels on each printed page: Check this box to set row titles that appear on each page when you print a PivotTable report.
- ◆ Mark Totals with *: Available only if you generated the pivot table from an OLAP data source. If checked, displays an asterisk after every subtotal and grand total to indicate that these values include any hidden items as well as displayed items.
- ◆ Page layout: You can specify the order in which you want the page fields to appear.
- ◆ Fields per column: You can specify the number of page fields to show before starting another row of page fields.
- ◆ For error values, show: You can specify a value to show for pivot table cells that display an error.
- ◆ For empty cells, show: You can specify a value to show for empty pivot table cells.
- ◆ Set print titles: Check this box to set column titles that appear at the top of each page when you print a PivotTable report.
- ◆ Save data with table layout: If you check this option, Excel stores an additional copy of the data (called a *pivot table cache*), enabling Excel to recalculate the table more quickly when you change the layout. If memory is an issue, you should keep this option unchecked (which slows updating a bit).
- ◆ Enable drill to details: If checked, you can double-click a cell in the pivot table to view the records that contributed to the summary value.
- ◆ Refresh on open: If checked, the pivot table refreshes whenever you open the workbook.
- ◆ Refresh every x minutes: If you are connected to an external database, you can specify how often you want the pivot table refreshed while the workbook is open.
- ◆ Save password: If you use an external database that requires a password, you can store the password as part of the query so that you don't have to reenter it.
- ◆ Background query: If checked, Excel runs the external database query in the background while you continue your work.
- ◆ Optimize memory: This option reduces the amount of memory used when you refresh an external database query.

Complete the first two steps of the PivotTable and PivotChart Wizard. If you want, set options for the pivot table by using the Options button that appears in the third dialog box of the wizard. Don't bother with the Layout button, however. Select a location for the pivot table and choose Finish. Excel displays a pivot table template similar to the one you see in Figure 18-11. The template provides you with hints about where to drop various types of fields.

Drag and drop fields from the PivotTable Field List toolbar onto the template. Or select the field name, choose the location from the drop-down list, and click the Add To button. Excel continues to update the pivot table as you add or remove fields. For this reason, you'll find this method easiest to use if you drag and drop *data* items last. In other words, set up the field items, then specify the data to summarize.

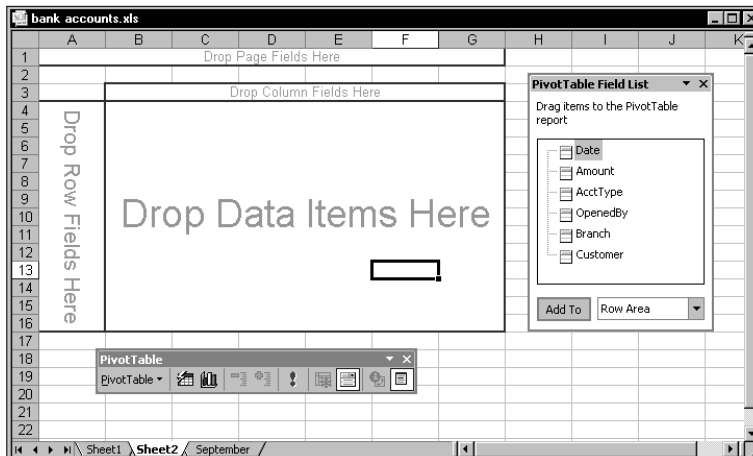
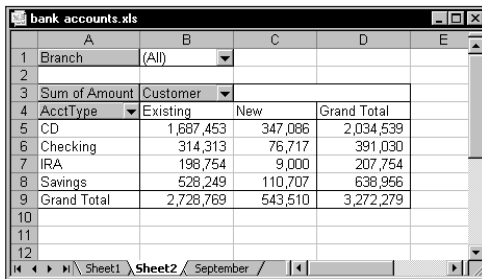


Figure 18-11: Use the PivotTable Field List toolbar to drag and drop fields onto the pivot table template that Excel displays.

If you make a mistake, simply drag the field off the template and drop it on the worksheet – Excel removes it from the pivot table template. All fields remain on the PivotTable Field List toolbar, even if you use them.

THE FINISHED PRODUCT

Figure 18-12 shows the result of this example. Notice that the page field displays as a drop-down box. You can choose which item in the page field to display by choosing it from the list. You also can choose an item called All, which displays all the data.



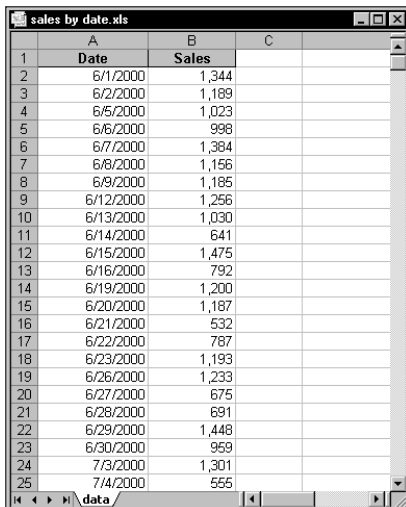
Branch	(All)			
Sum of Amount	Customer			
AcctType	Existing	New	Grand Total	
CD	1,687,453	347,086	2,034,539	
Checking	314,313	76,717	391,030	
IRA	198,754	9,000	207,754	
Savings	528,249	110,707	638,956	
Grand Total	2,728,769	543,510	3,272,279	

Figure 18-12: The pivot table created by the PivotTable and PivotChart Wizard

Grouping Pivot Table Items

One of the more useful features of a pivot table is the ability to combine items into groups. To group objects, select them, right-click, and choose Group and Outline → Group from the shortcut menu.

When a field contains dates, Excel can create groups automatically. Figure 18-13 shows a simple database table with two fields: Date and Sales. This table has 370 records and covers dates between June 1, 2000 and November 1, 2001. The goal is to summarize the sales information by month.



	A	B	C
1	Date	Sales	
2	6/1/2000	1,344	
3	6/2/2000	1,189	
4	6/5/2000	1,023	
5	6/6/2000	998	
6	6/7/2000	1,384	
7	6/8/2000	1,156	
8	6/9/2000	1,185	
9	6/12/2000	1,256	
10	6/13/2000	1,030	
11	6/14/2000	641	
12	6/15/2000	1,475	
13	6/16/2000	792	
14	6/19/2000	1,200	
15	6/20/2000	1,187	
16	6/21/2000	532	
17	6/22/2000	787	
18	6/23/2000	1,193	
19	6/26/2000	1,233	
20	6/27/2000	675	
21	6/28/2000	691	
22	6/29/2000	1,448	
23	6/30/2000	959	
24	7/3/2000	1,301	
25	7/4/2000	555	

Figure 18-13: You can use a pivot table to summarize the sales data by month.

Figure 18-14 shows a pivot table created from the data. Not surprisingly, it looks exactly like the input data because the dates have not been grouped. To group the items by month, right-click the Data heading and select Group and Show Detail → Group. You'll see the Grouping dialog box shown in Figure 18-15.

Copying a Pivot Table

A pivot table is a special type of *object*, and you cannot manipulate it as you may expect. For example, you can't insert a new row or enter formulas within the pivot table. If you want to manipulate a pivot table in ways not normally permitted, make a copy of it.

To copy a pivot table, select the table and choose Edit → Copy. Then activate a new worksheet and choose Edit → Paste Special. Select the Values option and click OK. The contents of the pivot table are copied to the new location so you can do whatever you like to them. You also might want to repeat the Edit → Paste Special command and select Formats (to copy the formatting from the pivot table).

This technique is also useful when you want to create a standard chart. If you attempt to create a chart from a pivot table, Excel will always create a pivot chart that contains field buttons. Sometimes you may prefer a standard chart.

Note that the copied information is no longer linked to the source data. If the source data changes, your copied pivot table does not reflect these changes.



In versions prior to Excel 2002, the shortcut menu command is Group and Outline → Group.

The screenshot shows an Excel spreadsheet window titled "sales by date.xls". The pivot table is located in the range A3:B22. The pivot table has two columns: "Date" and "Total". The data is as follows:

Sum of Sales	
Date	Total
6/1/2000	1344
6/2/2000	1189
6/5/2000	1023
6/6/2000	998
6/7/2000	1384
6/8/2000	1156
6/9/2000	1185
6/12/2000	1256
6/13/2000	1030
6/14/2000	641
6/15/2000	1475
6/16/2000	792
6/19/2000	1200
6/20/2000	1167
6/21/2000	532
6/22/2000	787
6/23/2000	1193
6/26/2000	1233

Figure 18-14: The pivot table, before grouping by month



Figure 18-15: Use the Grouping dialog box to group items in a pivot table.

In the list box, select Months and Years, and verify that the starting and ending dates are correct. Click OK. The Date items in the pivot table are grouped by years and by months (see Figure 18-16).

Sum of Sales		
Years	Date	Total
2000	Jun	23378
	Jul	21354
	Aug	21913
	Sep	22188
	Oct	22087
	Nov	21198
2001	Dec	21186
	Jan	22739
	Feb	21842
	Mar	24554
	Apr	20875
	May	23814
	Jun	19311
	Jul	20738
	Aug	23226
	Sep	20596
	Oct	21911
Grand Total		372890

Figure 18-16: The pivot table, after grouping by month



If you select only Months in the Grouping list box, months in different years combine together. For example, the June item would display sales for both 2000 and 2001.

Creating a Calculated Field or Calculated Item

Once you create a pivot table, you can create two types of *formulas* for further analysis:

- ◆ A calculated field: A new field created from other fields in the pivot table. A calculated field must reside in the Data area of the pivot table (you can't use a calculated field in the Page, Row, or Column areas).
- ◆ A calculated item: A calculated item uses the contents of other items within a field of the pivot table. A calculated item must reside in the Page, Row, or Column area of a pivot table (you can't use a calculated item in the Data area).

The formulas used to create calculated fields and calculated items are not standard Excel formulas. In other words, you do not enter the formulas into cells. Rather, you enter these formulas in a dialog box, and they are stored along with the pivot table data.



Beginning with Excel 2000, you can use an OLAP database as the source for a pivot table. You can't, however, create calculated fields or items in a pivot table based on an OLAP database.

The examples in this section use the worksheet database table shown in Figure 18-17. The table consists of five fields and 48 records. Each record describes monthly sales information for a particular sales representative. For example, Amy is a sales rep for the North region, and she sold 239 units in January for total sales of \$23,040.

Figure 18-18 shows the basic pivot table created from the data. This pivot table shows sales, broken down by month and sales rep.

The examples that follow will create:

- ◆ A calculated field, to compute average sales per unit
- ◆ A calculated item, to summarize the data by quarters

	A	B	C	D	E	F
1	SalesRep	Region	Month	Sales	Units Sold	
2	Amy	North	Jan	\$23,040	239	
3	Amy	North	Feb	\$24,131	79	
4	Amy	North	Mar	\$24,646	71	
5	Amy	North	Apr	\$22,047	71	
6	Amy	North	May	\$24,971	157	
7	Amy	North	Jun	\$24,218	92	
8	Amy	North	Jul	\$25,735	175	
9	Amy	North	Aug	\$23,638	87	
10	Amy	North	Sep	\$25,749	557	
11	Amy	North	Oct	\$24,437	95	
12	Amy	North	Nov	\$25,355	706	
13	Amy	North	Dec	\$25,899	180	
14	Bob	North	Jan	\$20,024	103	
15	Bob	North	Feb	\$23,822	267	
16	Bob	North	Mar	\$24,854	96	
17	Bob	North	Apr	\$22,838	74	
18	Bob	North	May	\$25,320	231	
19	Bob	North	Jun	\$24,733	164	
20	Bob	North	Jul	\$21,184	68	
21	Bob	North	Aug	\$23,174	114	

Figure 18-17: This data demonstrates calculated fields and calculated items.

	A	B	C	D	E	F	G
1	Region	(All)					
2							
3	Sum of Sales	SalesRep					
4	Month	Amy	Bob	Chuck	Doug	Grand Total	
5	Jan	23,040	20,024	19,886	26,264	89,214	
6	Feb	24,131	23,822	23,494	29,953	101,400	
7	Mar	24,646	24,854	21,824	25,041	96,365	
8	Apr	22,047	22,838	22,058	29,338	96,281	
9	May	24,971	25,320	20,280	25,150	95,721	
10	Jun	24,218	24,733	23,965	27,371	100,287	
11	Jul	25,735	21,184	23,032	25,044	94,995	
12	Aug	23,638	23,174	21,273	29,506	97,591	
13	Sep	25,749	25,999	21,584	29,061	102,393	
14	Oct	24,437	22,639	19,625	27,113	93,814	
15	Nov	25,355	23,949	19,832	25,953	95,089	
16	Dec	25,899	23,179	20,583	28,670	98,331	
17	Grand Total	293,866	281,715	257,436	328,464	1,161,481	
18							
19							

Figure 18-18: This pivot table was created from the data in Figure 18-17.

Creating a Calculated Field in a Pivot Table

Because a pivot table is a special type of data range, you can't insert new rows or columns within the pivot table. This means that you can't insert formulas to perform calculations with the data in a pivot table. However, you can create calculated fields for a pivot table. A *calculated field* consists of a calculation that can involve other fields.

A calculated field is basically a way to display new information in a pivot table. It essentially presents an alternative to creating a new *Data* field in your source database. A calculated field cannot be used as a Row, Column, or Page field.

In the sales example, for instance, suppose you want to calculate the average sales amount per unit. You can compute this value by dividing the Sales field by the Units Sold field. The result shows a new field (a calculated field) for the pivot table.

Use the following procedure to create a calculated field that consists of the Sales field divided by the Units Sold field:

1. Move the cell pointer anywhere within the pivot table.
2. Using the Pivot Table toolbar, choose PivotTable → Formulas → Calculated Field. Excel displays the Insert Calculated Field dialog box.
3. Enter a descriptive name in the Name field and specify the formula in the Formula field (see Figure 18-19). The formula can use other fields and worksheet functions. For this example, the calculated field name is Avg Unit Price, and the formula appears as the following:

```
=Sales/'Units Sold'
```

4. Click Add to add this new field.
5. Click OK to close the Insert Calculated Field dialog box.

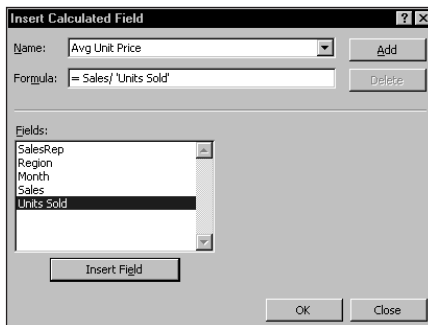


Figure 18-19: The Insert Calculated Field dialog box



You can create the formula manually by typing it, or by double-clicking items in the Fields list box. Double-clicking an item transfers it to the Formula field. Because the Units Sold field contains a space, Excel adds single quotes around the field name.

After you create the calculated field, Excel adds it to the Data area of the pivot table. You can treat it just like any other field, with one exception: You can't move it to the Page, Row, or Column area (it must remain in the Data area).

Figure 18-20 shows the pivot table after you've added the calculated field. The new field displays a Sum of Avg Unit Price (you can change this text, if desired, by editing any of the cells in which that text appears). The calculated field also appears on the PivotTable Field List toolbar, along with the other fields available for use in the pivot table.

Region	(All)						
		SalesRep					
Month	Data	Amy	Bob	Chuck	Doug	Grand Total	
Jan	Sum of Sales	23,040	20,024	19,886	26,264	89,214	
	Sum of Avg Unit Price	96	194	209	285	169	
Feb	Sum of Sales	24,131	23,822	23,494	29,953	101,400	
	Sum of Avg Unit Price	305	89	159	35	75	
Mar	Sum of Sales	24,646	24,854	21,824	25,041	96,365	
	Sum of Avg Unit Price	347	259	263	291	267	
Apr	Sum of Sales	22,047	22,838	22,058	29,338	96,281	
	Sum of Avg Unit Price	311	309	230	132	208	
May	Sum of Sales	24,971	25,320	20,280	25,150	95,721	
	Sum of Avg Unit Price	159	110	45	104	88	
Jun	Sum of Sales	24,218	24,733	23,965	27,371	100,267	
	Sum of Avg Unit Price	263	151	32	288	90	
Jul	Sum of Sales	25,735	21,184	23,032	25,044	94,995	
	Sum of Avg Unit Price	147	312	149	305	198	
Aug	Sum of Sales	23,638	23,174	21,273	29,506	97,591	
	Sum of Avg Unit Price	272	203	28	286	91	
Sep	Sum of Sales	25,749	25,999	21,584	29,061	102,393	
	Sum of Avg Unit Price	46	310	189	199	114	
Oct	Sum of Sales	24,437	22,639	19,625	27,113	93,814	
	Sum of Avg Unit Price	257	87	236	226	168	
Nov	Sum of Sales	25,355	23,949	19,832	25,953	95,069	
	Sum of Avg Unit Price	36	220	283	320	98	
Dec	Sum of Sales	25,899	23,179	20,583	28,670	98,331	
	Sum of Avg Unit Price	144	50	116	145	96	
Total	Sum of Sales	293,866	281,715	257,436	328,464	1,161,481	
Total	Sum of Avg Unit Price	117	138	86	142	118	

Figure 18-20: This pivot table uses a calculated field.



The formulas that you develop can also use worksheet functions, but the functions cannot refer to cells or named ranges.

Inserting a Calculated Item into a Pivot Table

The previous section describes how to create a calculated field. Excel also enables you to create a *calculated item* for a pivot table field. The sales example uses a field named Month, which consists of text strings. You can create a calculated item (called Qtr-1, for example) that displays the sum of Jan, Feb, and Mar.

You also can do this by grouping the items, but using grouping hides the individual months and shows only the total of the group. Creating a calculated item for quarterly totals is more flexible because it shows the total and the individual months.

To create a calculated item to sum the data for Jan, Feb, and Mar, use these steps:

1. Move the cell pointer to the Row, Column, or Page area of the pivot table that contains the item that will be calculated. In this example, the cell pointer should be in the Month area.
2. Use the Pivot Table toolbar, and choose PivotTable → Formulas → Calculated Item from the shortcut menu. Excel displays the Insert Calculated Item dialog box.
3. Enter a name for the new item in the Name field and specify the formula in the Formula field (see Figure 18-21). The formula can use items in other fields, but it can't use worksheet functions. For this example, the new item is named Qtr-1, and the formula appears as follows:

$$=Jan+Feb+Mar$$
4. Click Add.
5. Repeat Steps 3 and 4 to create additional calculated items for Qtr-2 (=Apr+May+Jun), Qtr-3 (=Jul+Aug+Sep), and Qtr-4 (=Oct+Nov+Dec).
6. Click OK to close the dialog box.

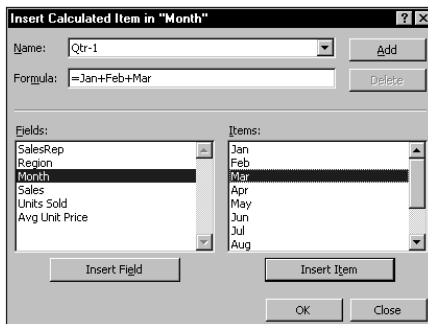


Figure 18-21: The Insert Calculated Item dialog box



If you use a calculated item in your pivot table, you may need to turn off the Grand Total display to avoid double counting.

After you create the items, they appear in the pivot table. Figure 18-22 shows the pivot table after you've added the four calculated items. Notice that the calculated items are added to the end of the Month items. You can rearrange the items by selecting and dragging. Figure 18-23 shows the pivot table after rearranging the items logically. (I also made the calculated items bold.)

Region	(All)					
Sum of Sales	SalesRep					
Month	Amy	Bob	Chuck	Doug	Grand Total	
Jan	23,040	20,024	19,886	26,264	89,214	
Feb	24,131	23,822	23,494	29,953	101,400	
Mar	24,646	24,854	21,824	25,041	96,365	
Apr	22,047	22,838	22,058	29,338	96,281	
May	24,971	25,320	20,280	25,150	95,721	
Jun	24,218	24,733	23,965	27,371	100,287	
Jul	25,735	21,184	23,032	25,044	94,995	
Aug	23,638	23,174	21,273	29,506	97,591	
Sep	25,749	25,999	21,584	29,061	102,393	
Oct	24,437	22,639	19,625	27,113	93,814	
Nov	25,355	23,949	19,832	25,953	95,089	
Dec	25,899	23,179	20,583	26,670	98,331	
Qtr-1	71,817	68,700	65,204	81,258	286,979	
Qtr-2	71,236	72,891	66,303	81,859	292,289	
Qtr-3	75,122	70,357	65,889	83,611	294,979	
Qtr-4	75,691	69,767	60,040	81,736	287,234	

Figure 18-22: This pivot table uses calculated items for quarterly totals.

Region	(All)					
Sum of Sales	SalesRep					
Month	Amy	Bob	Chuck	Doug	Grand Total	
Jan	23,040	20,024	19,886	26,264	89,214	
Feb	24,131	23,822	23,494	29,953	101,400	
Mar	24,646	24,854	21,824	25,041	96,365	
Qtr-1	71,817	68,700	65,204	81,258	286,979	
Apr	22,047	22,838	22,058	29,338	96,281	
May	24,971	25,320	20,280	25,150	95,721	
Jun	24,218	24,733	23,965	27,371	100,287	
Qtr-2	71,236	72,891	66,303	81,859	292,289	
Jul	25,735	21,184	23,032	25,044	94,995	
Aug	23,638	23,174	21,273	29,506	97,591	
Sep	25,749	25,999	21,584	29,061	102,393	
Qtr-3	75,122	70,357	65,889	83,611	294,979	
Oct	24,437	22,639	19,625	27,113	93,814	
Nov	25,355	23,949	19,832	25,953	95,089	
Dec	25,899	23,179	20,583	26,670	98,331	
Qtr-4	75,691	69,767	60,040	81,736	287,234	

Figure 18-23: The pivot table, after rearranging the calculated items



A calculated item appears in a pivot table only if the field on which it is based also appears. If you remove or pivot a field from either the Row or Column category into the Data category, the calculated item does not appear.

Summary

This chapter presented an introduction to pivot tables and demonstrated how to create a pivot table, group items, and create calculated fields and calculated items. A pivot table often provides an excellent alternative to creating formulas for summarizing a database.

The next chapter discusses the use of conditional formatting and data validation.

Chapter 19

Conditional Formatting and Data Validation

IN THIS CHAPTER

- ◆ An overview of Excel's conditional formatting feature
- ◆ Practical examples of using conditional formatting formulas
- ◆ An overview of Excel's data validation feature
- ◆ Practical examples of using data validation formulas

THIS CHAPTER EXPLORES TWO VERY useful Excel features: conditional formatting and data validation. You may not think these features have much to do with formulas. But as you'll see, when you toss formulas into the mix, these features can perform some amazing feats.



Excel 97 introduced conditional formatting and data validation. Therefore, this chapter does not apply if you use an earlier version of Excel.

Conditional Formatting

Conditional formatting enables you to apply cell formatting selectively and automatically, based on the contents of the cells. For example, you can set things up such that all negative values in a range have a light yellow background color. When you enter or change a value in the range, Excel examines the value and evaluates the conditional formatting rules for the cell. If the value is negative, the background is shaded. If not, no formatting is applied.

Conditional formatting is very useful for quickly identifying erroneous cell entries, or cells of a particular type. You can use a format (such as bright red cell shading) to make particular cells easy to identify.

Is this a handy feature? No doubt. But dig a little deeper and you'll see that a lot more lurks in the shadows, and this feature can do things you may not have thought possible. The key, as you'll see, is specifying your conditions by using formulas. In this section, I describe Excel's conditional formatting feature and point out some of its limitations – as well as a potentially serious design flaw.

Specifying Conditional Formatting

To apply conditional formatting to a cell or range:

1. Select the cell or range.
2. Choose Format → Conditional Formatting. Excel displays its Conditional Formatting dialog box, shown in Figure 19-1.

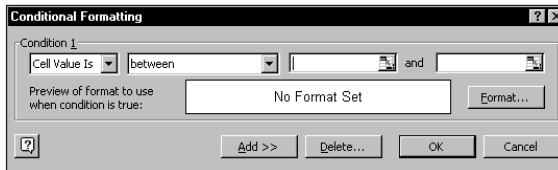


Figure 19-1: The Conditional Formatting dialog box

3. In the drop-down box, select either Cell Value Is (for simple conditional formatting), or Formula Is (for formatting based on a formula).
4. Specify the condition (or enter a formula).
5. Click the Format button and specify the formatting to apply if the condition is TRUE.
6. To add additional conditions (up to two more), click Add and then repeat steps 3 through 5.
7. Click OK.

After you've performed these steps, the cell or range will be formatted based on the condition(s) you specify. This formatting, of course, is dynamic: If you change the contents of a cell, Excel reevaluates the new contents and applies or removes the formatting accordingly.

Formatting Types You Can Apply

When you click the Format button in the Conditional Formatting dialog box, you get the Format Cells dialog box shown in Figure 19-2. This is a modified version of the standard Format Cells dialog box – it does not have the Number, Alignment, and Protection tabs, but it includes a Clear button. You can specify any of the following formats:

- ◆ Font style (regular, bold, or italic)
- ◆ Font underline
- ◆ Font color
- ◆ Font strikethrough
- ◆ Border outline
- ◆ Border line style
- ◆ Border line color
- ◆ Cell shading color
- ◆ Cell background pattern

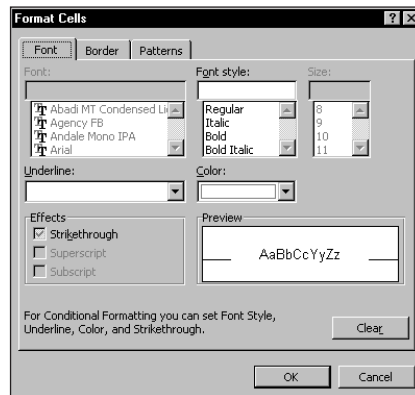


Figure 19-2: The Format Cells dialog box used in conditional formatting

Notice that you can't specify the font or font size; presumably, this is because the font size can affect row heights. The designers probably decided that changing row heights automatically could be distracting, or introduce other problems such as pagination when printing.



The colors available in the Format Cells dialog box are the 56 colors in the workbook's color palette. If none of these colors is satisfactory, you can modify the workbook's color palette. To do so, select Tools → Options, and click the Color tab in the Options dialog box. Select a color and click the Modify button to change the color. But exercise caution, because changing a color may affect other color formatting in your workbook.



The Find and Replace dialog box in Excel 2002 allows you to search your worksheet to locate cells that contain specific formatting. This feature does not locate cells that contain formatting resulting from conditional formatting.

Specifying Conditions

The leftmost drop-down list in the Conditional Formatting dialog box enables you to choose one of two options:

- ◆ Cell Value Is: For simple conditions
- ◆ Formula Is: For more complex, formula-based conditions

I discuss these two types of conditions in the sections that follow.

SIMPLE CONDITIONS

When you select Cell Value Is, you can specify conditions of the following types:

- ◆ between (you specify two values)
- ◆ not between (you specify two values)
- ◆ equal to (you specify one value)
- ◆ not equal to (you specify one value)
- ◆ greater than (you specify one value)
- ◆ less than (you specify one value)
- ◆ greater than or equal to (you specify one value)
- ◆ less than or equal to (you specify one value)

You can either enter the value(s) directly, or specify a cell reference.

FORMULA-BASED CONDITIONS

When you select Formula Is, you can specify a formula. Do so by specifying a cell that contains a formula, or by entering a formula directly into the dialog box (see Figure 19-3).

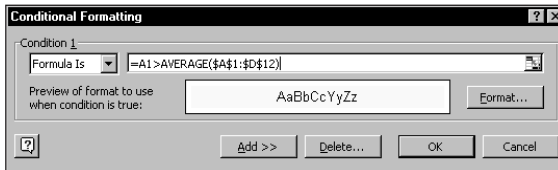


Figure 19-3: Entering a formula directly into the Conditional Formatting dialog box



You must specify a logical formula that returns either TRUE or FALSE. If the formula evaluates to TRUE, the condition is satisfied and the conditional formatting is applied. If the formula evaluates to FALSE, the conditional formatting is not applied.

As you'll see by studying the examples later in this chapter, the real power of conditional formatting is apparent when you enter a formula directly into the Conditional Formatting dialog box.

If the formula that you enter into the Conditional Formatting dialog box contains a cell reference, that reference is considered a relative reference, based on the upper left cell in the selected range. For example, suppose you want to set up a conditional formatting condition that applies shading to blank cells in the range B2:B10. Follow these steps:

1. Select the range B2:B10.
2. Choose Format → Conditional Formatting.
3. Select the Formula Is item from the drop-down list.
4. Enter the following formula in the formula box:
`=B2=""`
5. Click the Format button and specify a pattern for the cell shading.
6. Click OK twice.

Changing Font Color Using Custom Number Formats

In some cases, you can avoid conditional formatting and take advantage of a custom number format that changes font color conditionally. For example, the custom number format that follows displays positive values in black, negative values in red, and zero values in blue:

```
[Black]General;[Red]-General;[Blue]General
```

For more information about creating custom number formats, refer to Appendix C.

Notice that the formula entered contains a reference to the upper left cell in the selected range. To demonstrate that the reference is relative, select cell B5 and examine its conditional formatting formula. You'll see that the conditional formatting formula for this cell is:

```
=B5=""
```

Generally, when entering a conditional formatting formula for a range of cells, you'll use a reference to the upper left cell in the selected range. One exception: when you need to refer to a specific cell. For example, suppose you select range A1:B20, and you want to apply formatting to all cells in the range that exceed the value in cell C1. Enter this conditional formatting formula:

```
=A1>$C$1
```

In this case, the reference to cell C1 is an absolute reference; it will not be adjusted for the cells in the selected range. In other words, the conditional formatting formula for cell A2 looks like this:

```
=A2>$C$1
```

The relative cell reference is adjusted, but the absolute cell reference is not.

Working with Conditional Formats

This section describes some additional information about conditional formatting that you might find useful.

MULTIPLE CONDITIONS

As noted previously, you can specify as many as three conditions by clicking the Add button in the Conditional Formatting dialog box. For example, you might enter the following three conditions (and specify different formatting for each):

Cell Value Is less than 0
Cell Value Is equal to 0
Cell Value Is greater than 0

In this case, the sign of the value (negative, 0, or positive) determines the applied formatting.

If none of the specified conditions is TRUE, the cells keep their existing formats. If you specify multiple conditions and more than one condition is TRUE, Excel applies only the formatting for the first TRUE condition. For example, you may specify the following two conditions:

Cell Value Is between 1 and 12
Cell Value Is less than 6

Entering a value of 4 satisfies both conditions. Therefore, the cell will be formatted using the format specified for the first condition.

BE CAREFUL WHEN PASTING

It's important to keep in mind that it's very easy (too easy) to wipe out the conditional formatting in a cell or range by pasting copied data to the cell.



Copying a cell and pasting it to a cell or range that contains conditional formatting wipes out the conditional formatting in the destination range. You get no warning. This, of course, is a serious design flaw on the part of Microsoft—one that you should keep in mind if you use conditional formatting in your workbook.

COPYING CELLS THAT CONTAIN CONDITIONAL FORMATTING

Conditional formatting information is stored with a cell much like standard formatting information is stored with a cell. This means that when you copy a cell that contains conditional formatting, the conditional formatting is also copied.



To copy only the conditional formats, select cells you want to format and include at least one cell in the selection that has the conditional formats you want to copy. Select **Format** → **Conditional Formatting** and then click **OK**.

Inserting rows or columns within a range that contains conditional formatting causes the new cells to have the same conditional formatting.

DELETING CONDITIONAL FORMATTING

When you press Del to delete the contents of a cell, you do not delete the conditional formatting for the cell (if any). To remove all conditional formats (as well as all other cell formatting), select the cells and choose Edit → Clear → Formats.

To remove only conditional formatting (and leave the other formatting intact), you need to use the Conditional Formatting dialog box. Select the cells, then choose Format → Conditional Formatting. Click the Delete button in the Conditional Formatting dialog box and you get another dialog box (see Figure 19-4) that enables you to specify the conditions that you want to delete. This dialog box always displays check boxes for three conditions, even if you haven't defined that many.

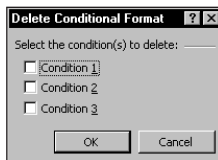


Figure 19-4: Use the Delete Conditional Format dialog box to remove one or more conditions.



You also can remove conditional formatting from a cell by simply copying a cell that *doesn't* have conditional formatting and then pasting it to the cell or range. This, of course, also copies the cell's value (or formula) as well as other formatting.

LOCATING CELLS THAT CONTAIN CONDITIONAL FORMATTING

You cannot tell, just by looking at a cell, whether it contains conditional formatting. You can, however, use Excel's Go To dialog box to select such cells.

Select Edit → Go To (or press F5) to display the Go To dialog box. Click the Special button, and then select the Conditional formats option (see Figure 19-5). To select all cells on the worksheet containing conditional formatting, select the All option. To select only the cells that contain the same conditional formatting as the active cell, select the Same option. Click OK and the cells will be selected for you.

USING REFERENCES TO OTHER SHEETS

If you enter a conditional formatting formula that uses one or more references to other sheets, Excel responds with an error message. If you need to refer to a cell on a different sheet, you must create a reference to that cell on the sheet that contains the conditional formatting. For example, if your conditional formatting formula needs to refer to cell A1 on Sheet3, you can insert the following formula into a cell on the active sheet.

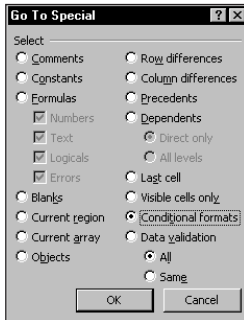


Figure 19–5: Use the Go To Special dialog box to locate cells that contain conditional formatting.

=Sheet3!A1

Then, use a reference to that cell in your conditional formatting formula.



Another option is to create a name for the cell (by using Insert → Name → Define). After defining the name, you can use the name in place of the cell reference in the Conditional Formatting dialog box. If you use this technique, the named cell can be in any worksheet in the workbook.

Conditional Formatting Formulas

This section contains a number of examples that demonstrate various uses for conditional formatting. Each of these examples uses a formula entered directly into the Conditional Formatting dialog box. You decide the type of formatting that you apply conditionally.



You can access all of the examples in this section on the companion CD-ROM.

IDENTIFYING NONNUMERIC DATA

The following conditional formatting formula applies formatting to cell A1 only if the cell contains text:

=ISTEXT(A1)

To apply this conditional formatting formula to a range, select the range first. The argument for the ISTEXT function should be the upper left cell in the range.

IDENTIFYING ABOVE-AVERAGE CELLS

I applied the following conditional formatting formula to range A1:D12. It applies formatting to all cells in the range A1:D12 that are above the average (see Figure 19-6):

```
=A1>AVERAGE($A$1:$D$12)
```

	A	B	C	D	E
1	1	12	23	34	
2	36	13	24	35	
3	3	14	25	2	
4	4	15	26	37	
5	5	16	27	38	
6	41	17	28	6	
7	7	18	12	40	
8	8	19	30	41	
9	9	20	31	42	
10	10	43	32	21	
11	11	22	33	44	
12	29	23	34	45	
13					
14					

Figure 19-6: Using conditional formatting to highlight all above-average cells

Notice that the first cell reference (A1) is a relative reference, but the range argument for the AVERAGE formula is absolute.

IDENTIFYING DATES IN A PARTICULAR MONTH

Conditional formatting also works with dates. The conditional formatting formula that follows applies formatting only if the cell contains a date in the month of June:

```
=MONTH(A1)=6
```

This formula assumes that cell A1 is the upper left cell in the selected range. It works by using the MONTH function, which returns the month number for a date.



The MONTH function does not distinguish between dates and nondates. In other words, the MONTH function is applied to all cells, even if they don't contain a date.

IDENTIFYING TODAY'S DATE

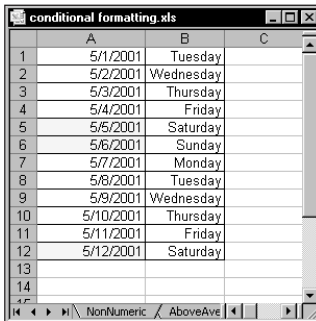
Excel's TODAY function returns the current date. If you have a series of dates in a worksheet, you can use conditional formatting to make it easy to identify data for the current date. The conditional formatting formula that follows applies formatting only if the cell contains the current date. This assumes that you selected a range beginning with cell A1 when you entered the conditional formatting formula.

```
=A1=TODAY()
```

IDENTIFYING WEEKEND DATES

Excel's WEEKDAY function returns an integer that represents the day of the week (1 is Sunday, 2 is Monday, and so on). You can use this function in a custom formatting formula to identify weekends. The following custom formatting formula applies formatting to cells that contain a date that falls on a Saturday or Sunday (see Figure 19-7):

```
=OR(WEEKDAY(A1)=7,WEEKDAY(A1)=1)
```



	A	B	C
1	5/1/2001	Tuesday	
2	5/2/2001	Wednesday	
3	5/3/2001	Thursday	
4	5/4/2001	Friday	
5	5/5/2001	Saturday	
6	5/6/2001	Sunday	
7	5/7/2001	Monday	
8	5/8/2001	Tuesday	
9	5/9/2001	Wednesday	
10	5/10/2001	Thursday	
11	5/11/2001	Friday	
12	5/12/2001	Saturday	
13			
14			

Figure 19-7: Using conditional formatting to highlight cells that contain a weekend date

This formula uses the OR function, so it returns TRUE if the WEEKDAY function returns either 7 or 1. You'll find that the WEEKDAY function returns 7 if its argument is an empty cell. Therefore, if your range contains empty cells, you should use this formula:

```
=IF(ISBLANK(A1),"",OR(WEEKDAY(A1)=7,WEEKDAY(A1)=1))
```


HIDING ERROR VALUES

You can use conditional formatting to *hide* error values in your cells. In this case, hiding the contents of a cell consists of setting its font color equal to its background color. The following conditional formatting formula applies formatting to the cell if it returns an error value (for example, #DIV/0!):

```
=ISERROR(A1)
```

The applied formatting sets the font color to the background color.



Although setting the background color equal to the font color technique works, it's usually not the best way to handle the display of error values. Cells that reference the erroneous cell display an error, and the user easily can change the background color. In many cases, a better approach is to use an IF function that displays an empty string if the formula returns an error. The following formula displays an empty string if B1/C1 generates an error:

```
=IF(ISERR(B1/C1), "", (B1/C1))
```



A new feature in Excel 2002 lets you specify how cell error values are printed. You can choose to print errors as blanks, dashes, or #N/A. You control this in the Sheet tab of the Page Setup dialog box.

IDENTIFYING THE MAXIMUM VALUE IN A RANGE

Excel's MAX function returns the maximum value in a range. If you want to make this value stand out, you can use a conditional formatting formula such as this one:

```
=A1=MAX($A$1:$A$30)
```

In this case, the conditional formatting is applied to all cells in A1:A30, and the maximum value in that range will be formatted. You can, of course, modify this formula to use the MIN function (which returns the smallest value in a range).

IDENTIFYING THE THREE LARGEST VALUES IN A RANGE

Excel's LARGE function returns the *n*th largest value in a range (*n* is specified as the second argument). The following conditional formatting formula applies formatting to the three largest values in the range A1:A30: This formula returns TRUE for cells that are greater than or equal to the third largest value in the range.

```
=A1>=LARGE($A$1:$A$30,3)
```

DISPLAYING ALTERNATE ROW SHADING

The conditional formatting formula that follows was applied to the range A1:D18, shown in Figure 19-8, to apply shading to alternate rows. This formula is quite useful for making your spreadsheets easier to read.

```
=MOD(ROW(),2)=0
```

	A	B	C	D	E
1	410	425	808	972	
2	506	568	329	792	
3	511	24	399	266	
4	202	271	849	546	
5	867	737	69	412	
6	913	497	115	261	
7	331	3	664	13	
8	363	606	604	409	
9	923	711	467	193	
10	93	371	751	92	
11	227	805	695	345	
12	32	426	459	281	
13	601	390	228	455	
14	962	649	84	134	
15	327	942	872	152	
16	92	377	756	165	
17	762	219	960	189	
18	778	665	645	504	
19					

Figure 19-8: Using conditional formatting to apply formatting to alternate rows

This formula uses the ROW function (which returns the row number) and the MOD function (which returns the remainder of its first argument divided by its second argument). For cells in even-numbered rows, the MOD function returns 0, and cells in that row are formatted. For alternate shading of columns, use the COLUMN function instead of the ROW function.

You can use variations on this conditional formatting formula to get other types of row shading. For example, the conditional formatting formula that follows shades every third row:

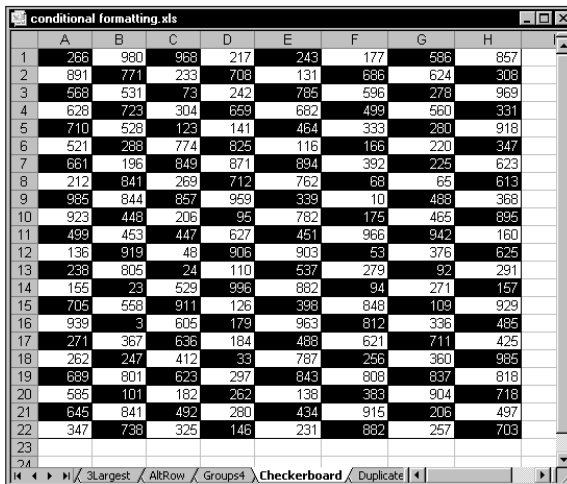
```
=MOD(ROW(),3)=0
```

The following conditional formatting formula applies alternate shading in groups of four rows (four rows shaded, followed by four rows not shaded):

```
=MOD(INT((ROW()-1)/4)+1,2)
```

Need checkerboard shading, as shown in Figure 19-9? This conditional formatting formula does just that:

```
=MOD(ROW(),2)=MOD(COLUMN(),2)
```



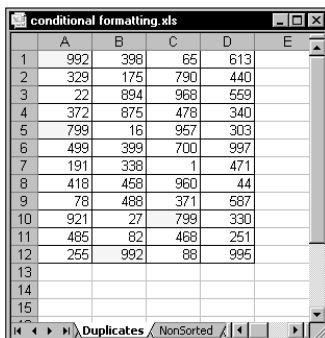
	A	B	C	D	E	F	G	H
1	266	980	968	217	243	177	586	857
2	891	771	233	708	131	686	624	308
3	568	531	73	242	785	596	278	969
4	628	723	304	659	682	499	560	331
5	710	528	123	141	464	333	280	918
6	521	288	774	826	116	166	220	347
7	661	196	849	871	894	392	225	623
8	212	841	269	712	762	68	65	613
9	985	844	857	959	339	10	488	368
10	923	448	206	95	782	175	465	895
11	499	453	447	627	451	966	942	160
12	136	919	48	906	903	53	376	625
13	236	805	24	110	537	279	92	291
14	155	23	529	996	882	94	271	157
15	705	558	911	126	398	848	109	929
16	939	3	605	179	963	812	336	485
17	271	367	636	184	468	621	711	425
18	262	247	412	33	787	256	360	985
19	689	801	623	297	843	808	837	818
20	585	101	182	262	138	383	904	718
21	645	841	492	280	434	915	206	497
22	347	738	325	146	231	882	257	703
23								
24								

Figure 19-9: Using conditional formatting to create a checkerboard effect

IDENTIFYING DUPLICATE VALUES IN A RANGE

You might find it helpful to identify duplicate values within a range (see Figure 19-10). You can use a conditional formatting formula such as the one that follows. In this case, formatting is applied to all cells that are not unique within the range A1:D12.

```
=COUNTIF($A$1:$D$12,A1)>1
```



	A	B	C	D	E
1	992	398	65	613	
2	329	175	790	440	
3	22	894	968	559	
4	372	875	478	340	
5	799	16	957	303	
6	499	399	700	997	
7	191	338	1	471	
8	418	458	960	44	
9	78	488	371	587	
10	921	27	799	330	
11	485	82	468	251	
12	255	992	88	995	
13					
14					
15					

Figure 19-10: Using conditional formatting to identify duplicate values in a range

To apply formatting only to nonduplicated values in a range, use a formula such as this:

```
=COUNTIF($A$1:$D$12,A1)=1
```

IDENTIFYING NONSORTED VALUES IN A RANGE

If you have a single-column range of values that should be in ascending order, you can use a conditional formatting formula to quickly spot values that are out of order. This example assumes that your sorted values begin in cell A1. Select the range of values beginning with A2, and then specify the following conditional formatting formula:

```
=A2<A1
```

Conditional formatting will be applied to any cell that is less than the cell above it.

IDENTIFYING UPWARD OR DOWNWARD TRENDS

In some cases, you might find it helpful to visually identify upward or downward trends in a column of data. This example assumes that the data begins in cell A1. You need to select the range beginning in A2 and then specify two conditions, as follows.

```
=A2>A1
```

```
=A2<A1
```

Specify a different format for each condition, so you can spot the trends without creating a chart. Figure 19-11 shows an example.

	A	B	C	D	E	F	G
1	100						
2	109						
3	106						
4	108						
5	112						
6	114						
7	112						
8	116						
9	123						
10	123						
11	124						
12	128						
13	127						
14	132						
15	136						
16	140						
17	141						
18	139						
19	139						
20	143						
21	151						
22	152						
23	149						
24	153						
25	157						
26	166						
27	165						
28	166						

Figure 19-11: Use conditional formatting to identify upward and downward trends.

IDENTIFYING CELLS CONTAINING MORE THAN ONE WORD

You also can use conditional formatting with text. For example, you can use the following conditional formatting formula to apply formatting to cells that contain more than one word:

```
=LEN(TRIM(A1))-LEN(SUBSTITUTE(A1," ",""))>0
```

This formula assumes that the selected range begins in cell A1. The formula works by counting the space characters in the cell (using the TRIM function to strip out multiple spaces). If the count is greater than 1, the formula returns TRUE and the conditional formatting is applied.

IDENTIFYING CELLS CONTAINING A SPECIFIC CHARACTER

The conditional formatting formula that follows applies formatting to cells (beginning in cell A1) that contain the letter A (either upper- or lowercase):

```
=LEN(A1)-LEN(SUBSTITUTE(LOWER(A1),"a",""))>0
```

DISPLAYING A RESULT ONLY WHEN ALL DATA IS ENTERED

This example uses conditional formatting to display a result only when you have entered all the necessary data. In Figure 19-12, a formula in cell B5 calculates the sum of the four values above. The objective is to hide the total until you enter all four values.

	A	B	C
1	Value 1:	36	
2	Value 2:	55	
3	Value 3:		
4	Value 4:	23	
5			
6			
7			
8			

Figure 19-12: Conditional formatting hides the contents of A5:B5 unless you enter a value for each cell in B1:B4.

Select A5:B5 and format these cells so the font color matches the background color—for example, make the font color white. This effectively makes these two cells invisible. With A5:B5 still selected, enter the following conditional formatting formula:

```
=COUNT($B$1:$B$4)=4
```

This formula returns TRUE only when all of the cells in B1:B4 are not empty. Specify the conditioning of your choice. For example, you can make the background color black. Figure 19-13 shows the result when you have entered all of the required data.

	A	B	C
1	Value 1:	36	
2	Value 2:	55	
3	Value 3:	32	
4	Value 4:	23	
5	TOTAL:	146	
6			
7			
8			

Figure 19-13: The contents of A5:B5 are visible only when all cells in B1:B4 contain data.

IDENTIFYING POSITIVE CHANGES

Figure 19-14 shows data for a group of students who took two tests. Conditional formatting is used to highlight the rows in which the students' post-test scores were higher than their pre-test scores.

	A	B	C	D
1	Student	Pre-test	Post-test	
2	Anna	89	93	
3	Bill	91	90	
4	Chris	75	81	
5	Darla	65	73	
6	Ernie	90	81	
7	Frank	93	100	
8	George	89	89	
9	Hilda	99	98	
10	Ishmael	54	69	
11	James	60	75	
12	Keith	89	88	
13				
14				
15				

Figure 19-14: Using conditional formatting to identify students who scored higher on the post-test

The conditional formatting formula for the range A2:C12 is:

```
= $C2 > $B2
```

Notice that this formula uses mixed references. The column part is absolute, but the row part is relative.

Using Custom Functions in Conditional Formatting Formulas

Conditional formatting formulas also work with custom worksheet functions created using VBA. This section provides four examples.



Part V provides an overview of VBA, with specific information about creating custom worksheet functions.

IDENTIFYING FORMULA CELLS

Oddly, Excel does not have a function that determines whether a cell contains a formula. When Excel lacks a feature, you often can overcome the limitation by using VBA. The following VBA function uses VBA's HasFormula property. The function, which is entered into a VBA module, returns TRUE if the cell (specified as its argument) contains a formula; otherwise, it returns FALSE.

```
Function ISFORMULACELL(cell) As Boolean
    ISFORMULACELL = cell.HasFormula
End Function
```

After you enter this function into a VBA module, you can use the function in your worksheet formulas. For example, the following formula returns TRUE if cell A1 contains a formula:

```
=ISFORMULACELL(A1)
```

And, you also can use this function in a conditional formatting formula. The worksheet in Figure 19-15, for example, uses conditional formatting to highlight all cells that contain a formula.



Another way to identify formula cells is to use the Edit → Go To command. This command displays the Go To dialog box. Click the Special button to display the Go To Special dialog box. Then choose the Formulas option and click OK. This will select all cells that contain a formula.

	A	B	C	D	E
1		Last Year	This Year	Difference	
2	Jan	143	155	12	
3	Feb	155	188	33	
4	Mar	133	122	-11	
5	Q1	431	465	34	
6	Apr	160	178	18	
7	May	187	203	16	
8	Jun	199	221	22	
9	Q2	546	602	56	
10	Jul	201	273	72	
11	Aug	177	212	35	
12	Sep	191	198	7	
13	Q3	569	683	114	
14	Oct	244	255	11	
15	Nov	199	188	-11	
16	Dec	211	233	22	
17	Q4	654	676	22	
18	Total	2200	2426	226	
19					
20					
21					

Figure 19-15: Using a custom VBA function to apply conditional formatting to cells that contain a formula

IDENTIFYING DATE CELLS

Excel also lacks a function to determine whether a cell contains a date. The following VBA function, which uses VBA's `IsDate` function, overcomes this limitation. The custom `HASDATE` function returns `TRUE` if the cell contains a date.

```
Function HASDATE(cell) As Boolean
    HASDATE = IsDate(cell)
End Function
```

You can use this function to improve the conditional formatting formulas presented earlier in this chapter (see “Identifying Dates in a Particular Month” and “Identifying Weekends”). Neither of the conditional formatting formulas presented could distinguish between cells that contain a date and cells that contain a normal value. You can use the `AND` function to ensure that the formatting applies only to date cells.

The following conditional formatting formula applies formatting to cell A1 if it contains a date and the month is June:

```
=AND(HASDATE(A1),MONTH(A1)=6)
```

The following conditional formatting formula applies formatting to cell A1 if it contains a date and the date falls on a weekend:

```
=AND(HASDATE(A1),OR(WEEKDAY(A1)=7,WEEKDAY(A1)=1))
```


IDENTIFYING LINK FORMULAS

You may want to identify cells that contain a link formula (a formula that uses a reference in a different workbook). The following VBA function returns TRUE if the cell contains a formula that contains an external link. The HASLINK function uses VBA's versatile Like operator to determine whether a formula contains a set of square brackets.

```
Function HASLINK(cell)
    If cell.HasFormula Then
        HASLINK = cell.Formula Like "*[[]*"
    Else
        HASLINK = False
    End If
End Function
```

To apply conditional formatting to cells that contain a link, you can create a conditional formatting formula such as the following:

```
=HASLINK(A1)
```



The HASLINK function is not perfect. In some cases it will falsely identify a formula as being a linked formula. For example, the following formula contains a set of square brackets, but it is not a linked formula. The HASLINK function, however, reports otherwise.

```
="["&A1&"]"
```

IDENTIFYING INVALID DATA

You might have a situation in which the data entered must adhere to some very specific rules, and you'd like to apply special formatting if the data entered is not valid. You might have part numbers that consist of seven characters: four uppercase alphabetic characters, followed by a hyphen, and then a two-digit number. For example: ADSS-09 or DYUU-43.

You can write a conditional formatting formula to determine if part numbers adhere to this structure, but the formula is very complex. The following formula, for example, returns TRUE only if the value in A1 meets the part number rules specified:

```
=AND(LEN(A1)=7,AND(LEFT(A1)>="A",LEFT(A1)<="Z"),
AND(MID(A1,2,1)>="A",MID(A1,2,1)<="Z"),AND(MID(A1,3,1)>="A",
MID(A1,3,1)<="Z"),AND(MID(A1,4,1)>="A",MID(A1,4,1)<="Z"),
MID(A1,5,1)="-",AND(VALUE(MID(A1,6,2))>=0,
VALUE(MID(A1,6,2))<=99))
```

For a simpler approach, write a custom VBA worksheet function. VBA's Like operator makes this sort of comparison relatively easy. The following VBA function procedure returns TRUE if its argument does not correspond to the part number rules outlined previously:

```
Function INVALIDPART(n) As Boolean
    If n Like "[A-Z][A-Z][A-Z][A-Z]-##" Then
        INVALIDPART = False
    Else
        INVALIDPART = True
    End If
End Function
```

After defining this function in a VBA module, you can enter the following conditional formatting formula to apply special formatting if cell A1 contains an invalid part number:

```
=INVALIDPART(A1)
```

Figure 19-16 shows a range that uses the INVALIDPART function in a conditional formatting formula. Cells that contain invalid part numbers have a colored background.

In many cases, you can simply take advantage of Excel's data validation feature – which is described next.

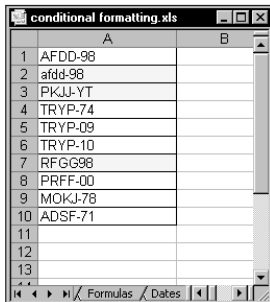


Figure 19-16: Using conditional formatting to highlight cells with invalid entries

Data Validation

The data validation feature, available in Excel 97 and later versions, is similar in many respects to the conditional formatting feature. This feature enables you to set up certain rules that dictate what you can enter into a cell. For example, you may

want to limit data entry to whole numbers between 1 and 12. If the user makes an invalid entry, you can display a custom message such as the one shown in Figure 19-17.

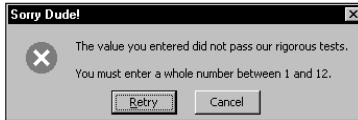


Figure 19-17: Displaying a message when the user makes an invalid entry

As with the conditional formatting feature, you can use a formula to specify your data validation criteria.



The data validation feature suffers the same problem as conditional formatting: If the user copies a cell and pastes it to a cell that contains data validation, the data validation rules are deleted. Consequently, the cell then accepts any type of data.

Specifying Validation Criteria

To specify the type of data allowable in a cell or range:

1. Select the cell or range.
2. Choose Data → Validation. Excel displays its Data Validation dialog box.
3. Click the Settings tab (see Figure 19-18).

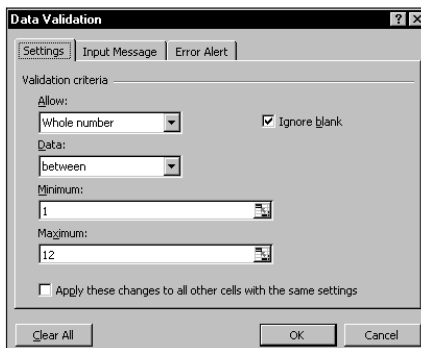


Figure 19-18: The Settings tab of the Data Validation dialog box

4. Choose an option from the drop-down box labeled Allow. To specify a formula, select Custom.
5. Specify the conditions by selecting from the drop-down box labeled Data. Your selection determines what other controls you can access.
6. Click the Input Message tab (see Figure 19-19) and specify which message to display when a user selects the cell. You can use this optional step to tell the user what type of data is expected.

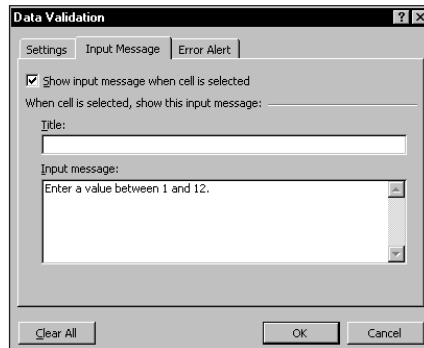


Figure 19-19: The Input Message tab of the Data Validation dialog box

7. Click the Error Alert tab (see Figure 19-20) and specify which error message to display when a user makes an invalid entry. The selection for Style determines what choices users have when they make invalid entries. To prevent an invalid entry, choose Stop. This step is optional.
8. Click OK.

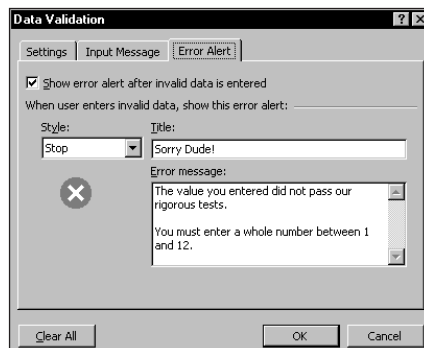


Figure 19-20: The Error Alert tab of the Data Validation dialog box

After you've performed these steps, the cell or range contains the validation criteria you specified.

Types of Validation Criteria You Can Apply

The Settings tab of the Data Validation dialog box enables you to specify any of the following data validation criteria:

- ◆ **Any value:** Selecting this option removes any existing data validation. Note, however, that the input message, if any, still displays if the check box is checked in the Input Message tab.
- ◆ **Whole number:** The user must enter a whole number. You specify a valid range of whole numbers by using the Data drop-down list. For example, you can specify that the entry must be a whole number greater than or equal to 100.
- ◆ **Decimal:** The user must enter a number. You specify a valid range of numbers by using the Data drop-down list. For example, you can specify that the entry must be greater than or equal to 0, and less than or equal to 1.
- ◆ **List:** The user must choose from a list of entries you provide. Specify the range that contains the list using the Source control (the range must be a single row or column). If you have a short list, you can enter it directly into the Source control (separate each item with list separator specified in your regional settings – a comma if you use the U.S. regional settings).



If you specify a range for a list, it must be on the same sheet. If your list is in a range on a different worksheet, you can provide a name for the range and then use the name as your list source (preceded by an equal sign). For example, if the list is contained in a range named *MyList*, enter the following:

```
=MyList
```

- ◆ **Date:** The user must enter a date. You specify a valid date range by using the Data drop-down list. For example, you can specify that the entered data must be greater than or equal to January 1, 2001, and less than or equal to December 31, 2001.
- ◆ **Time:** The user must enter a time. You specify a valid time range by using the Data drop-down list. For example, you can specify that the entered data must be greater than 12:00 PM.
- ◆ **Text length:** The length of the data (number of characters) is limited. You specify a valid length by using the Data drop-down list. For example, you can specify that the length of the entered data be 1 (a single alphanumeric character).

- ◆ Custom: A logical formula determines the validity of the user's entry. You can enter the formula directly into the Formula control, or specify a cell reference that contains a formula. This chapter contains examples of useful formulas.

The Settings tab of the Data Validation dialog box contains two other options:

- ◆ Ignore blank: If checked, blank entries are allowed.
- ◆ Apply these changes to all other cells with the same setting: If checked, the changes you make apply to all other cells that contain the original data validation criteria.

It's important to understand that, even with data validation in effect, the user could enter invalid data. If the Style setting in the Error Alert tab of the Data Validation dialog box is set to anything except Stop, invalid data *can* be entered.



After your data is entered, you can look for entries that are outside the limits you set. When you click Circle Invalid Data on the Formula Auditing toolbar, circles appear around cells that contain incorrect entries (see Figure 19-21). If you correct an invalid entry, the circle disappears.

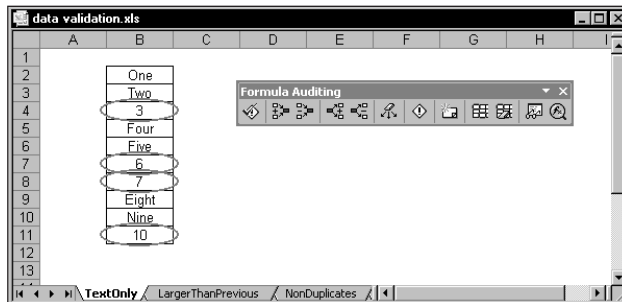


Figure 19-21: Circles are drawn around invalid entries (cells that contain a value).

Using Formulas for Data Validation Rules

For simple data validation, the data validation feature is quite straightforward and easy to use. But the real power of this feature becomes apparent when you use data validation formulas.



The formula that you specify must be a logical formula that returns either TRUE or FALSE. If the formula evaluates to TRUE, the data is considered valid and remains in the cell. If the formula evaluates to FALSE, a message box appears that displays the message specified in the Error Alert tab of the Data Validation dialog box.

As noted earlier, you specify a formula in the Data Validation dialog box by selecting the Custom option in the Allow drop-down list of the Settings tab. You can enter the formula directly into the Formula control, or enter a reference to a cell that contains a formula.

If the formula that you enter contains a cell reference, that reference will be considered to be a relative reference, based on the upper left cell in the selected range. This works exactly the same as using a formula for conditional formatting (see “Formula-Based Conditions,” earlier in this chapter).

Using Data Validation Formulas to Accept Only Specific Entries

Each of the following data validation examples uses a formula entered directly into the Data Validation dialog box. You can set up these formulas to accept only text, a certain value, nonduplicate entries, or text that begins with a specific letter.



All of the examples in this section are available on the companion CD-ROM.

ACCEPTING TEXT ONLY

To force a range to accept only text (no values), use the following data validation formula:

```
=ISTEXT(A1)
```

This formula assumes that the upper left cell in the selected range is cell A1.

ACCEPTING A LARGER VALUE THAN THE PREVIOUS CELL

The following data validation formula allows the user to enter a value only if it's greater than the value in the cell directly above it:

```
=A2>A1
```

This formula assumes that A2 is the upper left cell in the selected range. Note that you can't use this formula for a cell in row 1.

ACCEPTING NONDUPLICATE ENTRIES ONLY

The following data validation formula does not permit the user to make a duplicate entry in the range A1:C20:

```
=COUNTIF($A$1:$C$20,A1)=1
```

This formula assumes that A1 is the upper left cell in the selected range. Note that the first argument for COUNTIF is an absolute reference. The second argument is a relative reference, and it adjusts for each cell in the validation range. Figure 19-22 shows this validation criteria in effect, using a custom error alert message.

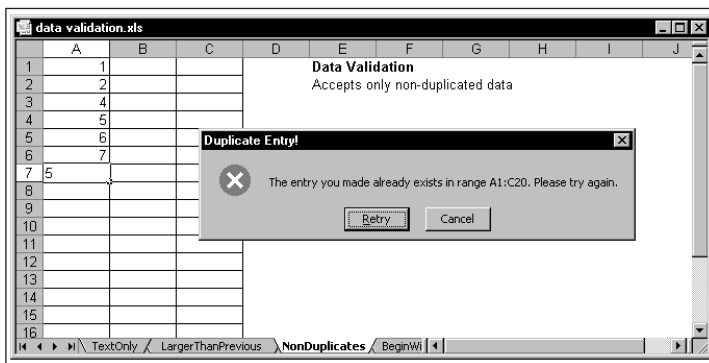


Figure 19-22: Using data validation to prevent duplicate entries in a range

ACCEPTING TEXT THAT BEGINS WITH “A”

The following data validation formula demonstrates how to check for a specific character. In this case, the formula ensures that the user's entry is a text string that begins with the letter A (either upper- or lowercase).

```
=LEFT(A1)="a "
```

This formula assumes that the upper left cell in the selected range is cell A1.

The following formula is a variation of this validation formula. In this case, the formula ensures that the entry begins with the letter A and contains exactly five characters.

```
=COUNTIF(A1,"A????")=1
```


Using Custom Worksheet Functions in Data Validation Formulas

Earlier in this chapter, I described how to use custom VBA functions for custom formatting (see "Using Custom Functions in Conditional Formatting Formulas"). For some reason, Excel does not permit you to use a custom VBA function in a data validation formula. If you attempt to do so, you get the following (erroneous) error message: *A named range you specified cannot be found.*

To bypass this limitation, you can use the custom function in a cell formula, and then specify a data validation formula that refers to that cell.



Part VI of this book covers custom VBA functions.

Summary

This chapter provided an overview of two useful features available in Excel 97 or later: conditional formatting and data validation. It also provided many examples of using formulas in conjunction with these features.

The next chapter covers a concept that I call megaformulas.

Chapter 20

Creating Megaformulas

IN THIS CHAPTER

- ◆ What is a megaformula, and why would you want to use such a thing?
- ◆ How to create a megaformula
- ◆ Examples of megaformulas
- ◆ Pros and cons of using megaformulas

THIS CHAPTER DESCRIBES A USEFUL TECHNIQUE that lets you combine several formulas into a single formula – what I call a *megaformula*. This technique can eliminate intermediate formulas and may even speed up recalculation. The downside, as you'll see, is that the resulting formula is virtually incomprehensible and may be impossible to edit.

What Is a Megaformula?

Often, spreadsheets require intermediate formulas to produce a desired result. In other words, a formula may depend on other formulas, which in turn depend on other formulas. After you get all these formulas working correctly, you often can eliminate the intermediate formulas and create a single (and more complex) formula. For lack of a better term, I call such a formula a *megaformula*.

What are the advantages of employing megaformulas? They use fewer cells (less clutter), and recalculation may be faster. And, you can impress people in the know with your formula-building abilities. The disadvantages? The formula probably will be impossible to decipher or modify, even by the person who created it.

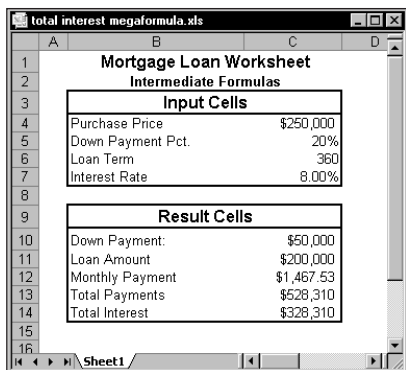


The techniques described in this chapter helped to create many of the complex formulas presented elsewhere in this book.

A limitation to the megaformula technique is that Excel formulas can contain no more than 1,024 characters.

Creating a Megaformula: A Simple Example

Creating a megaformula basically involves copying formula text and pasting it into another formula. Let's start with a relatively simple example. Examine the spreadsheet shown in Figure 20-1. This sheet uses formulas to calculate mortgage loan information.



Mortgage Loan Worksheet	
Intermediate Formulas	
Input Cells	
Purchase Price	\$250,000
Down Payment Pct.	20%
Loan Term	360
Interest Rate	8.00%
Result Cells	
Down Payment:	\$50,000
Loan Amount	\$200,000
Monthly Payment	\$1,467.53
Total Payments	\$528,310
Total Interest	\$328,310

Figure 20-1: This spreadsheet uses multiple formulas to calculate mortgage loan information.



This workbook is available on the companion CD-ROM.

The Result Cells section of the worksheet uses information entered into the Input Cells section and contains the formulas shown in Table 20-1.

TABLE 20-1 FORMULAS USED TO CALCULATE TOTAL INTEREST

Cell	Formula	What It Does
C10	=C4*C5	Calculates the down payment amount
C11	=C4-C10	Calculates the loan amount
C12	=PMT(C7/12,C6,-C11)	Calculates the monthly payment
C13	=C12*C6	Calculates the total payments
C14	=C13-C11	Calculates the total interest

Suppose you're *really* interested in the total interest paid (cell C14). You could, of course, simply hide the rows that contain the extraneous information. But, it's also possible to create a single formula that does the work of several intermediary formulas.

The formula that calculates total interest depends on the formulas in cells C11 and C13 (these are the direct precedent cells). In addition, the formula in cell C13 depends on the formula in cell C12. And cell C12, in turn, depends on cell C11. Therefore, calculating the total interest uses five formulas. The steps that follow describe how to create a single formula to calculate total interest so you can eliminate the intermediate formulas. C14 contains the following formula:

```
=C13-C11
```

The steps that follow describe how to convert this formula into a megaformula:

1. Substitute the formula contained in cell C13 for the reference to cell C13. Before doing this, add parentheses around the formula in C13. Now the formula in C14 is:

```
=(C12*C6)-C11
```

2. Substitute the formula contained in cell C12 for the reference to cell C12. Now the formula in C14 is:

```
=(PMT(C7/12,C6,-C11)*C6)-C11
```

3. Substitute the formula contained in cell C11 for the two references to cell C11. Before copying the formula, you need to insert parentheses around it. Now the formula in C14 is:

```
=(PMT(C7/12,C6,-(C4-C10))*C6)-(C4-C10)
```

4. Substitute the formula contained in C10 for the two references to cell C10. Before copying the formula, insert parentheses around it. After you've done so, the formula in C14 is:

```
=(PMT(C7/12,C6,-(C4-(C4*C5)))*C6)-(C4-(C4*C5))
```

At this point, the formula contains references only to input cells. You can safely delete the formulas in C10:C13. The single megaformula now does the work previously performed by the intermediary formulas.

Unless you're a world-class Excel formula wizard, it's quite unlikely that you could arrive at that formula without first creating intermediate formulas.



I designed this previous exercise to demonstrate how to create a megaformula. This technique is *not* the most efficient way to calculate total interest on a loan. Excel provides a more direct way to make that calculation: Use the CUMIPMT function contained in the Analysis ToolPak.

Copying Text from a Formula

Creating megaformulas involves copying formula text and then replacing a cell reference with the copied text. To copy the contents of a formula, activate the cell and press F2. Then select the formula text (without the equal sign) by pressing Shift+Home, followed by Shift+→. Then press Ctrl+C to copy the selected text to the clipboard. Activate the cell that contains the megaformula and press F2. Use the arrow keys, and hold down Shift to select the cell reference you want to replace. Finally, press Ctrl+V to replace the selected text with the clipboard contents.

In some cases, you need to insert parentheses around the copied formula text to make the formula calculate correctly. If the formula returns a different result after you paste the formula text, press Ctrl+Z to undo the paste. Insert parentheses around the formula you want to copy and try again.

Creating a megaformula essentially involves substituting formula text for cell references in a formula. You perform substitutions until the megaformula contains no references to formula cells. At each step along the way, you can check your work by ensuring that the formula continues to display the same result. In the previous example, a few of the steps required you to use parentheses around the copied formula.

Megaformula Examples

This section contains three additional examples of megaformulas. These examples provide a thorough introduction to applying the megaformula technique for streamlining a variety of tasks including cleaning up a list of names by removing middle names and initials, returning the position of the last space character in a string, and determining if a credit card number is valid.

Using a Megaformula to Remove Middle Names

Consider a worksheet with a column of people's names, like the one shown in Figure 20-2. Suppose you have a worksheet with thousands of such names, and you need to remove all the middle names and middle initials from the names. Editing the cells manually takes hours, and you're not up to writing a VBA macro. So that leaves a formula-based solution. Notice that not all the names have a middle name or initial, which makes the task a bit trickier. Although this is not a difficult task, it normally involves several intermediate formulas.

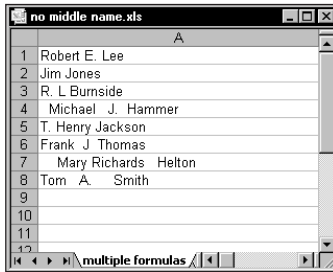


Figure 20-2: The goal is to remove the middle name or middle initial from each name.

Figure 20-3 shows the results of the more conventional solution, which requires six intermediate formulas as shown in Table 20-2. The names are in column A; column H displays the end result. Columns B through G hold the intermediate formulas.

	A	B	C	D	E	F	G	H
1	Robert E. Lee	Robert E. Lee	7	10	10	Robert	Lee	Robert Lee
2	Jim Jones	Jim Jones	4	#VALUE!	4	Jim	Jones	Jim Jones
3	R. L. Burnside	R. L. Burnside	3	5	5	R.	Burnside	R. Burnside
4	Michael J. Hammer	Michael J. Hammer	8	11	11	Michael	Hammer	Michael Hammer
5	T. Henry Jackson	T. Henry Jackson	3	9	9	T.	Jackson	T. Jackson
6	Frank J. Thomas	Frank J. Thomas	6	8	8	Frank	Thomas	Frank Thomas
7	Mary Richards Helton	Mary Richards Helton	5	14	14	Mary	Helton	Mary Helton
8	Tom A. Smith	Tom A. Smith	4	7	7	Tom	Smith	Tom Smith

Figure 20-3: Removing the middle names and initials requires six intermediate formulas.

TABLE 20-2 INTERMEDIATE FORMULAS IN THE FIRST ROW OF SHEET1 IN FIGURE 20-3

Cell	Intermediate Formula	What It Does
B1	=TRIM(A1)	Removes excess spaces
C1	=FIND(" ", B1, 1)	Locates the first space
D1	=FIND(" ", B1, C1+1)	Locates the second space, if any
E1	=IF(ISERROR(D1), C1, D1)	Uses the first space if no second space exists
F1	=LEFT(B1, C1-1)	Extracts the first name
G1	=RIGHT(B1, LEN(B1)-E1)	Extracts the last name
H1	=F1&" "&G1	Concatenates the two names



Notice that the result isn't perfect. For example, it will not work if the cell contains only one name (for example, Madonna). And, this method also fails if a name has two middle names (such as John Jacob Robert Smith). That occurs because the formula simply searches for the second space character in the name. In this example, the megaformula returns *John Robert Smith*. Later in this chapter, I present an array formula method to identify the last space character in a string.

With a bit of work, you can eliminate all the intermediate formulas and replace them with a single megaformula. You do so by creating all the intermediate formulas and then editing the final result formula (in this case, the formula in column H) by replacing each cell reference with a copy of the formula in the cell referred to. Fortunately, you can use the clipboard to copy and paste (see the previous sidebar, "Copying Text from a Formula"). Keep repeating this process until cell H1 contains nothing but references to cell A1. You end up with the following megaformula in one cell:

```
=LEFT(TRIM(A1),FIND(" ",TRIM(A1),1)-1)&" "&RIGHT
(TRIM(A1),LEN(TRIM(A1))-IF(ISERROR(FIND(" ",
TRIM(A1),FIND(" ",TRIM(A1),1)+1)),FIND(" ",TRIM(A1),1),
FIND(" ",TRIM(A1),FIND(" ",TRIM(A1),1)+1)))
```

When you're satisfied that the megaformula works, you can delete the columns that hold the intermediate formulas because they are no longer used.

THE STEP-BY-STEP PROCEDURE

If you're still not clear about this process, take a look at these step-by-step procedures:

1. Examine the formula in H1. This formula contains two cell references (F1 and G1):

```
=F1&" "&G1
```

2. Activate cell G1 and copy the contents of the formula (without the equal sign) to the clipboard.
3. Activate cell H1 and replace the reference to cell G1 with the clipboard contents. Now cell H1 contains the following formula:

```
=F1&" "&RIGHT(B1,LEN(B1)-E1)
```

4. Activate cell F1 and copy the contents of the formula (without the equal sign) to the clipboard.

5. Activate cell H1 and replace the reference to cell F1 with the clipboard contents. Now the formula in cell H1 is:

```
=LEFT(B1,C1-1)&" "&RIGHT(B1,LEN(B1)-E1)
```

6. Now cell H1 contains references to three cells (B1, C1, and E1). The formulas in those cells will replace each of the three references.

7. Replace the reference to cell E1 with the formula in E1. The result is:

```
=LEFT(B1,C1-1)&" "&RIGHT(B1,LEN(B1)-IF(ISERROR(D1),C1,D1))
```

8. Notice that the formula in cell H1 now contains two references to cell D1. Copy the formula from D1 and replace both of the references to cell D1. The formula now looks like this:

```
=LEFT(B1,C1-1)&" "&RIGHT(B1,LEN(B1)-IF(ISERROR(FIND(" ",B1,C1+1)),C1,FIND(" ",B1,C1+1)))
```

9. Replace the four references to cell C1 with the formula contained in cell C1. The formula in cell H1 is:

```
=LEFT(B1,FIND(" ",B1,1)-1)&" "&RIGHT(B1,LEN(B1)-IF(ISERROR(FIND(" ",B1,FIND(" ",B1,1)+1)),FIND(" ",B1,1),FIND(" ",B1,FIND(" ",B1,1)+1)))
```

10. Finally, replace the nine references to cell B1 with the formula in cell B1. The result is:

```
=LEFT(TRIM(A1),FIND(" ",TRIM(A1),1)-1)&" "&RIGHT(TRIM(A1),LEN(TRIM(A1))-IF(ISERROR(FIND(" ",TRIM(A1),FIND(" ",TRIM(A1),1)+1)),FIND(" ",TRIM(A1),1),FIND(" ",TRIM(A1),FIND(" ",TRIM(A1),1)+1)))
```

Notice that the formula in cell H1 now contains references only to cell A1. The megaformula is complete, and it performs exactly the same tasks as all the intermediate formulas (which you can now delete).



You can access the workbook for removing middle names and initials on the companion CD-ROM.

COMPARING SPEED AND EFFICIENCY

Because a megaformula is so complex, you may think that using one slows down recalculation. Actually, that's not the case. As a test, I created a workbook that used the megaformula 65,536 times. Then I created another workbook that used six

intermediate formulas to compute the 65,536 results. I compared the results with a custom VBA function that performs the same operation. Table 20-3 shows the statistics regarding the three methodologies.

TABLE 20-3 COMPARING INTERMEDIATE FORMULAS, A MEGAFORMULA, AND A VBA FUNCTION

Method	Recalculation Time (Seconds)	File Size
Intermediate formulas	10.8	24.4MB
Megaformula	6.2	8.9MB
VBA function	106.7	8.6MB

The actual results will of course vary depending on system speed and the amount of memory installed.

As you can see, using a megaformula in this case resulted in faster recalculations as well as a *much* smaller workbook. The VBA function was much slower—in fact, it wasn’t even in the same ballpark. This is fairly typical of VBA functions; they are always slower than built-in Excel functions.

Using a Megaformula to Return a String’s Last Space Character Position

As previously noted, the “remove middle name” example presented earlier contains a flaw: To identify the last name, the formula searches for the second space character. A better solution is to search for the *last* space character. Unfortunately, Excel doesn’t provide any simple way to locate the position of the first occurrence of a character from the *end* of a string. The example in this section solves that problem and describes a way to determine the position of the first occurrence of a specific character beginning from the end of a text string.



This technique involves arrays, so you might want to review the material in Part IV to familiarize yourself with this topic.

This example describes how to create a megaformula that returns the character position of the last *space character* in a string. You can, of course, modify the formula to work with any other character.

CREATING THE INTERMEDIATE FORMULAS

The general plan is to create an array of characters in the string, but in reverse order. Once that array is created, we can use the MATCH function to locate the first space character in the array.

Refer to Figure 20-4, which shows the results of the intermediate formulas. Cell A1 contains an arbitrary name, which happens to be comprised of 12 characters. The range B1:B12 contains the following array formula:

```
{=ROW(INDIRECT("1:"&LEN(A1)))}
```

	A	B	C	D	E	F	G
1	Jim E. Brown	1	12	n	6	7	
2		2	11	w			
3		3	10	o			
4		4	9	r			
5		5	8	B			
6		6	7				
7		7	6	.			
8		8	5	E			
9		9	4				
10		10	3	m			
11		11	2	i			
12		12	1	J			
13							

Figure 20-4: These intermediate formulas will eventually be converted to a single megaformula.

You enter this formula into the entire B1:B12 range by selecting the range, typing the formula, and pressing Ctrl+Shift+Enter. Don't type the curly brackets. Excel adds the brackets to indicate an array formula. This formula returns an array of 12 consecutive integers.

The range C1:C12 contains the following array formula:

```
{=LEN(A1)+1-B1:B12}
```

This formula essentially reverses the integers generated in column B.

The range D1:D12 contains the following array formula:

```
{=MID(A1,C1:C12,1)}
```

This formula uses the MID function to extract the individual characters in cell A1. The MID function uses the array in C1:C12 as its second argument. The result is an array of characters in reverse order.

The formula in cell E1 is:

```
=MATCH(" ",D1:D12,0)
```

This formula, which is *not* an array formula, uses the MATCH function to return the position of the first space character in the range D1:D12. In the example shown in Figure 20-4, the formula returns 6, which means that the first space character is six characters from the end of the text in A1.

The formula in cell F1 is:

```
=LEN(A1)+1-E1
```

This formula returns the character position of the last space in the string.

You may wonder how all of these formulas can possibly be combined into a single formula. Keep reading for the answer.

CREATING THE MEGAFORMULA

At this point, cell F1 contains the result we are looking for. The challenge is consolidating all of those intermediate formulas into a single formula. The goal is to produce a formula that contains only references to cell A1. These steps will get you to that goal:

1. The formula in cell F1 contains a reference to cell E1. Replace that reference with the text of the formula in cell E1. As a result, the formula in cell F1 becomes:

```
=LEN(A1)+1-MATCH(" ",D1:D12,0)
```

2. Now the formula contains a reference to D1:D12. This range contains a single array formula. Replacing the reference to D1:D12 with the array formula results in the following array formula in cell F1:

```
{=LEN(A1)+1-MATCH(" ",MID(A1,C1:C12,1),0)}
```



Because an array formula replaced the reference in cell F1, you now must enter the formula in F1 as an array formula (enter it with Ctrl+Shift+Enter).

3. Now the formula in cell F1 contains a reference to C1:C12, which also contains an array formula. Replace the reference to C1:C12 with the array formula in C1:C12 to get this array formula in cell F1:

```
{=LEN(A1)+1-MATCH(" ",MID(A1,LEN(A1)+1-B1:B12,1),0)}
```

4. Next, replace the reference to B1:B12 with the array formula in B1:B12. The result is:

```
{=LEN(A1)+1-MATCH(" ",MID(A1,LEN(A1)+1-ROW(INDIRECT("1:"&LEN(A1))),1),0)}
```

Now the array formula in cell F1 refers only to cell A1, which is exactly what we want. The megaformula does all of the work, and you can delete all of the intermediate formulas.

Although you use a 12-digit value and arrays stored in 12-row ranges to create the formula, the final formula does not use any of these range references. Consequently, the megaformula works with a value of any length.

PUTTING THE MEGAFORMULA TO WORK

Figure 20-5 shows a worksheet with names in column A. Column B contains the megaformula developed in the previous section. Column C contains a formula that extracts the characters beginning after the last space, which represents the last name of the name in column A.

	A	B	C	D
1	Paula M. Smith	9	Smith	
2	Michael Alan Jones	13	Jones	
3	Mike Helton	5	Helton	
4	Tom Alvin Jacobs	10	Jacobs	
5	John Jacob Robert Smith	18	Smith	
6	Mr. Hank R. Franklin	12	Franklin	
7	James Jackson Jr.	14	Jr.	
8	Jill M. Horneg	8	Horneg	
9	Rodger K. Moore	10	Moore	
10	Andy R. Maxwell	8	Maxwell	
11	Michelle Theresa Hunt	17	Hunt	
12				

Figure 20-5: Column B contains a megaformula that returns the character position of the last space of the name in column A.

Cell C1, for example, contains this formula:

```
=RIGHT(A1,LEN(A1)-B1)
```

If you like, you can eliminate the formulas in column B and create a specialized formula that returns the last name. To do so, substitute the formula in B1 for the reference to B1 in the formula. The result is the following array formula:

```
{=RIGHT(A1,LEN(A1)-(LEN(A1)+1-MATCH(" ",MID(A1,LEN(A1)+1-  
ROW(INDIRECT("1:"&LEN(A1))),1),0)))}
```



You must insert parentheses around the formula text copied from cell B1. Without the parentheses, the formula does not evaluate correctly.



The workbook for locating a string's last space character is available on the companion CD-ROM.

Using a Megaformula to Determine the Validity of a Credit Card Number

You may not know it, but you can determine the validity of a credit card number by using a relatively complex algorithm to analyze the digits of the number. In addition, you can determine the type of credit card by examining the initial digits and the length of the number. Table 20-4 shows information about four major credit cards.

TABLE 20-4 INFORMATION ABOUT FOUR CREDIT CARDS

Credit Card	Prefix Digits	Total Digits
Mastercard	51–55	16
Visa	4	13 or 16
American Express	34 or 37	15
Discover	6011	16



“Validity,” as used here, means whether the credit card number *itself* is a valid number. This technique, of course, cannot determine if the number represents an active credit card account.

You can test the validity of a credit card account number by processing its checksum digits. All account numbers used in major credit cards use a “mod 10” check digit algorithm. The general process follows these steps:

1. Add leading zeros to the account number to make the total digits equal 16.
2. Beginning with the first digit, double the value of alternate digits of the account number. If the result is a two-digit number, add the two digits together.

3. Add the eight values generated in Step 2 to the sum of the skipped digits of the original number.
4. If the sum obtained in Step 3 is evenly divisible by 10, the number is a valid credit card number.

The example in this section describes a megaformula that determines if a credit card number is a valid number.

THE BASIC FORMULAS

Figure 20-6 shows a worksheet set up to analyze a credit card number and determine its validity. This workbook uses quite a few formulas to make the determination.

	A	B	C	D	E	F	G
1						Credit Card Number: 4384842201065	INVALID
2						0004384842201065	
3							
4	Digit Number	Digit	Digit Multiplier	Equals	Sum of the digits		
5	1	0	2	0	0		
6	2	0	1	0	0		
7	3	0	2	0	0		
8	4	4	1	4	4		
9	5	3	2	6	6		
10	6	8	1	8	8		
11	7	4	2	8	8		
12	8	8	1	8	8		
13	9	4	2	8	8		
14	10	2	1	2	2		
15	11	2	2	4	4		
16	12	0	1	0	0		
17	13	1	2	2	2		
18	14	0	1	0	0		
19	15	6	2	12	3		
20	16	5	1	5	5		
21					58		
22							

Figure 20-6: The formulas in this workbook determine the validity of a credit card number.

In this workbook, the credit card number is entered in cell F1, with no spaces or hyphens. The formula in cell F2 follows. This formula appends leading zeros, if necessary, to make the card number exactly 16 digits. The other formulas use the string in cell F2.

```
=REPT("0",16-LEN(F1))&F1
```



When entering a credit card number that contains more than 15 digits, you must be careful that Excel does not round the number to 15 digits. You can precede the number with an apostrophe or preformat the cell as Text (using the Number Format tab of the Format Cells dialog box).

Column A contains a series of integers from 1 to 16, representing the digit positions of the credit card.

Column B contains formulas that extract each digit from cell F2. For example, the formula in cell B5 is:

```
=MID($F$2,A5,1)
```

Column C contains the multipliers for each digit: alternating 2s and 1s.

Column D contains formulas that multiply the digit in column B by the multiplier in column C. For example, the formula in cell D5 is:

```
=B5*C5
```

Column E contains formulas that sum the digits displayed in column D. A single digit value in column D is returned directly. For two-digit values, the sum of the digits is displayed in Column E. For example, if column D displays 12, the formula in column E returns 3 (that is, 1 + 2). The formula that accomplishes this is:

```
=INT((D5/10)+MOD((D5),10))
```

Cell E21 contains a simple SUM formula to add the values in column E:

```
=SUM(E5:E20)
```

The formula in cell G1, which follows, calculates the remainder when cell E21 is divided by 10. If the remainder is 0, the card number is valid and the formula displays *VALID*. Otherwise, the formula displays *INVALID*.

```
=IF(MOD(E21,10)=0,"VALID","INVALID")
```

CONVERT TO ARRAY FORMULAS

It's important to understand that the megaformula that we'll create will be an array formula because the intermediary formulas occupy multiple rows.

First, you need to convert all of the formulas to array formulas. Note that columns A and C consist of values, not formulas. To use the values in a megaformula, they must be generated by formulas – more specifically, array formulas.

Enter the following array formula into the range A5:A20. This array formula returns a series of 16 consecutive integers.

```
{=ROW(INDIRECT("1:16"))}
```

For column B, select B5:B20 and enter the following array formula, which extracts the digits from the credit card number:

```
{=MID($F$2,A5:A20,1)}
```

Next, column C requires an array formula that generates alternating values of 2 and 1. Such a formula, entered into the range C5:C20, is shown here:

```
{=(MOD(ROW(INDIRECT("1:16")),2)+1)}
```

For column D, select D5:D20 and enter the following array formula:

```
{=B5:B20*C5:C20}
```

Finally, select E5:E20 and enter this array formula:

```
{=INT((D5:D20/10)+MOD((D5:D20),10))}
```

Now there are five columns of 16 rows, but only five actual formulas. These are multicell array formulas.

BUILD THE MEGAFORMULA

To create the megaformula for this task, start with cell G1, which is the cell that has the final result. The original formula in G1 is:

```
=IF(MOD(E21,10)=0,"VALID","INVALID")
```

First, replace the reference to cell E21 with the formula in E21. Doing so results in the following formula in cell G1:

```
=IF(MOD(SUM(E5:E20),10)=0,"VALID","INVALID")
```

Next, replace the reference to E5:E20 with the array formula contained in that range. Now the formula becomes an array formula so you must enter it with Ctrl+Shift+Enter. After the replacement, the formula in G1 is:

```
{=IF(MOD(SUM(INT((D5:D20/10)+MOD((D5:D20),10))),10)=0,"VALID","INVALID")}
```

Replace the two references to range D5:D20 with the array formula contained in D5:20. Doing so results in the following array formula in cell G1:

```
{=IF(MOD(SUM(INT((B5:B20*C5:C20/10)+MOD((B5:B20*C5:C20),10))),10)=0,"VALID","INVALID")}
```


Next, replace the references to cell C5:C20 with the array formula in C5:C20. Note that you must have a set of parentheses around the copied formula text. The result is as follows:

```
{=IF(MOD(SUM(INT((B5:B20*(MOD(ROW(INDIRECT("1:16")),2)+1)/10)+
MOD((B5:B20*(MOD(ROW(INDIRECT("1:16")),2)+1)),10))),10)=0,
"VALID","INVALID")}
```

Replacing the references to B5:B20 with the array formula contained in B5:B20 yields the following:

```
{=IF(MOD(SUM(INT((MID($F$2,A5:A20,1)*(MOD(ROW(INDIRECT("1:16")),2)
+1)/10)+MOD((MID($F$2,A5:A20,1)*(MOD(ROW(INDIRECT("1:16")),
2)+1)),10))),10)=0,"VALID","INVALID")}
```

Substitute the array formula in range A5:A20 for the references to that range. The resulting array formula is:

```
{=IF(MOD(SUM(INT((MID($F$2,ROW(INDIRECT("1:16")),1)*(MOD(ROW
(INDIRECT("1:16")),2)+1)/10)+MOD((MID($F$2,ROW(INDIRECT("1:16")),1)*
(MOD(ROW(INDIRECT("1:16")),2)+1)),10))),10)=0,"VALID","INVALID")}
```

Finally, substitute the formula in cell F2 for the two references to cell F2. After making the substitutions, the formula is as follows:

```
{=IF(MOD(SUM(INT((MID(REPT("0",16-LEN(F1))&F1,
ROW(INDIRECT("1:16")),1)*(MOD(ROW(INDIRECT("1:16")),2)+1)/
10)+MOD((MID(REPT("0",16-LEN(F1))&F1,ROW(INDIRECT("1:16")),1)*
(MOD(ROW(INDIRECT("1:16")),2)+1)),10))),10)=0,"VALID",
"INVALID")}
```

You can delete the now superfluous intermediate formulas. The final megaformula, a mere 229 characters in length, does the work of 51 intermediary formulas!



You can access the credit card number validation workbook on the companion CD-ROM.

The Pros and Cons of Megaformulas

If you followed the examples in this chapter, you probably realize that the main advantage of creating a megaformula is to eliminate intermediate formulas. Doing so can streamline your worksheet, reduce the size of your workbook files, and even result in faster recalculations.

The downside? Creating a megaformula does, of course, require some additional time and effort. And, as you've undoubtedly noticed, a megaformula is virtually impossible for anyone (even the author) to figure out. If you decide to use megaformulas, take extra care to ensure that the intermediate formulas are performing correctly before you start building a megaformula. Even better, keep a single copy of the intermediate formulas somewhere in case you discover an error or need to make a change.

Summary

This chapter described a useful technique that involves combining multiple formulas into a single, complex formula (a megaformula). I presented several examples of creating such formulas.

The next chapter takes a look at formulas you can create for debugging purposes.

Chapter 21

Tools and Methods for Debugging Formulas

IN THIS CHAPTER

- ◆ What is formula debugging?
- ◆ How to identify and correct common formula errors
- ◆ A description of Excel's auditing tools
- ◆ Auditing tools available from third-party providers

ERRORS HAPPEN. AND WHEN YOU CREATE Excel formulas, errors happen very frequently. This chapter describes common formula errors, and discusses tools and methods that you can use to help create formulas that work as they are intended to work.

Formula Debugging?

The term *debugging* refers to the process of identifying and correcting errors in a computer program. Strictly speaking, an Excel formula is not a computer program. Formulas, however, are subject to the same types of problems that occur in a computer program. If you create a formula that does not work as it should, then you need to identify and correct the problem.

The ultimate goal in developing a spreadsheet solution is to generate accurate results. For simple worksheets, this is not difficult, and you can usually tell whether the results are correct. But as your worksheets grow in size or complexity, ensuring accuracy becomes more difficult.

Making a change in a worksheet – even a relatively minor change – may produce a ripple effect that introduces errors in other cells. For example, accidentally entering a value into a cell that previously held a formula is all too easy to do. This simple error can have a major impact on other formulas, and you may not discover the problem until long after you make the change. Or, you may *never* discover the problem.

Research on Spreadsheet Errors

Using a spreadsheet can be hazardous to your company's bottom line. It's far too easy to simply assume that your spreadsheet produces accurate results. If you use the results of a spreadsheet to make a major decision, it's especially important to make sure that the formulas return accurate and meaningful results.

Researchers have conducted quite a few studies that deal with spreadsheet errors. Generally, these studies have found that between 20 and 40 percent of all spreadsheets contain some type of error. If this type of research interests you, I urge you to check out the Spreadsheet Research (SSR) Web site maintained by Ray Panko of the University of Hawaii. The URL is:

<http://panko.cba.hawaii.edu/ssr/>

Formula Problems and Solutions

Formula errors tend to fall into one of the following six general categories:

- ◆ **Syntax errors:** You have a problem with the syntax of a formula. For example, a formula may have mismatched parentheses, or you may have spelled a function name incorrectly.
- ◆ **Logical errors:** A formula does not return an error, but it contains a logical flaw that causes it to return an incorrect result.
- ◆ **Incorrect reference errors:** The logic of the formula is correct, but the formula uses an incorrect cell reference. As a simple example, the range reference in a SUM formula may not include all of the data that you want to sum.
- ◆ **Circular references:** A circular reference occurs when a formula refers to its own cell, either directly or indirectly. Circular references are useful in a few cases, but most of the time a circular reference indicates a problem.
- ◆ **Array formula entry error:** When entering (or editing) an array formula, you must use Ctrl+Shift+Enter to enter the formula. If you fail to do so, Excel does not recognize the formula as an array formula.
- ◆ **Incomplete calculation errors:** The formulas simply aren't calculated fully. Microsoft has acknowledged some problems with Excel's calculation engine in some versions of Excel. To ensure that your formulas are fully calculated, use Ctrl+Alt+F9.

Syntax errors are usually the easiest to identify and correct. In most cases, you will know when your formula contains a syntax error. For example, Excel won't permit you to enter a formula with mismatched parentheses. Other syntax errors also usually result in an error display in the cell.

The remainder of this section describes some common formula problems and offers advice on identifying and correcting them.

Mismatched Parentheses

In a formula, every left parenthesis must have a corresponding right parenthesis. If your formula has mismatched parentheses, Excel usually won't permit you to enter it. An exception to this rule involves a simple formula that uses a function. For example, if you enter the following formula (which is missing a closing parenthesis), Excel accepts the formula and provides the missing parenthesis.

```
=SUM(A1:A500
```

A formula may have an equal number of left and right parentheses, but the parentheses may not match properly. For example, consider the following formula, which converts a text string such that the first character is uppercase and the remaining characters are lowercase. This formula has five pairs of parentheses, and they match properly.

```
=UPPER(LEFT(A1))&RIGHT(LOWER(A1),LEN(A1)-1)
```

The following formula also has five pairs of parentheses, but they are mismatched. The result displays a syntactically correct formula that simply returns the wrong result.

```
=UPPER(LEFT(A1)&RIGHT(LOWER(A1),LEN(A1)-1))
```

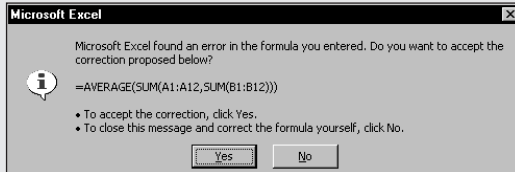
Often, parentheses that are in the wrong location will result in a syntax error—which is usually a message that tells you that you entered too many or too few arguments for a function.



Excel can help you out with mismatched parentheses. When you edit a formula, move the cursor to a parenthesis and pause. Excel displays it (and its matching parenthesis) in bold for about one second.

Using Formula AutoCorrect

When you enter a formula that has a syntax error, Excel attempts to determine the problem and offers a suggested correction. The accompanying figure shows an example of a proposed correction.



Exercise caution when accepting corrections for your formulas from Excel, because it does not always guess correctly. For example, I entered the following formula (which has mismatched parentheses):

```
=AVERAGE(SUM(A1:A12 ,SUM(B1:B12) )
```

Excel then proposed the following correction to the formula:

```
=AVERAGE(SUM(A1:A12 ,SUM(B1:B12) ) )
```

You may be tempted to accept the suggestion without even thinking. In this case, the proposed formula is syntactically correct – but not what I intended. The correct formula is:

```
=AVERAGE(SUM(A1:A12) ,SUM(B1:B12) )
```

Cells Are Filled with

A cell is filled with a series of hash marks (#) for one of two reasons:

- ◆ The column is not wide enough to accommodate the formatted numeric value. To correct it, you can make the column wider or use a different number format.
- ◆ The cell contains a formula that returns an invalid date or time. For example, Excel does not support dates prior to 1900 or the use of negative time values. Attempting to display either of these will result in a cell filled with hash marks. Widening the column won't fix it.

Blank Cells Are Not Blank

Some Excel users have discovered that by pressing the spacebar, the contents of a cell seem to erase. Actually, pressing the spacebar inserts an invisible space character, which is not the same as erasing the cell.

For example, the following formula returns the number of non-empty cells in range A1:A10. If you “erase” any of these cells by using the spacebar, these cells are included in the count, and the formula returns an incorrect result.

```
=COUNTA(A1:A10)
```

If your formula does not ignore blank cells the way that it should, check to make sure that the blank cells are really blank cells.

Formulas Returning an Error

A formula may return any of the following error values:

- ◆ #DIV/0!
- ◆ #NA
- ◆ #NAME?
- ◆ #NULL!
- ◆ #NUM!
- ◆ #REF!
- ◆ #VALUE!

The following sections summarize possible problems that may cause these errors.



A new feature in Excel 2002 lets you choose how error values are printed. To access this feature, select File → Page Setup and click the Sheet tab. You can choose to print error values as blank cells, dashes, or #N/A.

Tracing Error Values

The Trace Error button on the Auditing toolbar helps you to identify the cell that is causing an error value to appear. Often, an error in one cell is the result of an error in a precedent cell. Activate a cell containing an error, and then click the Trace Error button. Excel draws arrows to indicate the error source.

#DIV/0! ERRORS

Division by zero is not permitted. If you attempt to do so, Excel displays its familiar #DIV/0! error value.

Because Excel considers a blank cell to be zero, you also get this error if your formula divides by a missing value. This is a common problem when you create formulas for data that you haven't entered yet, as shown in Figure 21-1. The formula in cell D2, which was copied to the cells below it, is:

```
=(C2-B2)/C2
```

1	A	B	C	D	E	F
2	Month	Last Year	This Year	Change		
3	January	175	188	6.9%		
4	February	156	166	6.0%		
5	March	198	175	-13.1%		
6	April	144	187	23.0%		
7	May	132	149	11.4%		
8	June	198		#DIV/0!		
9	July	202		#DIV/0!		
10	August	184		#DIV/0!		
11	September	140		#DIV/0!		
12	October	198		#DIV/0!		
13	November	232		#DIV/0!		
14	December	255		#DIV/0!		

Figure 21-1: #DIV/0! errors occur when the data in column C is missing.

This formula calculates the percent change between the values in columns B and C. Data is not available for months beyond May, so the formula returns a #DIV/0! error.

To avoid the error display, you can use an IF function to check for a blank cell in column C:

```
= IF(C2=0, "", (C2-B2)/C2)
```

This formula displays an empty string if cell C2 is blank or contains 0; otherwise, it displays the calculated value.

Another approach is to use an IF function to check for *any* error condition. The following formula, for example, displays an empty string if the formula results in any type of error.

```
=IF(ISERROR((C2-B2)/C2), "", (C2-B2)/C2)
```

#N/A ERRORS

The #N/A error occurs if any cell referenced by a formula displays #N/A.



Some users like to enter =NA() or #N/A explicitly for missing data. This method makes it perfectly clear that the data is not available and hasn't been deleted accidentally. When you create a line chart from cells that contain #N/A, the missing data is interpolated. If you plot an empty cell, on the other hand, the line chart will show a gap for the missing data.

The #N/A error also occurs when a lookup function (HLOOKUP, LOOKUP, MATCH, or VLOOKUP) can't find a match.

#NAME? ERRORS

The #NAME? error occurs under these conditions:

- ◆ The formula contains an undefined range or cell name.
- ◆ The formula contains text that Excel *interprets* as an undefined name. A misspelled function name, for example, generates a #NAME? error.
- ◆ The formula uses a worksheet function that's defined in an add-in, and the add-in is not installed.



Excel has a bit of a problem with range names. If you delete a name for a cell or range and the name is used in a formula, the formula continues to use the name, even though it's no longer defined. As a result, the formula displays #NAME?. You may expect Excel to automatically convert the names to their corresponding cell references, but this does not happen.

#NULL! ERRORS

The #NULL! error occurs when a formula attempts to use an intersection of two ranges that don't actually intersect. Excel's intersection operator is a space. The following formula, for example, returns #NULL! because the two ranges don't intersect.

```
=SUM(B5:B14 A16:F16)
```

The following formula does not return #NULL!, but displays the contents of cell B9 – which represents the intersection of the two ranges.

```
=SUM(B5:B14 A9:F9)
```

#NUM! ERRORS

A formula returns a #NUM! error if any of the following occurs:

- ◆ You pass a non-numerical argument to a function when a numerical argument is expected.
- ◆ A function that uses iteration can't calculate a result. Examples of functions that use iteration are IRR and RATE.
- ◆ A formula returns a value that is too large or too small. Excel supports values between 1E-307 and 1E+307.

#REF! ERRORS

The #REF! error occurs when a formula uses an invalid cell reference. This error can occur in the following situations:

- ◆ You delete a cell that is referenced by the formula. For example, the following formula displays a #REF! error if row 1, column A, or column B is deleted.

```
=A1/B1
```

- ◆ You copy a formula to a location that invalidates the relative cell references. For example, if you copy the following formula from cell A2 to cell A1, the formula returns #REF! because it attempts to refer to a nonexistent cell.

```
=A1-1
```

- ◆ You delete a cell that is referenced by the formula. For example, the following formula will display a #REF! error if row 1, column A, or column B is deleted.

```
=A1/B1
```

- ◆ You cut a cell (using Edit → Cut) and then paste it to a cell that's referenced by a formula. The formula will display #REF!.

Pay Attention to the Colors

When you edit a cell that contains a formula, Excel color-codes the cell and range references in the formula. Excel also outlines the cells and ranges used in the formula by using corresponding colors. Therefore, you can see at a glance the cells that are used in the formula.

You also can manipulate the colored outline to change the cell or range reference. To change the references that are used, drag the outline's border or fill handle (at the lower-right corner of the outline).

#VALUE! ERRORS

The #VALUE! error is very common and can occur under the following conditions:

- ◆ An argument for a function is of an incorrect data type or the formula attempts to perform an operation using incorrect data. For example, a formula that adds a value to a text string returns the #VALUE! error.
- ◆ A function's argument is a range when it should be a single value.
- ◆ A custom worksheet function is not calculated. With some versions of Excel, inserting or moving a sheet may cause this error. You can use Ctrl+Alt+F9 to force a recalculation.
- ◆ A custom worksheet function attempts to perform an operation that is not valid. For example, custom functions cannot modify the Excel environment or make changes to other cells.
- ◆ You forget to press Ctrl+Shift+Enter when entering an array formula.

Absolute/Relative Reference Problems

As described in Chapter 2, a cell reference can be relative (for example, A1), absolute (for example, \$A\$1), or mixed (for example, \$A1 or A\$1). The type of cell reference that you use in a formula is relevant only if the formula will be copied to other cells.

A common problem is to use a relative reference when you should use an absolute reference. As shown in Figure 21-2, cell C1 contains a tax rate, which is used in the formulas in column C. The formula in cell C4 is:

```
=B4+(B4*$C$1)
```

	A	B	C	D
1		Tax Rate:	7.25%	
2				
3	Item	Price	Price + Tax	
4	C-092	\$149.95	\$160.82	
5	R-112	\$79.99	\$85.79	
6	G-972	\$39.95	\$42.85	
7				
8				
9				

Figure 21-2: Formulas in the range C4:C6 use an absolute reference to cell C1.

Notice that the reference to cell C1 is an absolute reference. When the formula is copied to other cells in column C, the formula continues to refer to cell C1. If the reference to cell C1 were a relative reference, the copied formulas would return an incorrect result.

Operator Precedence Problems

Excel has some straightforward rules about the order in which mathematical operations are performed. In Table 21-1, operations with a lower precedence are performed before operations with a higher precedence. This table, for example, shows that multiplication has a higher precedence than addition. Therefore, multiplication is performed first.

TABLE 21-1 OPERATOR PRECEDENCE IN EXCEL FORMULAS

Symbol	Operator	Precedence
-	Negation	1
%	Percent	2
^	Exponentiation	3
* and /	Multiplication and division	4
+ and -	Addition and subtraction	5
&t	Text concatenation	6
=, <, >, and <>	Comparison	7

When in doubt (or when you simply need to clarify your intentions), you should use parentheses to ensure that operations are performed in the correct order. For example, the following formula multiplies A1 by A2, and then adds 1 to the result. The multiplication is performed first, because it has a higher order of precedence.

```
= 1+A1*A2
```

The following is a clearer version of this formula. The parentheses aren't necessary, but, in this case, the order of operations is perfectly obvious.

```
=1+(A1*A2)
```

Notice that the negation operator symbol is exactly the same as the subtraction operator symbol. This, as you may expect, can cause some confusion. Consider these two formulas:

```
=-3^2  
=0-3^2
```

The first formula, as expected, returns 9. The second formula, however, returns -9. Squaring a number always produces a positive result, so how is it that Excel can return the -9 result?

In the first formula, the minus sign is a negation operator and has the highest precedence. However, in the second formula, the minus sign is a subtraction operator, which has a lower precedence than the exponentiation operator. Therefore, the value 3 is squared and the result is subtracted from zero, which produces a negative result.



Excel is a bit unusual in interpreting the negation operator. Other spreadsheet products (for example, 1-2-3 and Quattro Pro) return -9 for both formulas. In addition, Excel's VBA language also returns -9 for these expressions.

Using parentheses, as shown in the following formula, causes Excel to interpret the operator as a minus sign rather than a negation operator. This formula returns 9.

```
=0+(-3^2)
```

Formulas Are Not Calculated

If you use custom worksheet functions written in VBA, you may find that formulas that use these functions fail to get recalculated and may display incorrect results. To force a recalculation of all formulas, press Ctrl+Alt+F9.



Prior to Excel 2000, this key combination was not documented.

Actual versus Displayed Values

You may encounter a situation in which values in a range don't appear to add up properly. For example, Figure 21-3 shows a worksheet with the following formula entered into each cell in the range B2:B4:

=1/3

	A	B	C	D	E
1					
2		0.333			
3		0.333			
4		0.333			
5		1.000			
6					
7					
8					
9					

Figure 21-3: A simple demonstration of numbers that appear to add up incorrectly

Cell B5 contains the following formula:

=SUM(B2:B4)

All of the cells are formatted to display with three decimal places. As you can see, the formula in cell B5 appears to display an incorrect result (you may expect it to display 0.999). The formula, of course, *does* return the correct result. The formula uses the *actual* values in the range B2:B4, not the displayed values.

You can instruct Excel to use the displayed values by checking the Precision as displayed checkbox on the Calculation tab of the Options dialog box (choose Tools → Options to display this dialog box).



Checking the Precision as displayed checkbox also affects normal values (non-formulas) that have been entered into cells. For example, if a cell contains the value 4.68 and is displayed with no decimal places (that is, 5), then checking the Precision as displayed checkbox converts 4.68 to 5.00. This change is permanent and you can't restore the original value if you later uncheck the Precision as displayed checkbox.

Floating Point Number Errors

Computers, by their very nature, don't have infinite precision. Excel stores numbers in binary format by using eight bytes, which can handle numbers with 15-digit accuracy. Some numbers can't be expressed precisely by using eight bytes, so the number stores as an approximation.

To demonstrate how this may cause problems, enter the following formula into cell A1:

```
=(5.1-5.2)+1
```

The result should be 0.9. However, if you format the cell to display 15 decimal places, you'll discover that Excel calculates the formula with a result of 0.899999999999999. This occurs because the operation in parentheses is performed first, and this intermediate result stores in binary format by using an approximation. The formula then adds 1 to this value, and the approximation error is propagated to the final result.

In many cases, this type of error does not present a problem. However, if you need to test the result of that formula by using a logical operator, it *may* present a problem. For example, the following formula (which assumes that the previous formula is in cell A1) returns FALSE:

```
=A1=.9
```

One solution to this type of error is to use Excel's ROUND function. The following formula, for example, returns TRUE because the comparison is made by using the value in A1 rounded to one decimal place.

```
=ROUND(A1,1)=0.9
```

Here's another example of a "precision" problem. Try entering the following formula:

```
=1.333+1.225-1.333-1.225
```

If you use Excel 97 or a later version, the formula returns 0. Previous versions return 2.22044604925031E-16 (a number very close to zero).



Beginning with Excel 97, if an addition or subtraction operation results in a value at or very close to zero, the calculation engine compensates for any error introduced as a result of converting an operand to and from binary. When you perform the previous example in Excel 97 (and later versions), it correctly displays 0.

“Phantom Link” Errors

You may open a workbook, and see a message like the one shown in Figure 21-4. This message sometimes appears even when a workbook contains no linked formulas.

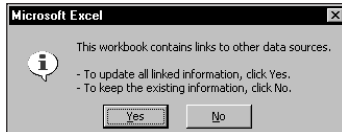


Figure 21-4: Excel's way of asking you if you want to update links in a workbook

In the vast majority of cases, this phantom link problem occurs because of an erroneous name. Select **Insert** → **Name** → **Define** and scroll through the list of names. If you see a name that refers to `#REF!`, delete the name.



These phantom links may be created when you copy a worksheet that contains names. See Chapter 3 for more information about names.

Circular Reference Errors

A *circular reference* is a formula that contains a reference to the cell that contains the formula. The reference may be direct or indirect. For help tracking down a circular reference, see “Excel’s Auditing Tools,” later in this chapter.



As described in Chapter 14, you may encounter some situations in which you create an intentional circular reference.

Excel’s Auditing Tools

Excel includes a number of tools that can help you track down formula errors. This section describes the auditing tools built into Excel.

Identifying Cells of a Particular Type

The Go To Special dialog box enables you to specify the type of cells that you want Excel to select. To display this dialog box, choose Edit → Go To (or press F5 or Ctrl+G). The Go To dialog box appears. Click the Special button, which displays the Go To Special dialog box, as shown in Figure 21-5.

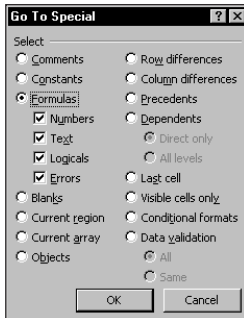


Figure 21-5: The Go To Special dialog box



If you select a multicell range before choosing Edit → Go To, the command operates only within the selected cells. If a single cell is selected, the command operates on the entire worksheet.

You can use the Go To Special dialog box to select cells of a certain type, which can often help in identifying errors. For example, if you choose the Formulas option, Excel selects all of the cells that contain a formula. If you zoom the worksheet out to a small size, you can get a good idea of the worksheet's organization (see Figure 21-6).

Selecting the formula cells may also help you to spot a common error—a formula that has been replaced accidentally with a value. If you find a cell that's not selected amid a group of selected formula cells, chances are good that the cell previously contained a formula that has been replaced by a value.

Viewing Formulas

You can become familiar with an unfamiliar workbook by displaying the formulas rather than the results of the formulas. Select Tools → Options, and check the checkbox labeled “Formulas” on the View tab. You may want to create a new window for the workbook before issuing this command. This way, you can see the formulas in one window and the results of the formula in the other window. Use the Window → New Window command to open a new window.

Figure 21-6: Zooming out and selecting all formula cells can give you a good overview of how the worksheet is designed.



You can use **Ctrl+`** to toggle between Formula view and Normal view.



In Excel 2002, you can also use the **Tools → Formula Auditing → Formula Auditing Mode** command to toggle formula view on and off. When formula view is in effect, the Formula Auditing Toolbar is also displayed.

Figure 21-7 shows an example of a worksheet displayed in two windows. The window on the top shows Normal view (formula results), and the window on the bottom displays the formulas.

When Formula view is in effect, Excel highlights the cells that are used by the formula in the active cell. In Figure 21-7, for example, the active cell is C11. The cells used by this formula are highlighted in both windows.

	A	B	C	D	E	F	G	H	I	J
1	Commission Rate	5.50%	Normal commission rate							
2	Sales Goal	15%	Improvement from prior month							
3	Bonus Rate	6.50%	Paid if Sales Goal is attained							
4										
5	Sales Rep	Last Month	This Month	Change	Pct. Change	Met Goal?	Commission			
6	Murray	101,233	108,444	7,211	7.1%	TRUE	7,048.86			
7	Knuckles	120,933	108,434	(12,499)	-10.3%	FALSE	5,963.87			
8	Lefty	139,832	165,901	26,069	18.6%	TRUE	10,783.57			
9	Lucky	98,323	100,083	1,760	1.8%	FALSE	5,504.57			
10	Scarface	78,322	79,923	1,601	2.0%	FALSE	4,395.77			
11	Total	538,643	562,785	24,142	4.5%		33,696.63			

	A	B	C	D	E
1	Commission Rate	0.055	Normal commission rate		
2	Sales Goal	0.15	Improvement from prior month		
3	Bonus Rate	0.065	Paid if Sales Goal is attained		
4					
5	Sales Rep	Last Month	This Month	Change	Pct. Change
6	Murray	101233	108444	=C6-B6	=D6/B6
7	Knuckles	120933	108434	=C7-B7	=D7/B7
8	Lefty	139832	165901	=C8-B8	=D8/B8
9	Lucky	98323	100083	=C9-B9	=D9/B9
10	Scarface	78322	79923	=C10-B10	=D10/B10
11	Total	=SUM(B6:B10)	=SUM(C6:C10)	=SUM(D6:D10)	=D11/B11

Figure 21-7: Displaying formulas (bottom window) and their results (top window)



If you need a printed list of your formulas, you'll soon find that Excel's Formula view is not really designed for printing. My Power Utility Pak add-in (on the companion CD-ROM) includes a utility that will generate a handy, printable list of all formulas in a worksheet or workbook.

Tracing Cell Relationships

To understand how to trace cell relationships, you need to familiarize yourself with the following two concepts:

- ◆ **Cell precedents:** Applicable only to cells that contain a formula, a formula cell's precedents are all the cells that contribute to the formula's result. A *direct precedent* is a cell that you use directly in the formula. An *indirect precedent* is a cell that is not used directly in the formula, but is used by a cell that you refer to in the formula.
- ◆ **Cell dependents:** These are formula cells that depend on a particular cell. A cell's dependents consist of all formula cells that use the cell. Again, the formula cell can be a direct dependent or an indirect dependent.

Identifying cell precedents for a formula cell often sheds light on why the formula is not working correctly. Conversely, knowing which formula cells depend on a particular cell is also helpful. For example, if you're about to delete a formula, you may want to check whether it has any dependents.

IDENTIFYING PRECEDENTS

You can identify cells used by a formula in the active cell in a number of ways.

- ◆ Press F2. The cells that are used directly by the formula are outlined in color, and the color corresponds to the cell reference in the formula. This technique is limited to identifying cells on the same sheet as the formula.
- ◆ Select Edit → Go To (or press F5) to display the Go To dialog box. Then click the Special button to display the Go To Special dialog box. Select the Precedents option, and then select either Direct only (for direct precedents only) or All levels (for direct and indirect precedents). Click OK and Excel highlights the precedent cells for the formula. This technique is limited to identifying cells on the same sheet as the formula.
- ◆ Press Ctrl+[to select all direct precedent cells on the active sheet.
- ◆ Press Ctrl+Shift+[to select all precedent cells (direct and indirect) on the active sheet.
- ◆ Display the Formula Auditing toolbar by selecting Tools → Formula Auditing → Show Formula Auditing Toolbar. Click the Trace Precedents button to draw arrows to indicate a cell's precedents. Click this button multiple times to see additional levels of precedents. Figure 21-8 shows a worksheet with precedent arrows drawn to indicate the precedents for the formula in cell C13.

IDENTIFYING DEPENDENTS

You can identify formula cells that use a particular cell in a number of ways.

- ◆ Select Edit → Go To (or press F5) to display the Go To dialog box. Then click the Special button to display the Go To Special dialog box. Select the Dependents option, and then select either Direct only (for direct dependents only) or All levels (for direct and indirect dependents). Click OK; Excel highlights the cells that depend on the active cell. This technique is limited to identifying cells on the active sheet only.
- ◆ Press Ctrl+] to select all direct dependent cells on the active sheet.
- ◆ Press Ctrl+Shift+] to select all dependent cells (direct and indirect) on the active sheet.

- ◆ Display the Formula Auditing toolbar by selecting Tools → Formula Auditing → Show Formula Auditing Toolbar. Click the Trace Dependents button to draw arrows to indicate a cell's dependents. Click this button multiple times to see additional levels of dependents.

Sales Rep	Last Month	This Month	Change	Pct. Change	Met Goal?	Commission
Murray	101,233	106,444	7,211	7.1%	TRUE	7,048.86
Knuckles	120,933	106,434	(12,499)	-10.3%	FALSE	5,963.87
Lefty	139,832	165,901	26,069	18.6%	FALSE	10,783.57
Lucky	98,323	100,063	1,760	1.8%	FALSE	5,504.57
Scarface	78,322	79,923	1,601	2.0%	FALSE	4,395.77
Total	538,643	562,785	24,142	4.5%		33,696.63
Average Commission Rate:		46.99%				

Figure 21-8: This worksheet displays lines that indicate cell precedents for the formula in cell C13.

Tracing Error Values

The Trace Error button on the Formula Auditing toolbar helps you to identify the cell that is causing an error value to appear. An error in one cell is often the result of an error in a precedent cell. Activate a cell that contains an error and click the Trace Error button. Excel draws arrows to indicate the error source.

Fixing Circular Reference Errors

If you accidentally create a circular reference formula, Excel displays a warning message. If you click OK, Excel displays the Circular Reference toolbar (see Figure 21-9). If you can't figure out the source of the problem, use the Navigate Circular Reference tools (a drop-down list control) on the toolbar to select a cell involved in the circular reference. Start by selecting the first cell listed, and then work your way down the list until you figure out the problem.

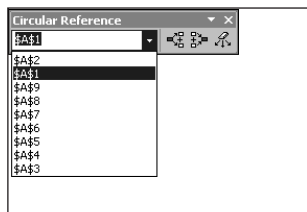


Figure 21-9: The Circular Reference toolbar

Using Excel 2002's Background Error Checking Feature

If you use Excel 2002, you may find it helpful to take advantage of the new automatic error checking feature.



The information in this section applies only to Excel 2002.

The Options dialog box in Excel 2002 contains several new tabs, including the Error Checking tab (see Figure 21-10). Error checking is turned on or off by using the checkbox labeled “Enable background error checking.” In addition, you can specify which types of errors to check for by using the checkboxes in the Rules section.

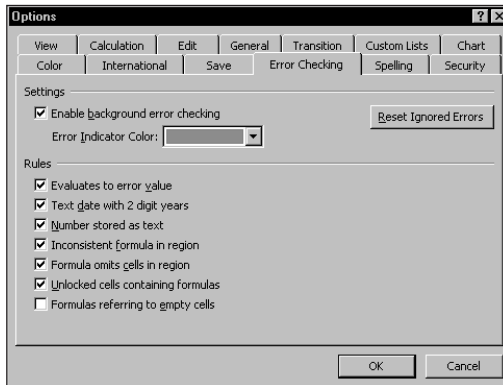


Figure 21-10: Excel 2002 can check your formulas for potential errors.

When error checking is turned on, Excel continually evaluates your worksheet, including its formulas. If a potential error is identified, Excel places a small triangle in the upper left corner of the cell. When the cell is activated, a Smart Tag appears. Clicking this Smart Tag provides you with some options. Figure 21-11 shows the options that appear when you click the Smart Tag in a cell that contains a #DIV/0 error. The options vary, depending on the type of error.

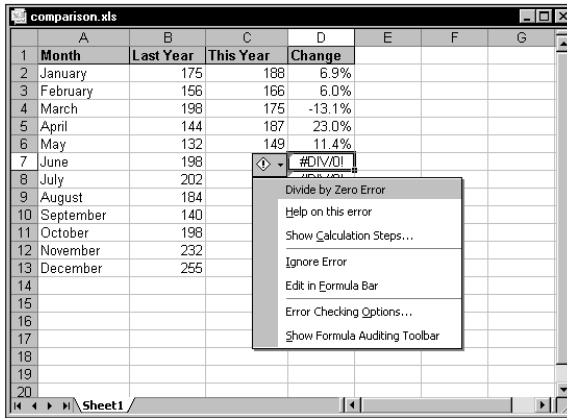


Figure 21-11: Clicking an error Smart Tag gives you a list of options.

In many cases, you will choose to ignore an error by selecting the Ignore Error option. Selecting this option eliminates the cell from subsequent error checks. However, all previously ignored errors can be reset so that they appear again (use the Reset Ignored Errors button in the Error Checking tab of the Options dialog box).

You can use the Tools → Error Checking command to display a dialog box that displays each potential error cell in sequence, much like using a spell checking program. Figure 21-12 shows the Error Checking dialog box. Note that this is a “modalless” dialog box, so you can still access your worksheet when the Error Checking dialog box is displayed.

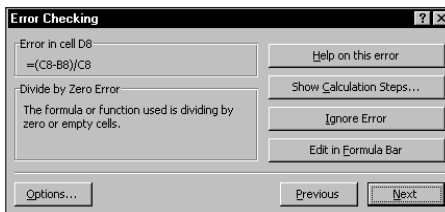


Figure 21-12: Using the Error Checking dialog box to cycle through potential errors identified by Excel



It's important to understand that the error checking feature is not perfect. In fact, it's not even *close* to perfect. In other words, you can't assume that you have an error-free worksheet simply because Excel does not identify any potential errors! Also, be aware that this error checking feature won't catch a very common type of error — that of overwriting a formula cell with a value.

Using Excel 2002's Formula Evaluator

Excel 2002 includes a new feature called the “Formula Evaluator,” which lets you see the various parts of a nested formula evaluated in the order that the formula is calculated.



The information in this section applies only to Excel 2002.

To use the Formula Evaluator, select the cell that contains the formula and choose Tools → Formula Auditing → Evaluate Formula. Or, click the Evaluate Formula button on the Formula Auditing toolbar. Either of these actions displays the Evaluate Formula dialog box, as shown in Figure 21-13.

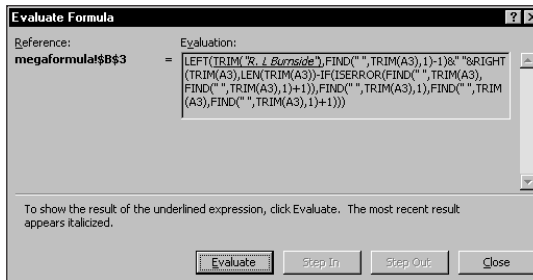


Figure 21-13: Excel 2002's Formula Evaluator shows a formula being calculated one step at a time.

Click the Evaluate button to show the result of calculating the expressions within the formula. Each click of the button performs another calculation. This feature may be useful in some situations, but overall it leaves much to be desired.

Third-Party Auditing Tools

A few third-party auditing tools for Excel are available – namely the Power Utility Pak, the Spreadsheet Detective, and the Excel Auditor. I describe them in the following sections.

Power Utility Pak

My Power Utility Pak includes a number of useful utilities relevant to auditing a worksheet. These utilities are:

- ◆ **Workbook Summary Report:** Produces a handy summary report of the entire workbook.
- ◆ **Workbook Link Report:** Produces a report that describes all links in the workbook.
- ◆ **Worksheet Map:** Creates a handy map that makes it easy to identify cells of various types (see Figure 21-14).

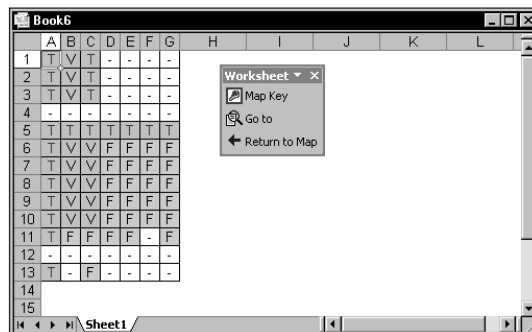


Figure 21-14: The Power Utility Pak produced this Worksheet Map.

- ◆ **Compare Sheets or Ranges:** Performs a cell-by-cell comparison of two worksheets or ranges.
- ◆ **Date Report:** Creates a useful report that summarizes all date-related cells.
- ◆ **Name Lister:** Lists all names in a workbook. Unlike Excel's Define Name dialog box, this utility also lists hidden names.
- ◆ **Formula Report:** Generates a useful (and printable) listing of all formulas in a worksheet (see Figure 21-15). This is much more useful than Excel's formula view.
- ◆ **VBA Project Summary Report:** Generates a report that describes the VBA procedures contained in a workbook.

Book9									
Formula Report for: C:\Excel Files\commission calc.xls									
Worksheet name: Sheet1									
Number of formulas: 26									
Report generated: 5/8/2001 5:26:24 PM									
Address	Row	Column	Display	Number Format	Formula	RC Formula	All Precedents	Direct Precedents	
D6	6	4	7,211	###	=C6-B6	=RC[-1]-RC[-2]	2	2	
E6	6	5	7.1%	0.0%	=D6/B6	=RC[-1]/RC[-3]	3	2	
F6	6	6	TRUE	General	=E6>=\$B\$3	=RC[-1]>=R3C2	5	2	
D7	7	4	-12,499	###	=C7-B7	=RC[-1]-RC[-2]	2	2	
E7	7	5	-10.3%	0.0%	=D7/B7	=RC[-1]/RC[-3]	3	2	
F7	7	6	FALSE	General	=E7>=\$B\$3	=RC[-1]>=R3C2	5	2	
D8	8	4	26,069	###	=C8-B8	=RC[-1]-RC[-2]	2	2	
E8	8	5	18.6%	0.0%	=D8/B8	=RC[-1]/RC[-3]	3	2	
F8	8	6	TRUE	General	=E8>=\$B\$3	=RC[-1]>=R3C2	5	2	
D9	9	4	1,760	###	=C9-B9	=RC[-1]-RC[-2]	2	2	
E9	9	5	1.8%	0.0%	=D9/B9	=RC[-1]/RC[-3]	3	2	
F9	9	6	FALSE	General	=E9>=\$B\$3	=RC[-1]>=R3C2	5	2	
D10	10	4	1,601	###	=C10-B10	=RC[-1]-RC[-2]	2	2	
E10	10	5	2.0%	0.0%	=D10/B10	=RC[-1]/RC[-3]	3	2	
F10	10	6	FALSE	General	=E10>=\$B\$3	=RC[-1]>=R3C2	5	2	
B11	11	2	538,643	###	=SUM(B6:B10)	=SUM(R[-5]C:R[-1]C)	5	5	
C11	11	3	562,785	###	=SUM(C6:C10)	=SUM(R[-5]C:R[-1]C)	5	5	
D11	11	4	24,142	###	=SUM(D6:D10)	=SUM(R[-5]C:R[-1]C)	15	5	
E11	11	5	4.5%	0.0%	=D11/B11	=RC[-1]/RC[-3]	17	2	
G6	6	7	7,049	###	=IF(F6,\$B\$3,\$B\$1)*C6	=IF(RC[-1],R3C2,R1C2)*RC[-4]	7	4	
G7	7	7	5,964	###	=IF(F7,\$B\$3,\$B\$1)*C7	=IF(RC[-1],R3C2,R1C2)*RC[-4]	7	4	
G8	8	7	10,784	###	=IF(F8,\$B\$3,\$B\$1)*C8	=IF(RC[-1],R3C2,R1C2)*RC[-4]	7	4	
G9	9	7	5,505	###	=IF(F9,\$B\$3,\$B\$1)*C9	=IF(RC[-1],R3C2,R1C2)*RC[-4]	7	4	
G10	10	7	4,396	###	=IF(F10,\$B\$3,\$B\$1)*C10	=IF(RC[-1],R3C2,R1C2)*RC[-4]	7	4	
G11	11	7	33,697	###	=SUM(G6:G10)	=SUM(R[-5]C:R[-1]C)	32	5	
C13	13	3	5.99%	0.00%	=G11/C11	=R[-2]C[4]/R[-2]C	34	2	

Figure 21-15: The Power Utility Pak can generate a useful report that lists all formulas in a worksheet.



A trial version of Power Utility Pak is available on this book's companion CD-ROM. You can use the coupon in the back of the book to order a copy at a significant discount.

Spreadsheet Detective

The Spreadsheet Detective, available from Southern Cross Software, is a comprehensive auditing tool for Excel workbooks. For complete information (including a free evaluation version), visit the following URL:

<http://www.uq.net.au/detective/>

Excel Auditor

The Excel Auditor is available from Byg Software, based in the United Kingdom. This product includes many tools to help you identify and correct spreadsheet errors. For complete information, visit the following URL:

<http://www.bygsoftware.com>

Summary

This chapter discussed the types of formula errors that you are likely to encounter. I described how to identify such errors and some general guidelines on correcting them. I also described the auditing tools that are built into Excel, plus three third-party auditing tools that you may find helpful.

The next chapter is the first of four chapters to provide information about creating custom worksheet functions by using VBA.

Part VI

Developing Custom Worksheet Functions

CHAPTER 22

Introducing VBA

CHAPTER 23

Function Procedure Basics

CHAPTER 24

VBA Programming Concepts

CHAPTER 25

VBA Custom Function Examples

Chapter 22

Introducing VBA

IN THIS CHAPTER

- ◆ An introduction to Visual Basic for Applications – Excel’s programming language
- ◆ How to use the Visual Basic Editor
- ◆ How to work in the code windows of the Visual Basic Editor

THIS CHAPTER INTRODUCES YOU to Visual Basic for Applications (VBA). VBA is Excel’s programming language, and it is used to create custom worksheet functions. Before you can create custom functions by using VBA, you need to have some basic background knowledge of VBA, as well as some familiarity with the Visual Basic Editor.

About VBA

Excel 5 was the first application on the market to feature Visual Basic for Applications. VBA is best thought of as Microsoft’s common application scripting language. It’s now included with all Office 2002 applications, and it’s also available in applications from other vendors. In Excel, VBA has two primary uses:

- ◆ Enables you to automate spreadsheet tasks.
- ◆ Enables you to create custom functions that you can use in your worksheet formulas.



Excel also includes another way of creating custom functions by using the XLM macro language. XLM is pretty much obsolete, but it is still supported for compatibility purposes. This book completely ignores the XLM language and focuses on VBA.

VBA is a complex topic – far too complex to be covered completely in this book. Because this book deals with formulas, I hone in on one important (and useful) aspect of VBA – that of creating custom worksheet functions. You can use a custom worksheet function (sometimes known as a *user-defined function*) in formulas.



If your goal is to become a VBA expert, this book nudges you in that direction, but it does not get you to your final destination. You may want to check out another book of mine: *Excel 2002 Power Programming with VBA*. This book covers all aspects of VBA programming for Excel.

Introducing the Visual Basic Editor

Before you can begin creating custom functions, you need to become familiar with the Visual Basic Editor, or VB Editor for short. The VB Editor enables you to work with *VBA modules*, which are containers for your VBA code.

In Excel 5 and Excel 95, a VBA module appeared as a separate sheet in a workbook. Beginning with Excel 97, VBA modules no longer show up as sheets in a workbook. Rather, you use the VB Editor to view and work with VBA modules. In Excel 97 and later versions, VBA modules are still stored with workbook files; they just aren't visible unless you activate the VB Editor.



This chapter assumes that you use Excel 97 or a later version. Previous versions don't have a separate VB Editor.

Activating the VB Editor

When you work in Excel, you can switch to the VB Editor by using any of the following techniques:

- ◆ Press Alt+F11.
- ◆ Select Tools → Macro → Visual Basic Editor.
- ◆ Click the Visual Basic Editor button, located on the Visual Basic toolbar (not visible, by default).

Figure 22-1 shows the VB Editor. Chances are that your VB Editor window won't look exactly like the window shown in the figure. This window is highly customizable. You can hide windows, change their sizes, "dock" them, rearrange them, and so on.

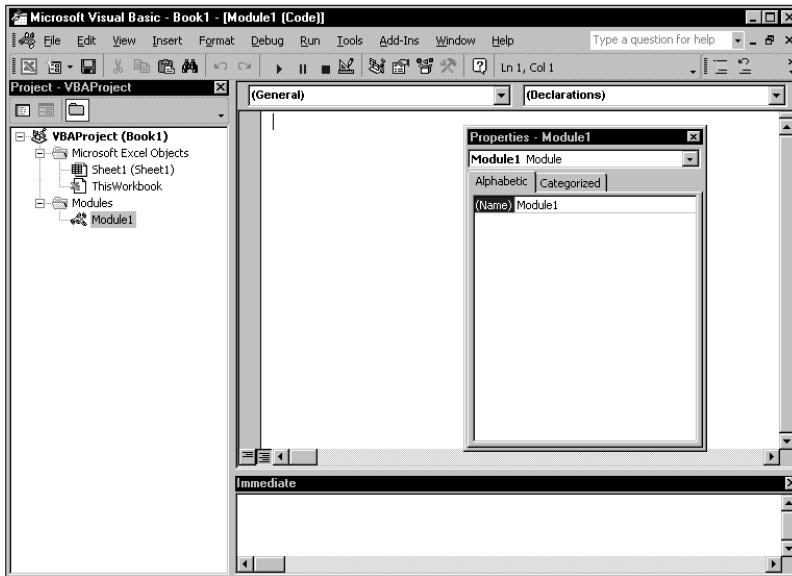


Figure 22-1: The Visual Basic Editor window

The VB Editor Components

The VB Editor consists of a number of components. I briefly describe some of the key components in the following sections.

MENU BAR

The VB Editor menu bar works like every other menu bar that you've encountered. It contains commands that you use to work with the various components in the VB Editor. The VB Editor also features shortcut menus. Right-click virtually anything in a VB Editor window and you get a shortcut menu of common commands.

TOOLBARS

The standard toolbar, directly under the menu bar by default, is one of six VB Editor toolbars that are available. VB Editor toolbars work just like those in Excel: You can customize toolbars, move them around, display other toolbars, and so forth.

PROJECT WINDOW

The Project window displays a tree diagram that consists of every workbook that's currently open in Excel (including add-ins and hidden workbooks). In the VB Editor, each workbook is known as a *project*. I discuss the Project window in more detail in the next section ("Using the Project Window"). If the Project window is not visible, press Ctrl+R.

CODE WINDOW

A code window contains VBA code. Every item in a project has an associated code window. To view a code window for an object, double-click the object in the Project window. Or, select the item and click the View Code button at the top of the Explorer window.

For example, to view the code window for the Sheet1 object for a particular workbook, double-click Sheet1 in the Project window. Unless you've added some VBA code, the code window will be empty. I discuss code windows later on in this chapter (see "Using Code Windows").

PROPERTIES WINDOW

The Properties window contains a list of all properties for the selected object. Use this window to examine and change properties. You can use the F4 shortcut key to display the Properties window.

IMMEDIATE WINDOW

The Immediate window is most useful for executing VBA statements directly, testing statements, and debugging your code. This window may or may not be visible. If the Immediate window is not visible, press Ctrl+G. To close the Immediate window, click the Close button on its title bar.

Using the Project Window

When you work in the VB Editor, each Excel workbook and add-in that's currently open is considered a project. You can think of a project as a collection of objects arranged as an outline. You can expand a project by clicking the plus sign (+) at the left of the project's name in the Project window. You contract a project by clicking the minus sign (-) to the left of a project's name. Figure 22-2 shows the Project window with three projects listed (one add-in and two workbooks).

If you try to expand a project that is protected with a password, you are prompted to enter the password.

Every project expands to show at least one "node" called "Microsoft Excel Objects." This node expands to show an item for each worksheet and chart sheet in the workbook (each sheet is considered an object), and another object called "ThisWorkbook" (which represents the Workbook object). If the project has any VBA modules, the project listing also shows a Modules node with the modules

listed there. A project may also contain a node called “Forms” (which contains UserForm objects), and a node called “Class Modules” (which contain Class Module objects). This book focuses exclusively on standard VBA modules and does not cover the objects contained in the Microsoft Excel Objects node, UserForms node, or Class Modules node.

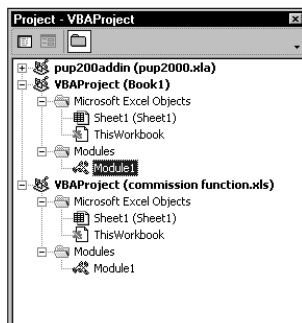


Figure 22-2: A Project window with three projects listed



If you use Excel 2002, a project may have another node called “References.” This node contains a list of all references that are used by the project. References are added or removed by using the Tools → References command. Unlike other items listed in the Project window, Reference items don’t have an associated code module.

RENAMING A PROJECT

By default, all projects are named “VBAProject.” In the Project window, the workbook name appears (in parentheses) next to the project name. For example, a project may appear as:

```
VBAProject (budget.xls)
```

You may prefer to change the name of your project to a more descriptive name. To do so:

1. Select the project in the Project window
2. Make sure that the Properties window is displayed (press F4 if it’s not displayed).
3. Change the name from VBAProject to something else

After making the change, the Project window displays the new name.

ADDING A NEW VBA MODULE

A new Excel workbook does not have any VBA modules. To add a VBA module to a project, select the project's name in the Project window and choose Insert → Module.



When you create custom functions, they *must* reside in a standard VBA module and not in a code window for a Sheet object or the ThisWorkbook object. If the code for your custom function does not reside in a VBA module, it won't work!

RENAMING A MODULE

VBA modules have default names, such as “Module1,” “Module2,” and so on. To rename a VBA module, select it in the Project window and then change the Name property by using the Properties window (a VBA module has only one property—Name). If the Properties window is not visible, press F4 to display it. Figure 22-3 shows a VBA module that is being renamed “MyModule.”

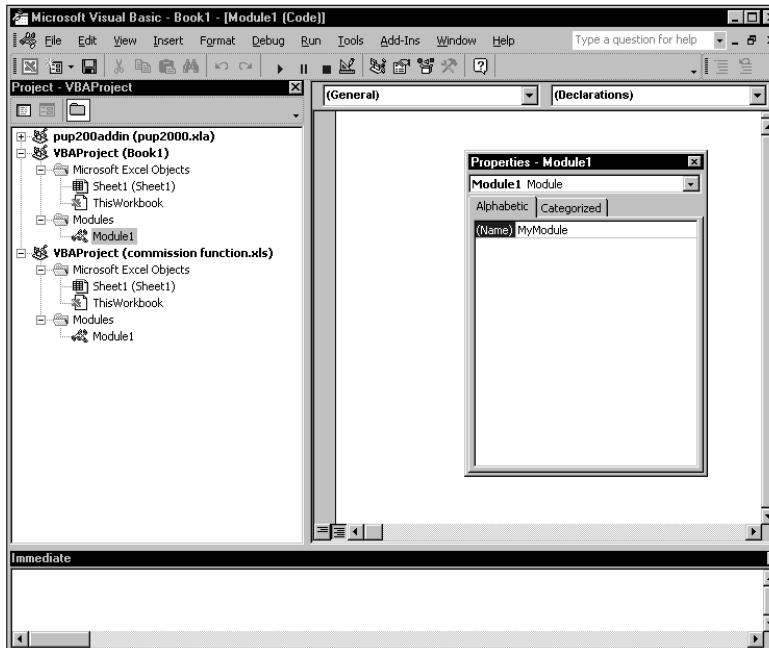


Figure 22-3: Use the Properties window to change the name of a VBA module.

REMOVING A VBA MODULE

If you want to remove a VBA module from a project, select the module's name in the Project window and choose File → Remove *.xxx*, (where *.xxx* is the name of the module). You are asked if you want to export the module before removing it. Exporting a module makes a backup file of the module's contents. You can import an exported module into any other project.

Using Code Windows

With the exception of Reference objects, each object in a project has an associated code window. To summarize, these objects can be:

- ◆ The workbook itself (the item named “ThisWorkbook” in the Project window)
- ◆ A worksheet or chart sheet in a workbook (for example, Sheet1 or Chart1 in the Project window)
- ◆ A VBA module
- ◆ A UserForm
- ◆ A *class module* (a special type of module that enables you to create new object classes)
- ◆ A reference (Excel 2002 only)



This book focuses exclusively on VBA modules, which is where custom worksheet functions are stored.

MINIMIZING AND MAXIMIZING WINDOWS

At any given time, the VB Editor may have lots of code windows. Figure 22-4 shows an example.

Code windows are much like worksheet windows in Excel. You can minimize them, maximize them, hide them, rearrange them, and so on. Most people find that it's much easier to maximize the code window that they're working on. Sometimes, however, you may want to have two or more code windows visible. For example, you may want to compare the code in two modules, or copy code from one module to another.

Minimizing a code window gets it out of the way. You also can click the Close button in a code window's title bar to close the window completely. To open it again, just double-click the appropriate object in the Project window.

You can't close a workbook from the VB Editor. You must reactivate Excel and close it from there.

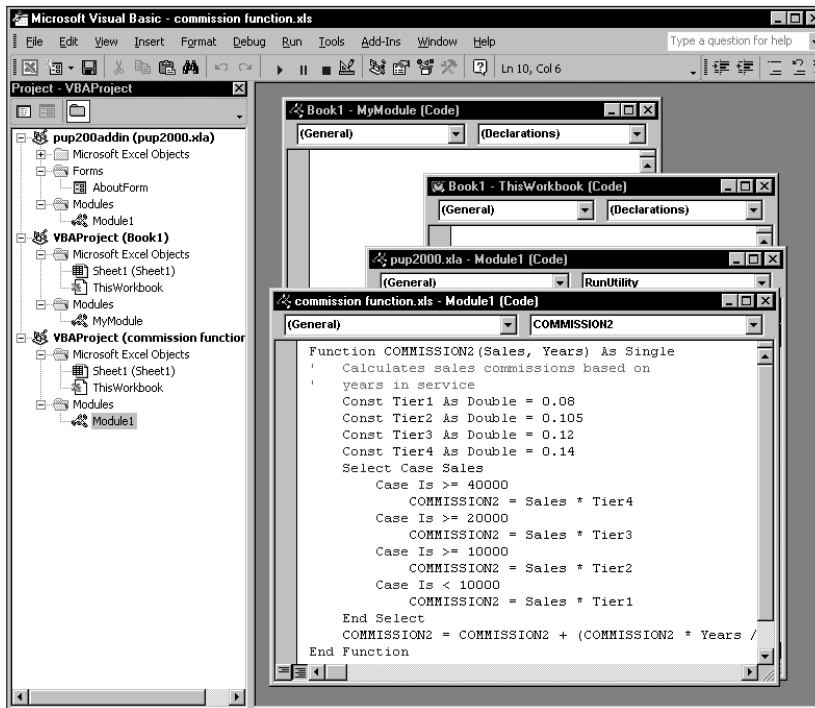


Figure 22-4: Code window overload

STORING VBA CODE

In general, a module can hold three types of code:

- ◆ **Sub procedures:** A *procedure* is a set of instructions that performs some action. For example, you may have a Sub procedure that combines various parts of a workbook into a concise report.
- ◆ **Function procedures:** A *function* is a set of instructions that returns a single value or an array. You can use Function procedures in worksheet formulas.
- ◆ **Declarations:** A *declaration* is information about a variable that you provide to VBA. For example, you can declare the data type for variables that you plan to use.

A single VBA module can store any number of procedures and declarations.



This book focuses exclusively on Function procedures. A Function procedure is the only type of procedure that you can use in worksheet formulas.

Entering VBA Code

This section describes the various ways of entering VBA code in a code window. For Function procedures, the code window will always be a VBA module. You can add code to a VBA module in three ways:

- ◆ Use your keyboard to type it.
- ◆ Use Excel's macro-recorder feature to record your actions and convert them into VBA code.
- ◆ Copy the code from another module and paste it into the module that you are working on.

ENTERING CODE MANUALLY

Sometimes, the most direct route is the best one. Type the code by using your keyboard. Entering and editing text in a VBA module works just as you expect. You can select text and copy it, or cut and paste it to another location.

Use the Tab key to indent the lines that logically belong together – for example, the conditional statements between an If and an End If statement. Indentation is not necessary, but it makes the code easier to read.

A single instruction in VBA can be as long as you want. For the sake of readability, however, you may want to break a lengthy instruction into two or more lines. To do so, end the line with a space followed by an underscore character, and then press Enter and continue the instruction on the following line. The following code, for example, is a single statement split over three lines.

```
If IsNumeric(MyCell) Then _  
    Result = "Number" Else _  
    Result = "Non-Number"
```

Notice that I indented the last two lines of this statement. Doing this is optional, but it helps to clarify the fact that these three lines comprise a single statement.

After you enter an instruction, the VB Editor performs the following actions to improve readability:

- ◆ It inserts spaces between operators. If you enter `Ans=1+2` (without any spaces), for example, VBA converts it to

```
Ans = 1 + 2
```


- ◆ The VB Editor adjusts the case of the letters for keywords, properties, and methods. If you enter the following text:

```
user=application.username
```

VBA converts it to

```
user = Application.UserName
```

- ◆ Because variable names are not case sensitive, the VB Editor adjusts the names of all variables with the same letters so that their case matches the case of letters that you most recently typed. For example, if you first specify a variable as myvalue (all lowercase) and then enter the variable as MyValue (mixed case), VBA changes all other occurrences of the variable to MyValue. An exception to this occurs if you declare the variable with Dim or a similar statement; in this case, the variable name always appears as it was declared.
- ◆ The VB Editor scans the instruction for syntax errors. If it finds an error, it changes the color of the line and may display a message describing the problem. You can set various options for the VB Editor in the Options dialog box (accessible by selecting Tools → Options).



Like Excel, the VB Editor has multiple levels of Undo and Redo. Therefore, if you find that you mistakenly deleted an instruction, you can click the Undo button (or press Ctrl+Z) repeatedly until the instruction returns. After undoing the action, you can select Edit → ReDo Delete (or click the ReDo Delete toolbar button) to redo previously undone changes.

USING THE MACRO RECORDER

Another way to get code into a VBA module is to record your actions by using Excel's macro recorder. No matter how hard you try, you cannot record a *Function procedure* (the type of procedure that is used for a custom worksheet function). All recorded macros are Sub procedures. Using the macro recorder *can* help you to identify various properties that you can use in your custom functions. For example, turn on the macro recorder to record your actions while you change the user name. Follow these steps in Excel:

1. Select Tools → Macro → Record New Macro.
2. In the Record Macro dialog box, accept the default settings and click OK to begin recording. Excel displays a small toolbar named "Stop Recording."

3. Select Tools → Options.
4. In the Options dialog box, click the General tab.
5. Make a change (any change) to the User Name box.
6. Click OK to close the Options dialog box.
7. Click the Stop Recording button on the Stop Recording toolbar.
8. Press Alt+F11 to activate the VB Editor.
9. In the Project window, select the project that corresponds to your workbook.
10. Double-click the VBA module that contains your recorded code. Generally, this will be the module with the highest number (for example, Module3).

You'll find a VBA procedure that looks something like this:

```
Sub Macro1()  
,  
,  
, Macro1 Macro  
, Macro recorded 6/1/2001 by Bob Smith  
,  
    With Application  
        .UserName = "Robert Smith"  
        .StandardFont = "Arial"  
        .StandardFontSize = "10"  
        .DefaultFilePath = "d:\xlfiles"  
        .EnableSound = False  
        .RollZoom = False  
    End With  
End Sub
```

Note that this is a Sub procedure, not a Function procedure. In other words, you can't use this procedure in a worksheet formula. If you examine the code, however, you'll see a reference to the UserName property. You can use this information when creating a Function procedure. For example, the following Function procedure uses the UserName property. This function, when used in a worksheet formula, returns the name of the user.

```
Function USER()  
    USER = Application.UserName  
End Function
```

You can consult the online help system to identify various properties, but using the macro recorder is often more efficient if you don't know exactly what you're looking for. After you identify what you need, you can check the online help for details.

COPYING VBA CODE

This section has covered entering code directly and recording your actions to generate VBA code. The final method of getting code into a VBA module is to copy it from another module. For example, you may have written a custom function for one project that would also be useful in your current project. Rather than reenter the code, you can open the workbook, activate the module, and use the normal Clipboard copy-and-paste procedures to copy it into your current VBA module.

You also can copy VBA code from other sources. For example, you may find a listing on a Web page or in a newsgroup. In such a case, you can select the text in your browser (or newsreader), copy it to the Clipboard, and then paste it into a module.

Saving Your Project

As with any application, you should save your work frequently while working in the VB Editor. To do so, use the File → Save command, press Ctrl+S, or click the Save button on the standard toolbar.



When you save your project, you actually save your Excel workbook. By the same token, if you save your workbook in Excel, you also save the changes made in the workbook's VB project.

The VB Editor does not have a File → Save As command. To save your project with a different name, activate Excel and use Excel's File → Save As command.

Summary

This chapter provided an introduction to VBA, which is the language used to create custom worksheet functions. I introduced the various components of the VB Editor and described several ways to enter code into a VBA module.

The next chapter covers the basics of VBA Function procedures.

Chapter 23

Function Procedure Basics

IN THIS CHAPTER

- ◆ Why you may want to create custom functions
- ◆ An introductory VBA function example
- ◆ About VBA Function procedures
- ◆ Using the Paste Function dialog box to add a function description and assign a function category
- ◆ Tips for testing and debugging functions
- ◆ Creating an add-in to hold your custom functions

PREVIOUS CHAPTERS IN THIS book examined Excel's worksheet functions and how you can use them to build more complex formulas. These functions, as well as those available in the Analysis ToolPak add-in, provide a great deal of flexibility when creating formulas. However, you may encounter situations that call for custom functions. This chapter discusses the reasons that you may want to use custom functions, how you can create a VBA Function procedure, and methods for testing and debugging them.

Why Create Custom Functions?

You are, of course, familiar with Excel's worksheet functions – even novices know how to use the most common worksheet functions, such as SUM, AVERAGE, and IF. Excel includes more than 300 predefined worksheet functions, plus additional functions available through the Analysis ToolPak add-in.

You can use Visual Basic for Applications (VBA) to create additional worksheet functions, which are known as *custom functions* or *user-defined functions* (UDFs). With all the functions that are available in Excel and VBA, you may wonder why you would ever need to create new functions. The answer: To simplify your work and give your formulas more power.

For example, you can create a custom function that can significantly shorten your formulas. Shorter formulas are more readable and easier to work with. However, it's important to understand that custom functions in your formulas are usually much slower than built-in functions. But on a fast system, the speed difference often goes unnoticed.

The process of creating a custom function is not difficult. In fact, many people (this author included) *enjoy* creating custom functions. This book provides you with the information that you need to create your own functions. In this and subsequent chapters, you'll find many custom function examples that you can adapt for your own use.

An Introductory VBA Function Example

Without further ado, I'll show you a simple VBA Function procedure. This function, named `USER`, does not accept any arguments. When used in a worksheet formula, this function simply displays the user's name, in uppercase characters. To create this function:

1. Start with a new workbook (this is not really necessary, but keep it simple for right now).
2. Press `Alt+F11` to activate the VB Editor.
3. Click your workbook's name in the Project window. If the Project window is not visible, press `Ctrl+R` to display it.
4. Choose `Insert → Module` to add a VBA module to the project.
5. Type the following code in the code window.

```
Function USER()  
' Returns the user's name  
    USER = Application.UserName  
    USER = UCase(USER)  
End Function
```



I gave this warning in the previous chapter, but it's worth repeating: When you create a custom function, make sure that it resides in a normal VBA module and not in a code module for a Sheet or ThisWorkbook.

To try out the `User` function, activate Excel (press `Alt+F11`) and enter the following formula into any cell in the workbook.

`=USER()`

What Custom Worksheet Functions Can't Do

As you develop custom worksheet functions, you should understand a key point. A function procedure used in a worksheet formula must be *passive*. In other words, it can't change things in the worksheet.

You may be tempted to try to write a custom worksheet function that changes the formatting of a cell. For example, you may find it useful to have a function that changes the color of text in a cell based on the cell's value. Try as you might, a function such as this is impossible to write — everybody tries this, and no one succeeds. No matter what you do, the function always returns an error because the code attempts to change something on the worksheet. Remember that a function can return only a value. It can't perform actions with objects.

If you entered the VBA code correctly, the Function procedure executes and your name displays (in uppercase characters) in the cell.



If your formula returns an error, make sure that the VBA code for the USER function is in a VBA module (and not a module for a Sheet or ThisWorkbook object). Also, make sure that the module is in the project associated with the workbook into which you enter the formula.

When Excel calculates your worksheet, it encounters the USER custom function. Each instruction in the function is evaluated, and the result is returned to your worksheet. You can use this function any number of times in any number of cells.

You'll find that this custom function works just like any other worksheet function. You can insert it into a formula by using the Insert → Function command or the Insert Function button. In the Insert Function dialog box, custom functions appear in the User Defined category. As with any other function, you can use it in a more complex formula. For example, try this:

```
= "Hello " & USER()
```

Or, use this formula to display the number of characters in your name:

```
=LEN(USER())
```

If you don't like the fact that your name is in uppercase, edit the procedure as follows:

```
Function USER()  
' Returns the user's name  
  USER = Application.UserName  
End Function
```

After editing the function, reactivate Excel and press F9 to recalculate. Any cell that uses the USER function displays a different result.

About Function Procedures

Function procedures have a structure. Here, we'll look at some of the technical details that apply to Function procedures. These are general guidelines for declaring functions, naming functions, using custom functions in formulas, and using arguments in custom functions.

Declaring a Function

The official syntax for declaring a function is as follows:

```
[Public | Private][Static] Function name ([arglist]) [As type]  
  [statements]  
  [name = expression]  
  [Exit Function]  
  [statements]  
  [name = expression]  
End Function
```

- ◆ *Public* indicates that the function is accessible to all other procedures in all other modules in the workbook (optional).
- ◆ *Private* indicates that the function is accessible only to other procedures in the same module (optional). If you use the Private keyword, your functions won't appear in the Insert Function dialog box.
- ◆ *Static* indicates that the values of variables declared in the function are preserved between calls (optional).
- ◆ *Function* is a keyword that indicates the beginning of a Function procedure (required).

- ◆ *Name* can be any valid variable name. When the function finishes, the result of the function is the value assigned to the function's name (required).
- ◆ *Arglist* is a list of one or more variables that represent arguments passed to the function. The arguments are enclosed in parentheses. Use a comma to separate arguments. (Arguments are optional.)
- ◆ *Type* is the data type returned by the function (optional).
- ◆ *Statements* are valid VBA statements (optional).
- ◆ *Exit Function* is a statement that causes an immediate exit from the function (optional).
- ◆ *End Function* is a keyword that indicates the end of the function (required).

Choosing a Name for Your Function

Each function must have a unique name, and function names must adhere to a few rules:

- ◆ You can use alphabetic characters, numbers, and some punctuation characters, but the first character must be alphabetic.
- ◆ You can use any combination of uppercase and lowercase letters.
- ◆ You can't use a name that looks like a worksheet cell's address (such as J21). Actually, you *can* use such a name for a function, but Excel won't interpret it as a function.
- ◆ VBA does not distinguish between cases. To make a function name more readable, you can use mixed cases (InterestRate rather than interestrate).
- ◆ You can't use spaces or periods. To make function names more readable, you can use the underscore character (Interest_Rate).
- ◆ The following characters can't be embedded in a function's name: #, \$, %, &, or !. These are type declaration characters that have a special meaning in VBA.
- ◆ A function name can consist of as many as 255 characters – but nobody creates function names that long!

Using Functions in Formulas

Using a custom VBA function in a worksheet formula is like using a built-in worksheet function, except that you must ensure that Excel can locate the Function procedure. If the Function procedure is in the same workbook as the formula, you don't have to do anything special. If it's in a different workbook, you may have to tell Excel where to find it. You can do so in three ways:

- ◆ **Precede the function's name with a file reference.** For example, if you want to use a function called CountNames that's defined in a workbook named Myfuncs.xls, you can use a formula like the following:

```
=Myfuncs.xls!CountNames(A1:A1000)
```

If you insert the function with the Paste Function dialog box, the workbook reference is inserted automatically.

- ◆ **Set up a reference to the workbook.** You do this with the VB Editor's Tools' References command (see Figure 23-1). If the function is defined in a referenced workbook, you don't need to use the worksheet name. Even when the dependent workbook is assigned as a reference, the Insert Function dialog box continues to insert the workbook reference (even though it's not necessary).

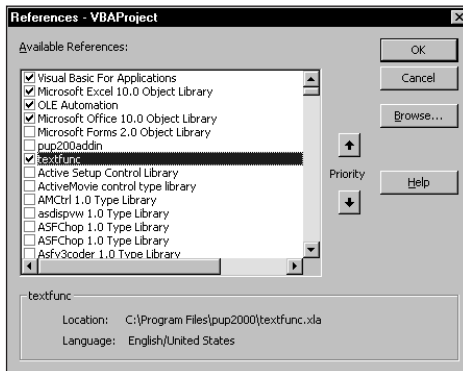


Figure 23-1: Use the References dialog box to create a reference to a project that contains a custom VBA function.



By default, all projects are named VBAProject — and that’s the name that appears in the Available References list in the References dialog box. To make sure that you select the correct project in the References dialog box, keep your eye on the bottom of the dialog box, which shows the workbook name for the selected item. Better yet, change the name of the project to be more descriptive. To change the name, select the project, press F4 to display the Properties window, and then change the Name property to something other than VBAProject.

- ◆ **Create an add-in.** When you create an add-in from a workbook that has Function procedures, you don’t need to use the file reference when you use one of the functions in a formula; however, the add-in must be installed. I discuss add-ins later in this chapter (see “Creating Add-Ins”).

Using Function Arguments

Custom functions, like Excel’s built-in functions, vary in their use of arguments. Keep the following points in mind regarding VBA Function procedure arguments:

- ◆ A function can have no argument.
- ◆ A function can have a fixed number of required arguments (from 1 to 60).
- ◆ A function can have a combination of required and optional arguments.



See Chapter 23 for examples of functions that use various types of arguments.

Using the Insert Function Dialog Box

Excel’s Insert Function dialog box is a handy tool that enables you to choose a particular worksheet function from a list of available functions. The Insert Function dialog box also displays a list of your custom worksheet functions and prompts you for the function’s arguments.



Custom Function procedures defined with the Private keyword don't appear in the Paste Function dialog box.

By default, custom functions are listed under the User Defined category, but you can have them appear under a different category. You also can add some text that describes the function.

Adding a Function Description

When you select a function in the Paste Function dialog box, a brief description of the function appears (see Figure 23-2).

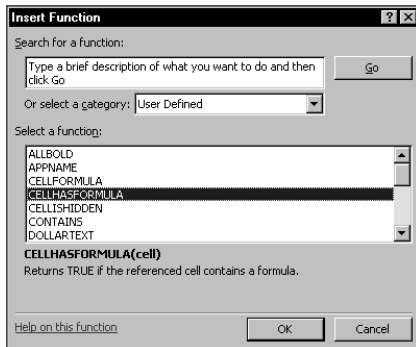


Figure 23-2: Excel's Insert Function dialog box displays a brief description of the selected function.



If you don't provide a description for your custom function, the Insert Function dialog box displays the following text: *Choose the help button for help on this function and its arguments.* In Excel 2002, the message is more accurate: *No help available.*

The following steps describe how to provide a description for a custom function.

1. Create your function in the VB Editor.
2. Activate Excel, and select Tools → Macro → Macros (or press Alt+F8). The Macro dialog box lists available Sub procedures, but not functions.

3. Type the name of your function in the Macro Name box. Make sure that you spell it correctly.
4. Click the Options button to display the Macro Options dialog box. If the Options button is not disabled, you probably spelled the function's name incorrectly.
5. Enter the function description in the Description box (see Figure 23-3). The Shortcut key field is irrelevant for functions.
6. Click OK, and then click Cancel.

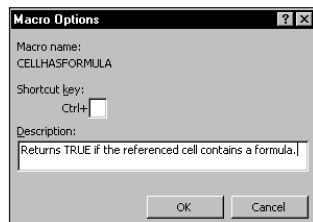


Figure 23-3: Providing a function description in the Macro Options dialog box



When you use the Insert Function dialog box to enter a function, the Function Arguments dialog box kicks in after you click OK. For built-in functions, the Function Arguments dialog displays a description for each of the function's arguments. Unfortunately, it is not possible to provide descriptions like these for your custom function arguments.

Specifying a Function Category

Oddly, Excel does not provide a direct way to assign a custom function to a particular function category. If you want your custom function to appear in a function category other than User Defined, you need to execute some VBA code in order to do so.

For example, assume that you've created a custom function named COMMISSION, and assume that you want this function to appear in the Financial category (that is, category 1) in the Insert Function dialog box. To accomplish this, you will need to execute the following VBA statement:

```
Application.MacroOptions Macro:="COMMISSION", Category:=1
```

One way to execute this statement is to use the Immediate window in the VB Editor. Figure 23-4 shows an example. Just type the statement and press Enter. Then, save the workbook, and the category assignment is also stored in the workbook. This statement needs to be executed only one time. In other words, it is not necessary to assign the function to a new category every time the workbook is opened.

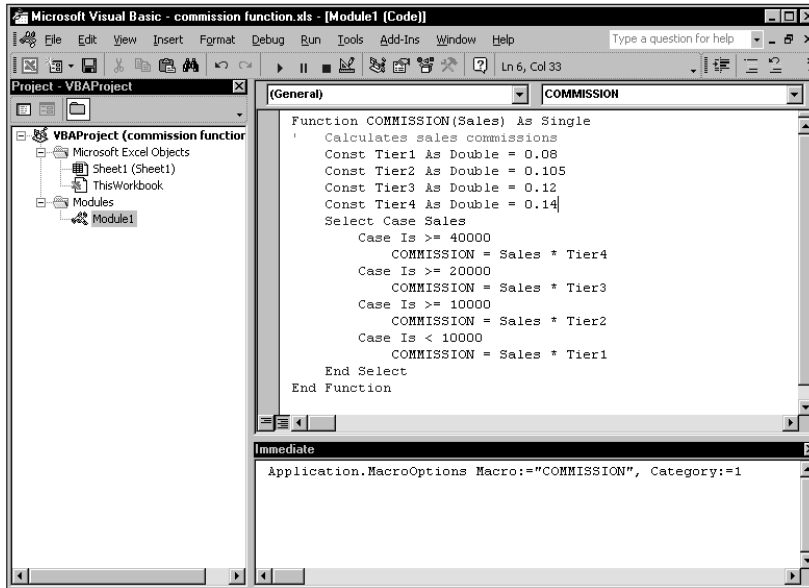


Figure 23-4: Executing a VBA statement that assigns a function to a particular function category

Alternatively, you can create a Sub procedure, and then execute the procedure.

```
Sub AssignToFunctionCategory()
    Application.MacroOptions Macro:="COMMISSION", Category:=1
End Sub
```

After you've executed the procedure, you can delete it.

You will, of course, substitute the actual name of your function, and you can specify a different function category. The AssignToFunctionCategory procedure can contain any number of statements – one for each of your functions.

Table 23-1 lists the function category numbers that you can use. Notice that a few of these categories (10–13) normally don't display in the Insert Function dialog box. If you assign your function to one of these categories, the category then appears.

TABLE 23-1 FUNCTION CATEGORIES

Category Number	Category Name
0	All (no specific category)
1	Financial
2	Date & Time
3	Math & Trig
4	Statistical
5	Lookup & Reference
6	Database
7	Text
8	Logical
9	Information
10	Commands
11	Customizing
12	Macro Control
13	DDE/External
14	User Defined
15	Engineering

Testing and Debugging Your Functions

Naturally, testing and debugging your custom function is an important step that you must take to ensure that it carries out the calculation that you intend. This section describes some debugging techniques that you may find helpful.



If you're new to programming, the information in this section will make a lot more sense after you're familiar with the material in Chapter 22.

VBA code that you write can contain three general types of errors:

- ◆ **Syntax errors:** An error in writing the statement – for example, a misspelled keyword, a missing operator, or mismatched parentheses. The VB Editor lets you know about syntax errors by displaying a pop-up error box. You can't use the function until you correct all syntax errors.
- ◆ **Runtime errors:** Errors that occur as the function executes. For example, attempting to perform a mathematical operation on a string variable generates a runtime error. Unless you spot it beforehand, you won't be aware of a runtime error until it occurs.
- ◆ **Logical errors:** Code that runs, but simply returns the wrong result



To force the code in a VBA module to be checked for syntax errors, select Debug → Compile. This highlights the first syntax error, if any exists. Correct the error and issue the command again until you find all of the errors.

An error in code is sometimes called a *bug*. The process of locating and correcting such an error is known as *debugging*.

When you test a Function procedure by using a formula in a worksheet, runtime errors can be difficult to locate because (unlike syntax errors) they don't appear in a pop-up error box. If a runtime error occurs, the formula that uses the function simply returns an error value (#VALUE!). This section describes several approaches to debugging custom functions.



When you test a custom function, it's a good idea to use the function in only one formula in the worksheet. If you use the function in more than one formula, the code is executed for each formula.

Using VBA's MsgBox Statement

The MsgBox statement, when used in your VBA code, displays a pop-up box. You can use MsgBox statements at strategic locations within your code to monitor the value of specific variables. The following example is a Function procedure that should reverse a text string passed as its argument. For example, passing *Hello* as the argument should return *olleH*. If you try to use this function in a formula, however, you will see that it does not work – it contains a logical error.

```

Function REVERSETEXT(text) As String
' Returns its argument, reversed
  TextLen = Len(text)
  For i = TextLen To 1 Step -1
    REVERSETEXT = Mid(text, i, 1) & REVERSETEXT
  Next i
End Function

```

You can insert a temporary MsgBox statement to help you figure out the source of the problem. Here's the function again, with the MsgBox statement inserted within the loop:

```

Function REVERSETEXT(text) As String
' Returns its argument, reversed
  TextLen = Len(text)
  For i = TextLen To 1 Step -1
    REVERSETEXT = Mid(text, i, 1) & REVERSETEXT
    MsgBox REVERSETEXT
  Next i
End Function

```

When this function is evaluated, a pop-up message box appears, once for each time through the loop. The message box shows the current value of REVERSETEXT. In other words, this technique enables you to monitor the results as the function is executed. Figure 23-5 shows an example.

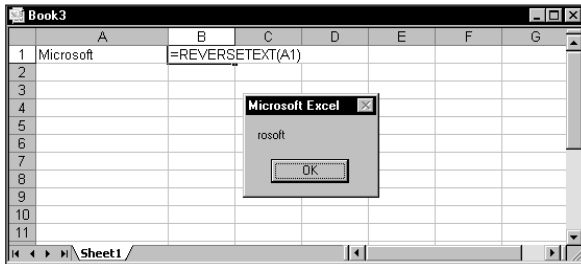


Figure 23-5: Use a MsgBox statement to monitor the value of a variable as a Function procedure executes.

The information displayed in the series of message boxes shows that the text string is being built within the loop, but the new text is being added to the beginning of the string, not the end. The corrected assignment statement is:

```
REVERSETEXT = REVERSETEXT & Mid(text, i, 1)
```


When the function is working properly, make sure that you remove all of the `MsgBox` statements. They get very annoying.

To display more than one variable in a message box, you need to concatenate the variables and insert a space character between each variable. The statement below, for example, displays the value of three variables (`x`, `y`, and `z`) in a message box.

```
MsgBox x & " " & y & " " & z
```

If you omit the blank space, you can't distinguish the separate values.

Using `Debug.Print` Statements in Your Code

If you find that using `MsgBox` statements is too intrusive, a better option is to insert some temporary code that writes values directly to VB Editor's Immediate window (see the sidebar, "Using the Immediate Window"). You use the `Debug.Print` statement to write the values of selected variables.

For example, if you want to monitor a value inside a loop, use a routine like the following:

```
Function VOWELCOUNT(r)
    Count = 0
    For i = 1 To Len(r)
        Ch = UCase(Mid(r, i, 1))
        If Ch Like "[AEIOU]" Then
            Count = Count + 1
            Debug.Print Ch, i
        End If
    Next i
    VOWELCOUNT = Count
End Function
```

In this case, the value of two variables (`Ch` and `i`) print to the Immediate window whenever the `Debug.Print` statement is encountered. Figure 23-6 shows the result when the function has an argument of *California*.

When your function is debugged, make sure that you remove the `Debug.Print` statements.

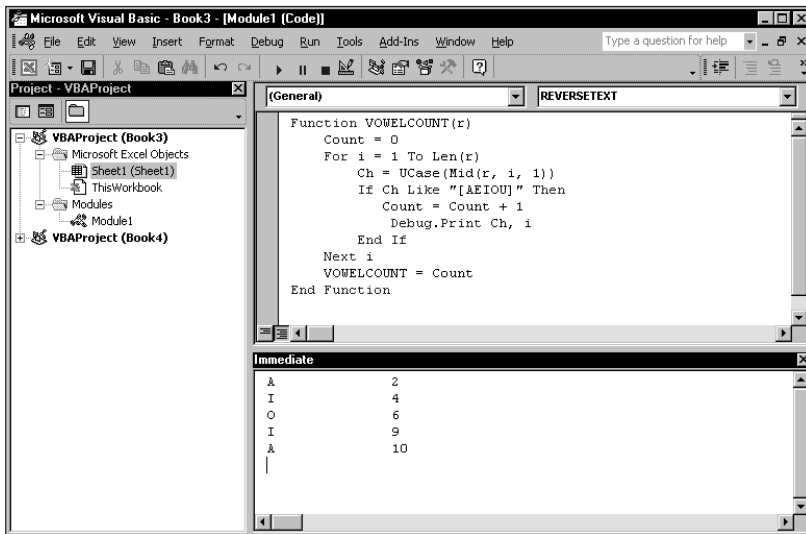


Figure 23-6: Using the VB Editor's Immediate window to display results while a function is running

Calling the Function from a Sub Procedure

Another way to test a Function procedure is to call the function from a Sub procedure. To do this, simply add a temporary Sub procedure to the module and insert a statement that calls your function. This is particularly useful because runtime errors display as they occur.

The following Function procedure contains an error (a runtime error). As I noted previously, the runtime errors don't display when testing a function by using a worksheet formula. Rather, the function simply returns an error (#VALUE!).

```
Function REVERSETEXT(text) As String
    ' Returns its argument, reversed
    TextLen = Len(text)
    For i = TextLen To 1 Step -1
        REVERSETEXT = REVERSETEXT And Mid(text, i, 1)
    Next i
End Function
```

To help identify the source of the runtime error, insert the following Sub procedure:

```
Sub Test()
    x = REVERSETEXT("Hello")
    MsgBox x
End Sub
```

This Sub procedure simply calls the REVERSETEXT function and assigns the result to a variable named x. The MsgBox statement displays the result.

You can execute the Sub procedure directly from the VB Editor. Simply move the cursor anywhere within the procedure and select Run → Run Sub/UserForm (or, just press F5). When you execute the Test procedure, you see the error message that is shown in Figure 23-7.

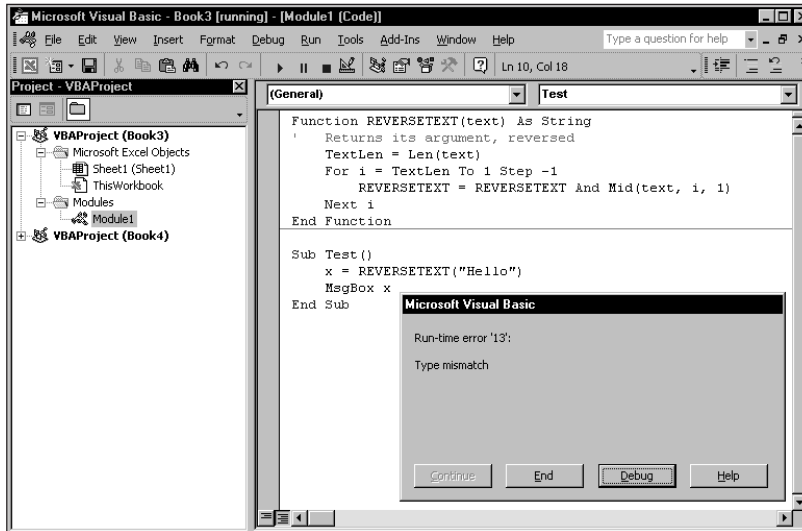


Figure 23-7: A runtime error identified by VBA

Click the Debug button, and the VB Editor highlights the statement causing the problem (see Figure 23-8). The error message does not tell you how to correct the error, but it does narrow your choices. After you've identified the statement that's causing the error, you can examine it more closely or use the Immediate window (see the sidebar, "Using the Immediate Window") to help locate the exact problem.

In this case, the problem is the user of the And operator instead of the concatenation operator (&). The correct statement is:

```
REVERSETEXT = REVERSETEXT & Mid(text, i, 1)
```



When you click the Debug button, the procedure is still running — it's just halted and is in "break mode." After you make the correction, press F5 to continue execution, press F8 to continue execution on a line-by-line basis, or click the Rest button to halt execution.

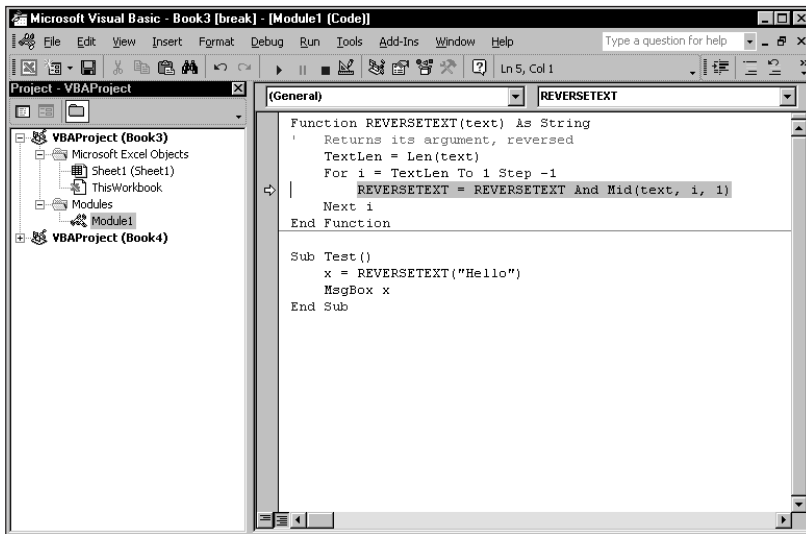


Figure 23-8: The highlighted statement has generated a runtime error.

Using the Immediate Window

The VB Editor's Immediate window can be helpful when debugging code. To activate the Immediate window, press Ctrl+G.

You can type VBA statements in the Immediate window and see the result immediately. For example, type the following code in the Immediate window and press Enter:

```
Print Sqr(1156)
```

The VB Editor prints the result of this square root operation (34). To save a few keystrokes, you can use a single question mark (?) in place of the Print keyword.

The Immediate window is particularly useful for debugging runtime errors when VBA is in break mode. For example, you can use the Immediate window to check the current value for variables, or to check the data type of a variable.

Errors often occur because data is of the wrong type. The following statement, for example, displays the data type of a variable named Counter (which you probably think is an Integer variable).

```
? TypeName(Counter)
```

If you discover that Counter is of a data type other than Integer, you may have solved your problem.

Setting a Breakpoint in the Function

Another debugging option is to set a breakpoint in your code. Execution pauses when VBA encounters a breakpoint. You can then use the Immediate window to check the values of variable, or you can use F8 to step through your code line by line.

To set a breakpoint, move the cursor to the statement at which you want to pause execution, and select Debug → Toggle Breakpoint. Alternatively, you can press F9, or click the vertical bar to the left of the code window. Any of these actions highlights the statement to remind you that a breakpoint is in effect (you also see a dot in the code window margin). You can set any number of breakpoints in your code. To remove a breakpoint, move the cursor to the statement and press F9. Figure 23-9 shows a Function procedure that contains a breakpoint.

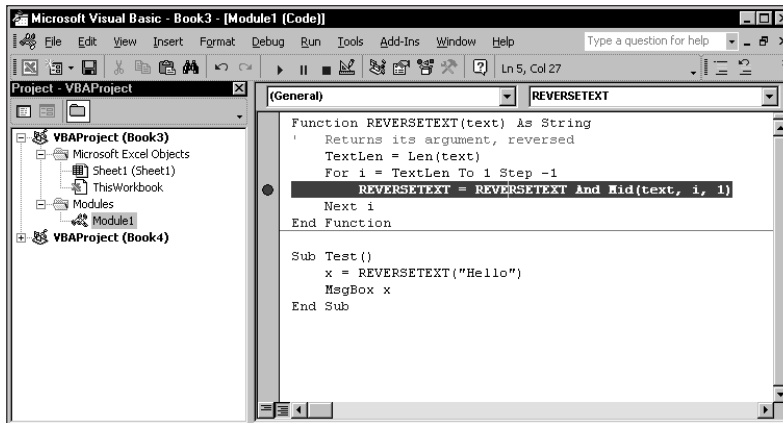


Figure 23-9: The highlighted statement contains a breakpoint.

Creating Add-Ins

If you create some custom functions that you use frequently, you may want to store these functions in an add-in file. A primary advantage to this is that you can use the functions in formulas in any workbook without a filename qualifier.

Assume that you have a custom function named ZAPSPACES and that it's stored in Myfuncs.xls. To use this function in a formula in a workbook other than Myfuncs.xls, you need to enter the following formula:

```
=Myfuncs.xls!ZAPSPACES(A1:C12)
```

If you create an add-in from Myfuncs.xls and the add-in is loaded, you can omit the file reference and enter a formula like the following:

```
=ZAPSPACES(A1:C12)
```

Creating an add-in from a workbook is simple. The following steps describe how to create an add-in from a normal workbook file:

1. Develop your functions, and make sure that they work properly.
2. Activate the VB Editor and select the workbook in the Project window. Choose Tools → .xxx Properties and click the Protection tab. Select the Lock Project for Viewing checkbox and enter a password (twice). Click OK.

You only need to do this step if you want to prevent others from viewing or modifying your macros or custom dialog boxes.

3. Reactivate Excel. Choose File → Properties, click the Summary tab, and enter a brief, descriptive title in the Title field and a longer description in the Comments field.

This step is not required, but it makes the add-in easier to use by displaying descriptive text in the Add-Ins dialog box.

4. Select File → Save As.
5. In the Save As dialog box, select Microsoft Excel add-in (*.xla) from the Save As Type drop-down list (see Figure 23-10).
6. If you don't want to use the default directory, select a different directory.
7. Click Save. A copy of the workbook is saved (with an .xla extension), and the original XLS workbook remains open.

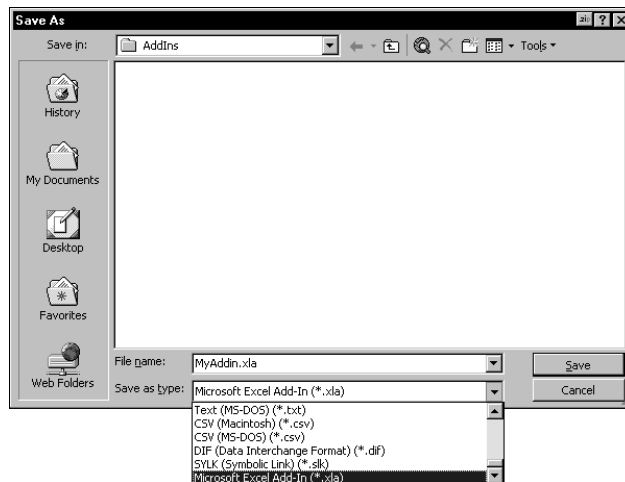


Figure 23-10: Saving a workbook as an add-in

A Few Words about Security

Microsoft has never promoted Excel as a product that creates applications with secure source code. The password feature provided in Excel is sufficient to prevent casual users from accessing parts of your application that you want to keep hidden. But, the truth is, several password-cracking utilities are available. The security features in Excel 2002 are much better than those in previous versions, but it's possible that these can also be cracked. If you must absolutely be sure that no one ever sees your code or formulas, Excel is not your best choice as a development platform.



A workbook that you convert to an add-in must have at least one worksheet. For example, if your workbook contains only chart sheets or Excel 5/95 dialog sheets, the Microsoft Excel add-in (*.xla) option does not appear in the Save As dialog box.



With previous versions of Excel (before Excel 97), to modify an add-in, you had to open the original XLS file, make your changes, and then recreate the add-in. For Excel 97 and later versions, this is no longer necessary. As long as the add-in is not protected, you can make changes to the add-in in the VB Editor, and then save your changes. If the add-in is protected, you must enter the password to unprotect it. Therefore, with Excel 97 or later, keeping an XLS version of your add-in is not necessary.

After you create your add-in, you can install it by using the standard procedure: Select Tools → Add-Ins, and click the Browse button in the Add-Ins dialog box. Then, locate your *.xla file.

Summary

This chapter covered some essential details to help you develop effective custom functions. I discussed the type of arguments that you can use, and I described how to make your function appear in a specific category in the Paste Function dialog box. This chapter also presented some techniques to help debug functions and ended with instructions for creating an add-in to hold your functions.

The next chapter discusses VBA programming concepts.

Chapter 24

VBA Programming Concepts

IN THIS CHAPTER

- ◆ Introducing an example function procedure
- ◆ Using comments in your code
- ◆ Understanding VBA's language elements, including variables, data types, and constants
- ◆ Using assignment expressions in your code
- ◆ Declaring arrays and multidimensional arrays
- ◆ Using VBA's built-in functions
- ◆ Controlling the execution of your Function procedures
- ◆ Using ranges in your code

THIS CHAPTER DISCUSSES SOME of the key language elements and programming concepts in VBA. If you've used other programming languages, then much of this information may sound familiar. VBA has a few unique wrinkles, however, so even experienced programmers may find some new information.

An Introductory Example Function Procedure

To get the ball rolling, I'll begin with an example Function procedure. This function, named `REMOVESPACES`, accepts a single argument and returns that argument without any spaces. For example, the following formula uses the `REMOVESPACES` function and returns *ThisIsATest*.

```
=REMOVESPACES("This Is A Test")
```


To create this function, insert a VBA module into a project, and then enter the following Function procedure into the code window of the module:

```
Function REMOVESPACES(cell) As String
' Removes all spaces from cell
  Dim CellLength As Integer
  Dim Temp As String
  Dim i As Integer
  CellLength = Len(cell)
  Temp = ""
  For i = 1 To CellLength
    Character = Mid(cell, i, 1)
    If Character <> Chr(32) Then Temp = Temp & Character
  Next i
  REMOVESPACES = Temp
End Function
```

Look closely at this function's code line by line:

- ◆ The first line of the function is called the function's *declaration line*. Notice that the procedure starts with the keyword *Function*, followed by the name of the function (REMOVESPACES). This function uses only one argument (cell); the argument's name is enclosed in parentheses. *As String* defines the data type of the function's return value. The "As" part of the function declaration is optional.
- ◆ The second line is simply a comment (optional) that describes what the function does. The initial apostrophe designates this line as a comment.
- ◆ The next three lines use the Dim keyword to declare the three variables used in the procedure: CellLength, Temp, and i. Declaring a variable is not necessary, but (as you'll see later) it's an excellent practice.
- ◆ The procedure's next line assigns a value to a variable named CellLength. This statement uses VBA's Len function to determine the length of the contents of the argument (cell).
- ◆ The next statement creates a variable named Temp and assigns it an empty string.
- ◆ The next four statements comprise a For-Next loop. The statements between the For statement and the Next statement are executed a number of times; the value of CellLength determines the number of times. For example, assume the cell passed as the argument contains the text "Bob Smith." The statements within the loop would execute nine times, one time for each character in the string.

- ◆ Within the loop, the Character variable holds a single character that is extracted using VBA's Mid function (which works just like Excel's MID function). The If statement determines whether the character is not a space (VBA's Chr function is equivalent to Excel's CHAR function, and an argument of 32 represents a space character). If the character is not a space, then the character is appended to the end of the string stored in the Temp variable. If the character is a space, the Temp variable is unchanged and the next character is processed. If you prefer, you can replace this statement with the following:

```
If Character <> " ") Then Temp = Temp & Character
```

- ◆ When the loop finishes, the Temp variable holds all of the characters that were originally passed to the function in the cell argument, except for the spaces.
- ◆ The string contained in the Temp variable is assigned to the function's name. This string is the value that the function returns.
- ◆ The Function procedure ends with an End Function statement.

The REMOVESPACES procedure uses some common VBA language elements, including:

- ◆ A comment (the line preceded by the apostrophe)
- ◆ Variable declarations
- ◆ Three assignment statements
- ◆ Three built-in VBA functions (Len, Mid, and Chr)
- ◆ A looping structure (For-Next)
- ◆ An If-Then structure
- ◆ String concatenation (using the & operator)

Not bad for a first effort, eh? The remainder of this chapter provides more information on these (and many other) programming concepts.



The REMOVESPACES function listed here is for instructional purposes only. You can accomplish the same effect by using Excel's SUBSTITUTE function, which is much more efficient than using a custom VBA function. The following formula, for example, removes all space characters from the text in cell A1.

```
=SUBSTITUTE(A1," ","")
```

Using Comments in Your Code

A *comment* is descriptive text embedded within your code. VBA completely ignores the text of a comment. It's a good idea to use comments liberally to describe what you do (because the purpose of a particular VBA instruction is not always obvious).

You can use a complete line for your comment, or you can insert a comment *after* an instruction on the same line. A comment is indicated by an apostrophe. VBA ignores any text that follows an apostrophe up until the end of the line. An exception occurs when an apostrophe is contained within quotation marks. For example, the following statement does not contain a comment, even though it has an apostrophe:

```
Result = "Can't calculate"
```

The following example shows a VBA Function procedure with three comments:

```
Function MYFUNC()  
' This function does nothing of value  
  x = 0 'x represents nothingness  
' Return the result  
  MYFUNC = x  
End Function
```

When developing a function, you may want to test it without including a particular instruction or group of instructions. Instead of deleting the instruction, simply convert it to a comment by inserting an apostrophe at the beginning. VBA then ignores the instruction(s) when the routine is executed. To convert the comment back to an instruction, delete the apostrophe.



The VB Editor's Edit toolbar contains two very useful buttons. Select a group of instructions and then use the Comment Block button to convert the instructions to comments. The Uncomment Block button converts a group of comments back to instructions.

Using Variables, Data Types, and Constants

A *variable* is a named storage location in your computer's memory. Variables can accommodate a wide variety of *data types*—from simple Boolean values (TRUE or FALSE) to large, double-precision values (see the following section). You assign a value to a variable by using the assignment operator, which is an equal sign.

The following are some examples of assignment statements that use various types of variables. The variable names are to the left of the equal sign. Each statement assigns the value to the right of the equal sign to the variable on the left.

```
x = 1
InterestRate = 0.075
LoanPayoffAmount = 243089
DataEntered = False
x = x + 1
MyNum = YourNum * 1.25
HallofFamer = "Tony Gwynn"
DateStarted = #3/14/2001#
```

VBA has many *reserved words*, which are words that you can't use for variable or procedure names. If you attempt to use one of these words, you get an error message. For example, although the reserved word *Next* may make a very descriptive variable name, the following instruction generates a syntax error:

```
Next = 132
```

Unfortunately, sometimes syntax error messages aren't descriptive. The preceding instruction generates a syntax error in Excel 2002 (earlier versions of Excel may produce a different error). So if an assignment statement produces an error that does not seem to make sense, check the online help to make sure that your variable name does not have a special use in VBA.

Defining Data Types

VBA makes life easy for programmers because it can automatically handle all of the details involved in dealing with data. *Data type* refers to how data is stored in memory – as integers, real numbers, strings, and so on.

Although VBA can take care of data typing automatically, it does so at a cost – slower execution and less efficient use of memory. If you want optimal speed for your functions, you need to be familiar with data types. Generally, it's best to use the data type that uses the smallest number of bytes, yet still be able to handle all of the data that will be assigned to it. When VBA works with data, execution speed is a function of the number of bytes that VBA has at its disposal. In other words, the fewer bytes used by data, the faster VBA can access and manipulate the data. Table 24-1 lists VBA's assortment of built-in data types.

TABLE 24-1 VBA'S DATA TYPES

Data Type	Bytes Used	Range of Values
Byte	1 byte	0 to 255
Boolean	2 bytes	TRUE or FALSE
Integer	2 bytes	-32,768 to 32,767
Long	4 bytes	-2,147,483,648 to 2,147,483,647
Single	4 bytes	-3.402823E38 to -1.401298E-45 (for negative values); 1.401298E-45 to 3.402823E38 (for positive values)
Double	8 bytes	-1.79769313486231E308 to -4.94065645841247E-324 (negative values); 4.94065645841247E-324 to 1.79769313486232E308 (positive values)
Currency	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Date	8 bytes	January 1, 0100 to December 31, 9999
Object	4 bytes	Any object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a double data type
Variant (with characters)	22 bytes + string length	0 to approximately 2 billion

Declaring Variables

Before you use a variable in a procedure, you may want to *declare* it. Declaring a variable tells VBA its name and data type. Declaring variables provides two main benefits:

- ◆ *Your procedures run faster and use memory more efficiently.* The default data type – variant – causes VBA to repeatedly perform time-consuming checks and reserve more memory than necessary. If VBA knows the data type for a variable, it does not have to investigate; it can reserve just enough memory to store the data.
- ◆ *If you use an Option Explicit statement, you avoid problems involving misspelled variable names.* Suppose that you use an undeclared variable named CurrentRate. At some point in your procedure, however, you insert the statement CurentRate = .075. This misspelled variable name, which is very difficult to spot, will likely cause your function to return an incorrect result. See the sidebar, “Forcing Yourself to Declare all Variables.”

You declare a variable by using the Dim keyword. For example, the following statement declares a variable named Count to be an integer.

```
Dim Count As Integer
```

You also can declare several variables with a single Dim statement. For example,

```
Dim x As Integer, y As Integer, z As Integer
Dim First As Long, Last As Double
```



Unlike some languages, VBA does not permit you to declare a group of variables to be a particular data type by separating the variables with commas. For example, the following statement — although valid — does *not* declare all the variables as integers:

```
Dim i, j, k As Integer
```

In the preceding statement, only *k* is declared to be an integer. To declare all variables as integers, use this statement:

```
Dim i As Integer, j As Integer, k As Integer
```

If you don't declare the data type for a variable that you use, VBA uses the default data type – variant. Data stored as a variant acts like a chameleon: It changes type depending on what you do with it. The following procedure demonstrates how a variable can assume different data types.

```
Function VARIANT_DEMO()
    MyVar = "123"
    MyVar = MyVar / 2
    MyVar = "Answer: " & MyVar
    VARIANT_DEMO = MyVar
End Function
```

Forcing Yourself to Declare All Variables

To force yourself to declare all the variables that you use, include the following as the first instruction in your VBA module:

```
Option Explicit
```

This statement causes your procedure to stop whenever VBA encounters an undeclared variable name. VBA issues an error message (*Compile error: Variable not defined*), and you must declare the variable before you can proceed.

To ensure that the Option Explicit statement appears in every new VBA module, enable the Require Variable Declaration option on the Editor tab of the VB Editor's Options dialog box.

In the VARIANT_DEMO Function procedure, MyVar starts out as a three-character text string that looks like a number. Then this "string" is divided by two and MyVar becomes a numeric data type. Next, MyVar is appended to a string, converting MyVar back to a string. The function returns the final string: *Answer: 61.5*.

Using Constants

A variable's value may – and often does – change while a procedure is executing (that's why it's called a *variable*). Sometimes, you need to refer to a named value or string that never changes; in other words, a *constant*.

You declare a constant by using the Const statement. Here are some examples:

```
Const NumQuarters as Integer = 4  
Const Rate = .0725, Period = 12  
Const CompanyName as String = "Acme Snapholytes"
```

The second statement declares two constants with a single statement, but it does not declare a data type. Consequently, the two constants are variants. Because a constant never changes its value, you normally want to declare your constants as a specific data type. The *scope* of a constant depends on where it is declared within your module:

- ◆ To make a constant available within a single procedure only, declare it after the Sub or Function statement to make it a local constant.
- ◆ To make a constant available to all procedures in a module, declare it before the first procedure in the module.
- ◆ To make a constant available to all modules in the workbook, use the Public keyword and declare the constant before the first procedure in a

module. The following statement creates a constant that is valid in all VBA modules in the workbook.

```
Public AppName As String = "Budget Tools"
```



If you attempt to change the value of a constant in a VBA procedure, you get an error — as you would expect. A constant is a constant, not a variable.

Using constants throughout your code in place of hard-coded values or strings is an excellent programming practice. For example, if your procedure needs to refer to a specific value (such as an interest rate) several times, it's better to declare the value as a constant and use the constant's name rather than its value in your expressions. This technique makes your code more readable and makes it easier to change should the need arise — you have to change only one instruction rather than several.

Using Strings

Like Excel, VBA can manipulate both numbers and text (strings). VBA supports two types of strings:

- ◆ *Fixed-length strings* are declared with a specified number of characters. The maximum length is 65,535 characters.
- ◆ *Variable-length strings* theoretically can hold up to 2 billion characters.

Each character in a string takes 1 byte of storage. When you declare a string variable with a Dim statement, you can specify the maximum length if you know it (that is, a fixed-length string), or you can let VBA handle it dynamically (a variable-length string). In some cases, working with fixed-length strings may be slightly more efficient in terms of memory usage.

In the following example, the MyString variable is declared to be a string with a fixed length of 50 characters. YourString is also declared as a string, but with an unspecified length.

```
Dim MyString As String * 50  
Dim YourString As String
```

Using Dates

You can use a string variable to store a date, of course, but then you can't perform date calculations using the variable. Using the Date data type is a better way to work with dates.

A variable defined as a Date uses 8 bytes of storage and can hold dates ranging from January 1, 0100, to December 31, 9999. That's a span of nearly 10,000 years—more than enough for even the most aggressive financial forecast! The Date data type is also useful for storing time-related data. In VBA, you specify dates and times by enclosing them between two pound signs (#).



The range of dates that VBA can handle is much larger than Excel's own date range, which begins with January 1, 1900. Therefore, be careful that you don't attempt to use a date in a worksheet that lies outside of Excel's acceptable date range.

Here are some examples of declaring variables and constants as Date data types:

```
Dim Today As Date
Dim StartTime As Date
Const FirstDay As Date = #1/1/2002#
Const Noon = #12:00:00#
```



Date variables display dates according to your system's short date format, and times appear according to your system's time format (either 12 or 24 hours). You can modify these system settings by using the Regional Settings option in the Windows Control Panel.

Using Assignment Expressions

An *assignment expression* is a VBA instruction that evaluates an expression and assigns the result to a variable or an object. An *expression* is a combination of keywords, operators, variables, and constants that yields a string, number, or object. An expression can perform a calculation, manipulate characters, or test data.

If you know how to create formulas in Excel, you'll have no trouble creating expressions in VBA. With a worksheet formula, Excel displays the result in a cell. Similarly, you can assign a VBA expression to a variable or use it as a property value.

VBA uses the equal sign (=) as its assignment operator. Note the following examples of assignment statements (the expressions are to the right of the equal sign):

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
MultiSheets = True
```

Expressions often use functions. These can be VBA's built-in functions, Excel's worksheet functions, or custom functions that you develop in VBA. I discuss VBA's built-in functions later in this chapter.

Operators play a major role in VBA. Familiar operators describe mathematical operations, including addition (+), multiplication (*), division (/), subtraction (-), exponentiation (^), and string concatenation (&). Less familiar operators are the backslash (\) that's used in integer division, and the Mod operator that's used in modulo arithmetic. The Mod operator returns the remainder of one integer divided by another. For example, the following expression returns 2:

```
17 Mod 3
```

You may be familiar with Excel's MOD function. Note that, in VBA, Mod is an operator, not a function.

VBA also supports the same comparative operators used in Excel formulas: Equal to (=), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), and not equal to (<>). Additionally, VBA provides a full set of logical operators, as shown in Table 24-2. Refer to the online help for additional information and examples of these operators.

TABLE 24-2 VBA'S LOGICAL OPERATORS

Operator	What It Does
Not	Performs a logical negation on an expression.
And	Performs a logical conjunction on two expressions
Or	Performs a logical disjunction on two expressions
Xor	Performs a logical exclusion on two expressions
Eqv	Performs a logical equivalence on two expressions
Imp	Performs a logical implication on two expressions

The order of precedence for operators in VBA exactly matches that in Excel. Of course, you can add parentheses to change the natural order of precedence.

Using Arrays

An *array* is a group of elements of the same type that have a common name; you refer to a specific element in the array by using the array name and an index number. For example, you may define an array of 12 string variables so that each variable corresponds to the name of a different month. If you name the array

MonthNames, you can refer to the first element of the array as MonthNames(0), the second element as MonthNames(1), and so on, up to MonthNames(11).

Declaring an Array

You declare an array with a Dim or Public statement just as you declare a regular variable. You also can specify the number of elements in the array. You do so by specifying the first index number, the keyword To, and the last index number – all inside parentheses. For example, here's how to declare an array comprised of exactly 100 integers:

```
Dim MyArray(1 To 100) As Integer
```

When you declare an array, you need to specify only the upper index, in which case VBA (by default) assumes that 0 is the lower index. Therefore, the following two statements have the same effect:

```
Dim MyArray(0 to 100) As Integer  
Dim MyArray(100) As Integer
```

In both cases, the array consists of 101 elements.

If you want VBA to assume that 1 is the lower index for all arrays that declare only the upper index, include the following statement before any procedures in your module:

```
Option Base 1
```

If this statement is present, the following two statements have the same effect (both declare an array with 100 elements):

```
Dim MyArray(1 to 100) As Integer  
Dim MyArray(100) As Integer
```

Declaring Multidimensional Arrays

The array examples in the preceding section are one-dimensional arrays. VBA arrays can have up to 60 dimensions, although it's rare to need more than 3 dimensions (a 3-D array). The following statement declares a 100-integer array with two dimensions (2-D):

```
Dim MyArray(1 To 10, 1 To 10) As Integer
```

You can think of the preceding array as occupying a 10×10 matrix. To refer to a specific element in a 2-D array, you need to specify two index numbers. For example, here's how you can assign a value to an element in the preceding array:

```
MyArray(3, 4) = 125
```

A *dynamic array* does not have a preset number of elements. You declare a dynamic array with a blank set of parentheses:

```
Dim MyArray() As Integer
```

Before you can use a dynamic array in your code, however, you must use the ReDim statement to tell VBA how many elements are in the array (or ReDim Preserve if you want to keep the existing values in the array). You can use the ReDim statement any number of times, changing the array's size as often as you like.

Arrays crop up later in this chapter in the sections that discuss looping.

Using VBA's Built-in Functions

VBA has a variety of built-in functions that simplify calculations and operations. Many of VBA's functions are similar (or identical) to Excel's worksheet functions. For example, the VBA function UCase, which converts a string argument to uppercase, is equivalent to the Excel worksheet function UPPER.



To display a list of VBA functions while writing your code, type **VBA** followed by a period (.). The VB Editor displays a list of all functions (see Figure 24-1). If this does not work for you, make sure that you select the Auto List Members option. Choose Tools → Options, and click the Editor tab. In addition to functions, the displayed list also includes built-in constants. The VBA functions are all described in the online help. To view help, just move the cursor over a function name and press F1.

Here's a statement that calculates the square root of a variable by using VBA's Sqr function, and then assigns the result to a variable named *x*.

```
x = Sqr(MyValue)
```

Having knowledge of VBA's functions can save you lots of work. For example, consider the REMOVESPACES Function procedure presented at the beginning of this chapter. That function uses a For-Next loop to examine each character in a string, and builds a new string. A much simpler (and more efficient) version of that Function procedure uses VBA's Replace function. The following is a rewritten version of the Function procedure.

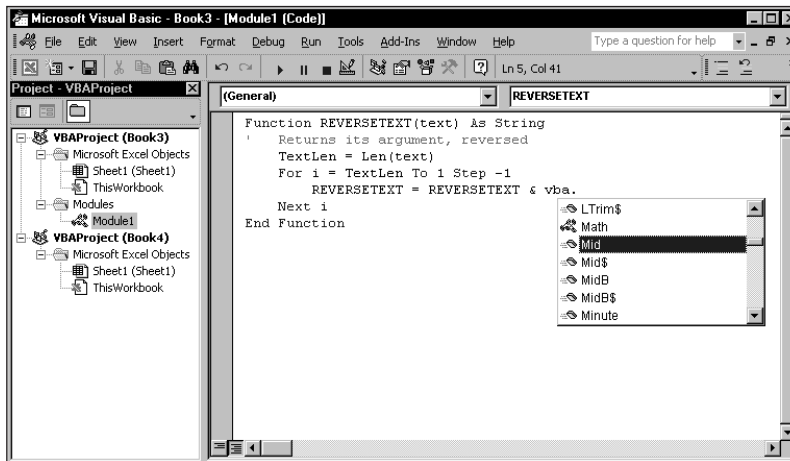


Figure 24-1: Displaying a list of VBA functions in the VB Editor

```
Function REMOVESPACES(cell) As String
    ' Removes all spaces from cell
    REMOVESPACES = Replace(cell, " ", "")
End Function
```



The Replace function was introduced in the version of VBA included with Excel 2000. This function is not available if you use an earlier version of Excel.

You can use many (but not all) of Excel's worksheet functions in your VBA code. To use a worksheet function in a VBA statement, just precede the function name with WorksheetFunction and a dot.



For compatibility with earlier versions of Excel, use Application rather than WorksheetFunction. The WorksheetFunction object was introduced in Excel 97, and it won't work with Excel 95. The following statements are equivalent:

```
Result = Application.Max(x, y, z)
Result = WorksheetFunction.Max(x, y, z)
```

The following code demonstrates how to use an Excel worksheet function in a VBA statement. Excel's infrequently used ROMAN function converts a decimal number into a Roman numeral.

```
DecValue = 1999
RValue = WorksheetFunction.Roman(DecValue)
```

The variable RomanValue contains the string MCMXCIX. Fans of old movies are often dismayed when they learn that Excel does not have a function to convert a Roman numeral to its decimal equivalent. You can, of course, create such a function. Are you up for a challenge?

It's important to understand that you can't use worksheet functions that have an equivalent VBA function. For example, VBA can't access Excel's SQRT worksheet function because VBA has its own version of that function: Sqr. Therefore, the following statement generates an error:

```
x = Application.SQRT(123) 'error
```

Controlling Execution

Some VBA procedures start at the top and progress line by line to the bottom. Often, however, you need to control the flow of your routines by skipping over some statements, executing some statements multiple times, and testing conditions to determine what the routine does next.

This section discusses several ways of controlling the execution of your VBA procedures:

- ◆ If-Then constructs
- ◆ Select Case constructs
- ◆ For-Next loops
- ◆ Do While loops
- ◆ Do Until loops
- ◆ On Error statements

The If-Then Construct

Perhaps the most commonly used instruction grouping in VBA is the If-Then construct. This instruction is one way to endow your applications with decision-making capability. The basic syntax of the If-Then construct is:

```
If condition Then true_instructions [Else false_instructions]
```

The If-Then construct executes one or more statements conditionally. The Else clause is optional. If included, it enables you to execute one or more instructions when the condition that you test is not true.

The following Function procedure demonstrates an If-Then structure without an Else clause. The example deals with time. VBA uses the same date-and-time serial number system as Excel. The time of day is expressed as a fractional value—for example, noon is represented as .5. VBA's Time function returns a value that represents the time of day, as reported by the system clock. In the following example, the function starts out by assigning an empty string to GreetMe. The If-Then statement checks the time of day. If the time is before noon, the Then part of the statement executes and the function returns *Good Morning*.

```
Function GreetMe()  
    GreetMe = ""  
    If Time < 0.5 Then GreetMe= "Good Morning"  
End Function
```

The following function uses two If-Then statements. It displays either *Good Morning* or *Good Afternoon*:

```
Function GreetMe()  
    If Time < 0.5 Then GreetMe = "Good Morning"  
    If Time >= 0.5 Then GreetMe = "Good Afternoon"  
End Function
```

Notice that the second If-Then statement uses >= (greater than or equal to). This covers the extremely remote chance that the time is precisely 12:00 noon when the function is executed.

Another approach is to use the Else clause of the If-Then construct. For example,

```
Function GreetMe()  
    If Time < 0.5 Then GreetMe = "Good Morning" Else _  
        GreetMe = "Good Afternoon"  
End Function
```

Notice that the preceding example uses the line continuation sequence (a space followed by an underscore); If-Then-Else is actually a single statement.

The following is another example that uses the If-Then construct. This Function procedure calculates a discount based on a quantity (assumed to be an integer value). It accepts one argument (quantity) and returns the appropriate discount based on that value.

```
Function Discount(quantity)  
    If quantity <= 5 Then Discount = 0  
    If quantity >= 6 Then Discount = 0.1
```

```
If quantity >= 25 Then Discount = 0.15
If quantity >= 50 Then Discount = 0.2
If quantity >= 75 Then Discount = 0.25
End Function
```

Notice that each If-Then statement in this procedure is always executed, and the value for Discount can change as the function is executed. The final value, however, is the desired value.

The preceding examples all used a single statement for the Then clause of the If-Then construct. However, you often need to execute multiple statements if a condition is TRUE. You can still use the If-Then construct, but you need to use an End If statement to signal the end of the statements that comprise the Then clause. Here's an example that executes two statements if the If clause is TRUE.

```
If x > 0 Then
    y = 2
    z = 3
End If
```

You can also use multiple statements for an If-Then-Else construct. Here's an example that executes two statements if the If clause is TRUE, and two other statements if the If clause is not TRUE.

```
If x > 0 Then
    y = 2
    z = 3
Else
    y = -2
    z = -3
End If
```

The Select Case Construct

The Select Case construct is useful for choosing among three or more options. This construct also works with two options and is a good alternative to If-Then-Else. The syntax for Select Case is as follows:

```
Select Case testexpression
    [Case expressionlist-n
        [instructions-n]]
    [Case Else
        [default_instructions]]
End Select
```


The following example of a Select Case construct shows another way to code the GreetMe examples presented in the preceding section:

```
Function GreetMe()  
    Select Case Time  
        Case Is < 0.5  
            GreetMe = "Good Morning"  
        Case 0.5 To 0.75  
            GreetMe = "Good Afternoon"  
        Case Else  
            GreetMe = "Good Evening"  
    End Select  
End Function
```

And here's a rewritten version of the Discount function from the previous section, this time using a Select Case construct:

```
Function Discount(quantity)  
    Select Case quantity  
        Case Is <= 5  
            Discount = 0  
        Case 6 To 24  
            Discount = 0.1  
        Case 25 To 49  
            Discount = 0.15  
        Case 50 To 74  
            Discount = 0.2  
        Case Is >= 75  
            Discount = 0.25  
    End Select  
End Function
```

Any number of instructions can be written below each Case statement; they all execute if that case evaluates to TRUE.

Looping Blocks of Instructions

Looping is the process of repeating a block of VBA instructions within a procedure. You may know the number of times to loop, or it may be determined by the values of variables in your program. VBA offers a number of looping constructs:

- ◆ For-Next Loops
- ◆ Do While Loops
- ◆ Do Until Loops

FOR-NEXT LOOPS

The following is the syntax for a For-Next loop:

```
For counter = start To end [Step stepval]  
    [instructions]  
    [Exit For]  
    [instructions]  
Next [counter]
```

The following listing is an example of a For-Next loop that does not use the optional Step value or the optional Exit For statement. This function accepts two arguments and returns the sum of all integers between (and including) the arguments.

```
Function SumIntegers(first, last)  
    total = 0  
    For num = first To last  
        total = total + num  
    Next num  
    SumIntegers = total  
End Function
```

The following formula, for example, returns 55 – the sum of all integers from 1 to 10.

```
=SumIntegers(1,10)
```

In this example, num (the loop counter variable) starts out with the same value as the first variable, and increases by 1 each time the loop repeats. The loop ends when num is equal to the last variable. The total variable simply accumulates the various values of num as it changes during the looping.



When you use For-Next loops, you should understand that the loop counter is a normal variable — it is not a special type of variable. As a result, you can change the value of the loop counter within the block of code executed between the For and Next statements. This is, however, a *very bad* practice and can cause problems. In fact, you should take special precautions to ensure that your code does not change the loop counter.

You also can use a Step value to skip some values in the loop. Here's the same function rewritten to sum *every other* integer between the first and last arguments.

```
Function SumIntegers2(first, last)
```

```

total = 0
For num = first To last Step 2
    total = total + num
Next num
SumIntegers2 = Total
End Function

```

The following formula returns 25, which is the sum of 1, 3, 5, 7, and 9.

```
=SumIntegers2(1,10)
```

For-Next loops can also include one or more Exit For statements within the loop. When this statement is encountered, the loop terminates immediately, as the following example demonstrates.

```

Function RowOfLargest(c)
    NumRows = Rows.Count
    MaxVal = WorksheetFunction.Max(Columns(c))
    For r = 1 To NumRows
        If Cells(r, c) = MaxVal Then
            RowOfLargest = r
            Exit For
        End If
    Next r
End Function

```

The RowOfLargest function accepts a column number (1 through 256) for its argument, and returns the row number of the largest value in that column. It starts by getting a count of the number of rows in the worksheet (this varies, depending on the version of Excel). This number is assigned to the *NumRows* variable. The maximum value in the column is calculated by using Excel's MAX function, and this value is assigned to the MaxVal variable.

The For-Next loop checks each cell in the column. When the cell equal to MaxVal is found, the row number (variable *r*, the loop counter) is assigned to the function's name and the Exit For statement ends the procedure. Without the Exit For statement, the loop continues to check all cells in the column – which can take quite a long time!

The previous examples use relatively simple loops. But you can have any number of statements in the loop, and you can even nest For-Next loops inside other For-Next loops. The following is VBA code that uses nested For-Next loops to initialize a 10 × 10 × 10 array with the value -1. When the three loops finish executing, each of the 1,000 elements in MyArray contains -1.

```

Dim MyArray(1 to 10, 1 to 10, 1 to 10)
For i = 1 To 10

```

```
For j = 1 To 10
    For k = 1 To 10
        MyArray(i, j, k) = -1
    Next k
Next j
Next i
```

DO WHILE LOOPS

A Do While loop is another type of looping structure available in VBA. Unlike a For-Next loop, a Do While loop executes while a specified condition is met. A Do While loop can have one of two syntaxes:

```
Do [While condition]
    [instructions]
    [Exit Do]
    [instructions]
Loop
```

or

```
Do
    [instructions]
    [Exit Do]
    [instructions]
Loop [While condition]
```

As you can see, VBA enables you to put the While condition at the beginning or the end of the loop. The difference between these two syntaxes involves the point in time when the condition is evaluated. In the first syntax, the contents of the loop may never be executed. In the second syntax, the contents of the loop are always executed at least one time.

The following example is the RowOfLargest function presented in the previous section, rewritten to use a Do While loop (using the first syntax).

```
Function RowOfLargest(c)
    NumRows = Rows.Count
    MaxVal = Application.Max(Columns(c))
    r = 1
    Do While Cells(r, c) <> MaxVal
        r = r + 1
    Loop
    RowOfLargest = r
End Function
```

The variable *r* starts out with a value of 1, and increments within the Do While loop. The looping continues as long as the cell being evaluated is not equal to *MaxVal*. When the cell is equal to *MaxVal*, the loop ends and the function is assigned the value of *r*. Notice that if the maximum value is in row 1, the looping does not occur.

The following procedure uses the second Do While loop syntax. The loop always executes at least once.

```
Function RowOfLargest(c)
    NumRows = Rows.Count
    MaxVal = Application.Max(Columns(c))
    r = 0
    Do
        r = r + 1
    Loop While Cells(r, c) <> MaxVal
    RowOfLargest = r
End Function
```

Do While loops can also contain one or more Exit Do statements. When an Exit Do statement is encountered, the loop ends immediately.

DO UNTIL LOOPS

The Do Until loop structure closely resembles the Do While structure. The difference is evident only when the condition is tested. In a Do While loop, the loop executes *while* the condition is true. In a Do Until loop, the loop executes *until* the condition is true. Do Until also has two syntaxes:

```
Do [Until condition]
    [instructions]
    [Exit Do]
    [instructions]
Loop
```

or

```
Do
    [instructions]
    [Exit Do]
    [instructions]
Loop [Until condition]
```

The following example demonstrates the first syntax of the Do Until loop. This example makes the code a bit clearer because it avoids the negative comparison required in the Do While example.

```
Function RowOfLargest(c)
    NumRows = Rows.Count
    MaxVal = Application.Max(Columns(c))
    r = 1
    Do Until Cells(r, c) = MaxVal
        r = r + 1
    Loop
    RowOfLargest = r
End Function
```

Finally, the following function is the same procedure, but is rewritten to use the second syntax of the Do Until loop.

```
Function RowOfLargest(c)
    NumRows = Rows.Count
    MaxVal = Application.Max(Columns(c))
    r = 0
    Do
        r = r + 1
    Loop Until Cells(r, c) = MaxVal
    RowOfLargest = r
End Function
```

The On Error Statement

Undoubtedly, you've used one of Excel's worksheet functions in a formula and discovered that the formula returns an error value (for example, #VALUE!). A formula can return an error value in a number of situations, including:

- ◆ You omitted one or more required argument(s).
- ◆ An argument was not the correct data type (for example, text instead of a value).
- ◆ An argument is outside of a valid numeric range (division by zero, for example).

In many cases, you can ignore error handling within your functions. If the user does not provide the proper number of arguments, the function simply returns an error value. It's up to the user to figure out the problem. In fact, this is how Excel's worksheet functions handle errors.

In other cases, you want your code to know if errors occurred and then do something about them. Excel's On Error statement enables you to identify and handle errors.

To simply ignore an error, use the following statement:

```
On Error Resume Next
```

If you use this statement, you can determine whether an error occurs by checking the `Number` property of the `Err` object. If this property is equal to zero, an error did not occur. If `Err.Number` is equal to anything else, an error *did* occur.

The following example is a function that returns the name of a cell or range. If the cell or range does not have a name, an error occurs and the formula that uses the function returns a `#VALUE!` error.

```
Function RANGENAME(rng)
    RANGENAME = rng.Name.Name
End Function
```

The following list shows an improved version of the function. The `On Error Resume Next` statement causes VBA to ignore the error. The `If Err` statement checks to see if an error occurs. If so, the function returns an empty string.

```
Function RANGENAME(rng)
    On Error Resume Next
    RANGENAME = rng.Name.Name
    If Err.Number <> 0 Then RANGENAME = ""
End Function
```

The following statement instructs VBA to watch for errors, and if an error occurs, continues executing at a different named location – in this case, a statement labeled `ErrorHandler`.

```
On Error GoTo ErrorHandler
```

The following Function procedure demonstrates this statement. The `DIVIDETWO` function accepts two arguments (*num1* and *num2*) and returns the result of *num1* divided by *num2*.

```
Function DIVIDETWO(num1, num2)
    On Error GoTo ErrorHandler
    DIVIDETWO = num1 / num2
    Exit Function
ErrorHandler:
    DIVIDETWO = "ERROR"
End Function
```

The `On Error GoTo` statement instructs VBA to jump to the statement labeled `ErrorHandler` if an error occurs. As a result, the function returns a string (`ERROR`) if

any type of error occurs while the function is executing. Note the use of the Exit Function statement. Without this statement, the code continues executing and the error handling code *always* executes. In other words, the function always returns *ERROR*.

It's important to understand that the DIVIDETWO function is *non-standard* in its approach. Returning a string when an error occurs (*ERROR*) is not how Excel's functions work. Rather, they return an actual error value.



Chapter 25 contains several examples of the On Error statement, including an example that demonstrates how to return an actual error value from a function.

Using Ranges

Many of the custom functions that you develop will work with the data contained in a cell or in a range of cells. Recognize that a range can be a single cell or a group of cells. This section describes some key concepts to make this task easier. The information in this section is intended to be practical, rather than comprehensive. If you want more details, consult the online help.



Chapter 25 contains many practical examples of functions that use ranges. Studying these examples helps to clarify the information in this section.

The For Each-Next Construct

Your Function procedures often need to loop through a range of cells. For example, you may write a function that accepts a range as an argument. Your code needs to examine each cell in the range and do something. The For Each-Next construct is very useful for this sort of thing. The syntax of the For Each-Next construct is:

```
For Each element In group
    [instructions]
    [Exit For]
    [instructions]
Next [element]
```

The following Function procedure accepts a range argument, and returns the sum of the squared values in the range.


```
Function SUMOFSQUARES(rng as Range)
    Dim total as Double
    Dim cell as Range
    total = 0
    For Each cell In rng
        total = total + cell ^ 2
    Next cell
    SUMOFSQUARES = total
End Function
```

The following is a worksheet formula that uses the SumOfSquares function.

```
=SumOfSquares(A1:C100)
```

In this case, the function's argument is a range that consists of 300 cells.



In the preceding example, *cell* and *rng* are both variable names. There's nothing special about either name; you can replace them with any valid variable name.

Referencing a Range

VBA code can reference a range in a number of different ways:

- ◆ The Range property
- ◆ The Cells property
- ◆ The Offset property

THE RANGE PROPERTY

You can use the Range property to refer to a range directly, by using a cell address or name. The following example assigns the value in cell A1 to a variable named *Init*. In this case, the statement accesses the range's Value property.

```
Init = Range("A1").Value
```

In addition to the Value property, VBA enables you to access a number of other properties of a range. For example, the following statement counts the number of cells in a range and assigns the value to the *Cnt* variable.

```
Cnt = Range("A1:C300").Count
```

The Range property is also useful for referencing a single cell in a multicell range. For example, you may create a function that is supposed to accept a single-cell argument. If the user specifies a multicell range as the argument, you can use the Range property to extract the upper left cell in the range. The following example uses the Range property (with an argument of "A1") to return the value in the upper left cell of the range represented by the cell argument.

```
Function Square(cell as Range)
    Dim CellValue as Double
    CellValue = cell.Range("A1").Value
    Square = CellValue ^ 2
End Function
```

Assume that the user enters the following formula:

```
=Square(C5:C12)
```

The Square function works with the upper left cell in C5:C12 (which is C5), and returns the value squared.



Many of Excel's worksheet functions work in this way. For example, if you specify a multicell range as the first argument for the LEFT function, Excel uses the upper left cell in the range. However, Excel is not consistent. If you specify a multicell range as the argument for the SQRT function, Excel returns an error.

THE CELLS PROPERTY

Another way to reference a range is to use the Cells property. The Cells property accepts two arguments (a row number and a column number), and returns a single cell. The following statement assigns the value in cell A1 to a variable named FirstCell:

```
FirstCell = Cells(1, 1).Value
```

The following statement returns the upper left cell in the range C5:C12.

```
UpperLeft = Range("C5:C12").Cells(1,1)
```



If you use the Cells property without an argument, it returns a range that consists of all cells on the worksheet. In the following example, the *TotalCells* variable contains the total number of cells in the worksheet.

```
TotalCells = Cells.Count
```

The following statement uses Excel's COUNTA function to determine the number of non-empty cells in the worksheet:

```
NonEmpty =WorksheetFunction.COUNTA(Cells)
```

THE OFFSET PROPERTY

The Offset property (like the Range and Cells properties) also returns a Range object. The Offset property is used in conjunction with a range. It takes two arguments that correspond to the relative position from the upper left cell of the specified Range object. The arguments can be positive (down or right), negative (up or left), or zero. The following example returns the value one cell below cell A1 (i.e., cell A2), and assigns it to a variable named NextCell:

```
NextCell = Range("A1").Offset(1,0).Value
```

The following Function procedure accepts a single-cell argument, and uses a For-Next loop to return the sum of the 10 cells below it:

```
Function SumBelow(cell as Range)
    Dim Total as Double
    Dim i as Integer
    Total = 0
    For i = 1 To 10
        Total = Total + cell.Offset(i, 0)
    Next i
    SumBelow = Total
End Function
```

Some Useful Properties of Ranges

Previous sections gave examples that used the Value property for a range. VBA gives you access to many additional range properties. Some of the more useful properties for function writers are briefly described in the following sections. For complete information on a particular property, refer to Excel's online help.

THE FORMULA PROPERTY

The Formula property returns the formula (as a string) contained in a cell. If you try to access the Formula property for a range that consists of more than one cell, you get an error. If the cell does not have a formula, this property returns a string, which is the cell's value as it appears in the formula bar. The following function simply displays the formula for the upper left cell in a range:

```
Function CELLFORMULA(cell)
    CELLFORMULA = cell.Range("A1").Formula
End Function
```

You can use the `HasFormula` property to determine whether a cell has a formula.

THE ADDRESS PROPERTY

The `Address` property returns the address of a range as a string. By default, it returns the address as an absolute reference (for example, `A1:C12`). The following function, which is not all that useful, returns the address of a range.

```
Function RANGEADDRESS(rng)
    RANGEADDRESS = rng.Address
End Function
```

For example, the following formula returns the string `A1:C3`:

```
=RANGEADDRESS(A1:C3)
```

THE COUNT PROPERTY

The `Count` property returns the number of cells in a range. The following function uses the `Count` property:

```
Function CELLCOUNT(rng)
    CELLCOUNT = rng.Count
End Function
```

The following formula returns 9:

```
=CELLCOUNT(A1:C3)
```

THE PARENT PROPERTY

The `Parent` property returns an object that corresponds to an object's *container* object. For a `Range` object, the `Parent` property returns a `Worksheet` object (the worksheet that contains the range).

The following function uses the `Parent` property and returns the name of the worksheet of the range passed as an argument:

```
Function SHEETNAME(rng)
    SHEETNAME = rng.Parent.Name
End Function
```

The following formula, for example, returns the string `Sheet1`:

```
=SHEETNAME(Sheet1!A16)
```

THE NAME PROPERTY

The Name property returns a Name object for a cell or range. To get the actual cell or range name, you need to access the Name property of the Name object. If the cell or range does not have a name, the Name property returns an error.

The following Function procedure displays the name of a range or cell passed as its argument. If the range or cell does not have a name, the function returns an empty string. Note the use of On Error Resume Next. This handles situations in which the range does not have a name.

```
Function RANGENAME(rng)
    On Error Resume Next
    RANGENAME = rng.Name.Name
    If Err.Number <> 0 Then RANGENAME = ""
End Function
```

THE NUMBERFORMAT PROPERTY

The NumberFormat property returns the number format (as a string) assigned to a cell or range. The following function simply displays the number format for the upper left cell in a range:

```
Function NUMBERFORMAT(cell)
    NUMBERFORMAT = cell.Range("A1").NumberFormat
End Function
```

THE FONT PROPERTY

The Font property returns a Font object for a range or cell. To actually do anything with this Font object, you need to access its properties. For example, a Font object has properties such as Bold, Italic, Name, Color, and so on. The following function returns TRUE if the upper left cell of its argument is formatted as bold:

```
Function ISBOLD(cell)
    ISBOLD = cell.Range("A1").Font.Bold
End Function
```

THE COLUMNS AND ROWS PROPERTIES

The Columns and Rows properties work with columns or rows in a range. For example, the following function returns the number of columns in a range by accessing the Count property:

```
Function COLUMNCOUNT(rng)
    COLUMNCOUNT = rng.Columns.Count
End Function
```

THE ENTIREROW AND ENTIRECOLUMN PROPERTIES

The EntireRow and EntireColumn properties enable you to work with an entire row or column for a particular cell. The following function accepts a single cell argument and then uses the EntireColumn property to get a range consisting of the cell's entire column. It then uses Excel's COUNTA function to return the number of non-empty cells in the column.

```
Function NONEMPTY(cell)
    NONEMPTY = WorksheetFunction.CountA(cell.EntireColumn)
End Function
```

THE HIDDEN PROPERTY

The Hidden property is used with rows or columns. It returns TRUE if the row or column is hidden. If you try to access this property for a range that does not consist of an entire row or column, you get an error. The following function accepts a single cell argument, and returns TRUE if either the cell's row or the cell's column is hidden:

```
Function CELLISHIDDEN(cell)
    If cell.EntireRow.Hidden Or cell.EntireColumn.Hidden Then
        CELLISHIDDEN = True
    Else
        CELLISHIDDEN = False
    End If
End Function
```

You can also write this function without using an If-Then-Else construct. In the following function, the expression to the right of the equal sign returns either TRUE or FALSE – and this value is assigned returned by the function.

```
Function CELLISHIDDEN(cell)
    CELLISHIDDEN = cell.EntireRow.Hidden Or _
        cell.EntireColumn.Hidden
End Function
```

The Set Keyword

An important concept in VBA is the ability to create a new Range object and assign it to a variable – more specifically, an *object variable*. You do so by using the Set keyword. The following statement creates an object variable named MyRange:

```
Set MyRange = Range("A1:A10")
```

After the statement executes, you can use the `MyRange` variable in your code in place of the actual range reference. Examples in subsequent sections help to clarify this concept.



Creating a Range object is not the same as creating a named range. In other words, you can't use the name of a Range object in your formulas.

The Intersect Function

The `Intersect` function returns a range that consists of the intersection of two other ranges. For example, consider the two ranges selected in Figure 24-2. These ranges, `D3:D10` and `B5:F5`, contain one cell in common (`D5`). In other words, `D5` is the intersection of `D3:D10` and `B5:F5`.

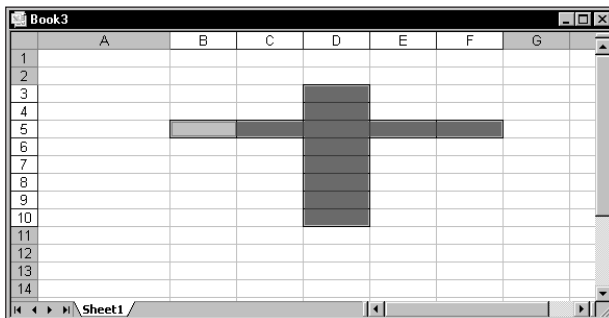


Figure 24-2: Use the `Intersect` function to work with the intersection of two ranges.

The following Function procedure accepts two range arguments, and returns the count of the number of cells that the ranges have in common:

```
Function CELLSINCOMMON(rng1, rng2)
    Dim CommonCells As Range
    On Error Resume Next
    Set CommonCells = Intersect(rng1, rng2)
    If Err.Number = 0 Then
        CELLSINCOMMON = CommonCells.Count
    Else
        CELLSINCOMMON = 0
    End If
End Function
```

The CELLSINCOMMON function uses the Intersect function to create a range object named CommonCells. Note the use of On Error Resume Next. This statement is necessary because the Intersect function returns an error if the ranges have no cells in common. If the error occurs, it is ignored. The final statement checks the Number property of the Err object. If 0, then no error occurs and the function returns the value of the Count property for the CommonCells object. If an error does occur, then Err.Number has a value other than 0 and the function returns 0.

The Union Function

The Union function combines two or more ranges into a single range. The following statement uses the Union function to create a range object that consists of the first and third columns of a worksheet:

```
Set TwoCols = Union(Range("A:A"), Range("C:C"))
```

The Union function can take any number of arguments.

The UsedRange Property

The UsedRange property returns a Range object that represents the used range of the worksheet. Press Ctrl+End to activate the lower right cell of the used range. The UsedRange property can be *very useful* in making your functions more efficient.

Consider the following Function procedure. This function accepts a range argument and returns the number of formula cells in the range.

```
Function FORMULACOUNT(rng As Range)
    Dim cnt As Long
    Dim cell As Range
    cnt = 0
    For Each cell In rng
        If cell.HasFormula Then cnt = cnt + 1
    Next cell
    FORMULACOUNT = cnt
End Function
```

In many cases, the preceding function works just fine. But what if the user enters a formula like this one?

```
=FORMULACOUNT("A:C")
```

With an argument that consists of one or more entire columns, the function does not work well because it loops through every cell in the range, even those that are well beyond the area of the sheet that's actually used. The following function is rewritten to make it more efficient:


```
Function FORMULACOUNT(rng As Range)
    Dim cnt As Long
    Dim cell As Range
    cnt = 0
    Set WorkRange = Intersect(rng, rng.Parent.UsedRange)
    For Each cell In WorkRange
        If cell.HasFormula Then cnt = cnt + 1
    Next cell
    FORMULACOUNT = cnt
End Function
```

This function creates a Range object named `WorkRange` that consists of the intersection of the range passed as an argument and the used range of the worksheet. In other words, `WorkRange` consists of a subset of the range argument that only includes cells in the used range of the worksheet.

Summary

This chapter provided an introduction to VBA's language elements, including variables, data types, constants, and arrays. It also discussed the various methods that you can use to control the flow of execution of your Function procedures. The chapter also presented several examples of functions that demonstrate how to work with ranges and use VBA's built-in functions.

The next and final chapter contains examples of custom functions.

Chapter 25

VBA Custom Function Examples

IN THIS CHAPTER

- ◆ Simple custom function examples
- ◆ A custom function to determine a cell's data type
- ◆ A custom function to make a single worksheet function act like multiple functions
- ◆ A custom function for generating random numbers and selecting cells at random
- ◆ Custom functions for calculating sales commissions
- ◆ Custom functions for manipulating text
- ◆ Custom functions for counting and summing cells
- ◆ Custom functions that deal with dates
- ◆ A custom function example for returning the last nonempty cell in a column or row
- ◆ Custom functions that work with multiple worksheets
- ◆ Advanced custom function techniques

THIS CHAPTER IS JAM-PACKED with a wide variety of useful (or potentially useful) VBA custom functions. You can use many of the functions as they are written. You may need to modify other functions to meet your particular needs. For maximum speed and efficiency, these function procedures declare all variables that are used.

Simple Functions

The functions in this section are relatively simple, but they can be very useful. Most of them are based on the fact that VBA can obtain lots of useful information that's not normally available for use in a formula. For example, your VBA code can access a cell's `HasFormula` property to determine whether a cell contains a formula. Oddly, Excel does not have a built-in worksheet function that tells you this.



The companion CD-ROM contains a workbook that includes all of the functions in this section.

Does a Cell Contain a Formula?

The following `CELLHASFORMULA` function accepts a single-cell argument and returns `TRUE` if the cell has a formula.

```
Function CELLHASFORMULA(cell) As Boolean
' Returns TRUE if cell has a formula
  CELLHASFORMULA = cell.Range("A1").HasFormula
End Function
```

If a multi-cell range argument is passed to the function, the function works with the upper-left cell in the range.

Returning a Cell's Formula

The following `CELLFORMULA` function returns the formula for a cell as a string. If the cell does not have a formula, it returns an empty string.

```
Function CELLFORMULA(cell) As String
' Returns the formula in cell, or an
' empty string if cell has no formula
  Dim UpperLeft As Range
  Set UpperLeft = cell.Range("A1")
  If UpperLeft.HasFormula Then
    CELLFORMULA = UpperLeft.Formula
  Else
    CELLFORMULA = ""
  End If
End Function
```

This function creates a `Range` object variable named `UpperLeft`. This variable represents the upper-left cell in the argument that is passed to the function.

Is the Cell Hidden?

The following `CELLISHIDDEN` function accepts a single cell argument and returns `TRUE` if the cell is hidden. It is considered a hidden cell if either its row or its column is hidden.

Using the Functions in this Chapter

If you see a function listed in this chapter that you find useful, you can use it in your own workbook. All of the Function procedures in this chapter are available on the companion CD-ROM. Just open the appropriate workbook (see Appendix E for a description of the files), activate the VB Editor, and copy and paste the function listing to a VBA module in your workbook. If you prefer, you can collect a number of functions and create an add-in (see Chapter 23 for details).

It's impossible to anticipate every function that you'll ever need. However, the examples in this chapter cover a wide variety of topics, so it's likely that you can locate an appropriate function and adapt the code for your own use.

```
Function CELLISHIDDEN(cell) As Boolean
' Returns TRUE if cell is hidden
  Dim UpperLeft As Range
  Set UpperLeft = cell.Range("A1")
  CELLISHIDDEN = UpperLeft.EntireRow.Hidden Or _
    UpperLeft.EntireColumn.Hidden
End Function
```

Returning a Worksheet Name

The following SHEETNAME function accepts a single argument (a range) and returns the name of the worksheet that contains the range. It uses the Parent property of the Range object. The Parent property returns an object—the object that contains the Range object.

```
Function SHEETNAME(rng) As String
' Returns the sheet name for rng
  SHEETNAME = rng.Parent.Name
End Function
```

The following function is a variation on this theme. It does not use an argument; rather, it relies on the fact that a function can determine the cell from which it was called by using Application.Caller.

```
Function SHEETNAME2() As String
' Returns the sheet name of the cell that
' contains the function
  SHEETNAME2 = Application.Caller.Parent.Name
End Function
```

Understanding Object Parents

Objects in Excel are arranged in a hierarchy. At the top of the hierarchy is the Application object (Excel itself). Excel contains other objects, these objects contain other objects, and so on. The following hierarchy depicts how a Range object fits into this scheme.

Application Object (Excel)

Workbook Object

Worksheet Object

Range Object

In the lingo of object-oriented programming, a Range object's parent is the Worksheet object that contains it. A Worksheet object's parent is the workbook that contains the worksheet. And, a Workbook object's parent is the Application object. Armed with this knowledge, you can make use of the Parent property to create a few useful functions.

In this function, Application.Caller returns a Range object that corresponds to the cell that contains the function. For example, suppose that you have the following formula in cell A1:

```
=SHEETNAME()
```

When the SHEETNAME function is executed, Application.Caller returns a Range object corresponding to the cell that contains the function. The Parent property returns the Worksheet object; and the Name property returns the name of the worksheet.

Returning a Workbook Name

The next function, WORKBOOKNAME, returns the name of the workbook. Notice that it uses the Parent property twice. The first Parent property returns a Worksheet object; the second Parent property returns a Workbook object, and the Name property returns the name of the workbook..

```
Function WORKBOOKNAME() As String
' Returns the workbook name of the cell
' that contains the function
    WORKBOOKNAME = Application.Caller.Parent.Parent.Name
End Function
```

Returning the Application's Name

The following function, although not very useful, carries this discussion of object parents to the next logical level by accessing the Parent property three times. This function returns the name of the Application object, which is always the string *Microsoft Excel*.

```
Function APPNAME() As String
' Returns the application name of the cell
' that contains the function
  APPNAME = Application.Caller.Parent.Parent.Parent.Name
End Function
```

Returning Excel's Version Number

The following function returns Excel's version number. For example, if you use Excel 2002, it returns the text string *10.0*.

```
Function EXCELVERSION() as String
' Returns Excel's version number
  EXCELVERSION = Application.Version
End Function
```

Note that the EXCELVERSION function returns a string, not a value. The following function returns TRUE if the application is Excel 97 or later (Excel 97 is version 8). This function uses VBA's Val function to convert the text string to a value.

```
Function EXCEL97ORLATER() As Boolean
  EXCEL97ORLATER = Val(Application.Version) >= 8
End Function
```

Returning Cell Formatting Information

This section contains a number of custom functions that return information about a cell's formatting. These functions are useful if you need to sort data based on formatting (for example, sorting all bold cells together).



The functions in this section use the following statement:

```
Application.Volatile True
```

This statement causes the function to be reevaluated when the workbook is calculated. You'll find, however, that these functions don't always return the

correct value. This is because changing cell formatting, for example, does not trigger Excel's recalculation engine. To force a global recalculation (and update all of the custom functions), press Ctrl+Alt+F9.

The following function returns TRUE if its single-cell argument has bold formatting.

```
Function ISBOLD(cell) As Boolean
' Returns TRUE if cell is bold
  Application.Volatile True
  ISBOLD = cell.Range("A1").Font.Bold
End Function
```

The following function returns TRUE if its single-cell argument has italic formatting.

```
Function ISITALIC(cell) As Boolean
' Returns TRUE if cell is italic
  Application.Volatile True
  ISITALIC = cell.Range("A1").Font.Italic
End Function
```

Both of the preceding functions have a slight flaw: They return an error if the cell has mixed formatting. For example, it's possible that only some characters are bold. The following function returns TRUE only if all the characters in the cell are bold. It uses VBA's IsNull function to determine whether the Bold property of the Font object returns Null. If so, the cell contains mixed bold formatting.

```
Function ALLBOLD(cell) As Boolean
' Returns TRUE if all characters in cell
' are bold
  Dim UpperLeft As Range
  Set UpperLeft = cell.Range("A1")
  ALLBOLD = False
  If UpperLeft.Font.Bold Then ALLBOLD = True
End Function
```

The following FILLCOLOR function returns an integer that corresponds to the color index of the cell's interior (the cell's fill color). If the cell's interior is not filled, the function returns -4142.

```
Function FILLCOLOR(cell) As Integer
' Returns an integer corresponding to
' cell's interior color
```

```
Application.Volatile True
FILLCOLOR = cell.Range("A1").Interior.ColorIndex
End Function
```

The following function returns the number format string for a cell.

```
Function NUMBERFORMAT(cell) As String
' Returns a string that represents
' the cell's number format
Application.Volatile True
NUMBERFORMAT = cell.Range("A1").NumberFormat
End Function
```

If the cell uses the default number format, the function returns the string *General*.

Determining a Cell's Data Type

Excel provides a number of built-in functions that can help determine the type of data contained in a cell. These include ISTEXT, ISLOGICAL, and ISERROR. In addition, VBA includes functions such as ISEMPTY, ISDATE, and ISNUMERIC.

The following function accepts a range argument and returns a string (*Blank, Text, Logical, Error, Date, Time, or Value*) that describes the data type of the upper left cell in the range.

```
Function CELLTYPE(cell)
' Returns the cell type of the upper left
' cell in a range
Dim UpperLeft As Range
Application.Volatile
Set UpperLeft = cell.Range("A1")
Select Case True
    Case UpperLeft.NumberFormat = "@"
        CELLTYPE = "Text"
    Case IsEmpty(UpperLeft)
        CELLTYPE = "Blank"
    Case WorksheetFunction.IsText(UpperLeft)
        CELLTYPE = "Text"
    Case WorksheetFunction.IsLogical(UpperLeft)
        CELLTYPE = "Logical"
    Case WorksheetFunction.IsErr(UpperLeft)
        CELLTYPE = "Error"
    Case IsDate(UpperLeft)
        CELLTYPE = "Date"
```



```

Case InStr(1, UpperLeft.Text, ":") <> 0
    CELLTYP = "Time"
Case IsNumeric(UpperLeft)
    CELLTYP = "Value"
End Select
End Function

```

Figure 25-1 shows the CELLTYP function in use. Column B contains formulas that use the CELLTYP function with an argument from column A. For example, cell B1 contains the following formula:

```
=CELLTYP(A1)
```

	A	B	C	D
1	1	Value	A simple value	
2	8.6	Value	Formula that returns a value	
3	Budget Sheet	Text	Simple text	
4	FALSE	Logical	Logical formula	
5	TRUE	Logical	Logical value	
6	#DIV/0!	Error	Formula error	
7	5/17/2001	Date	Formula that returns a date	
8	4:00 PM	Time	A time	
9	143	Text	Value preceded by apostrophe	
10	434	Text	Cell formatted as Text	
11	A1:C4	Text	Text with a colon	
12		Blank	Empty cell	
13		Text	Cell with a single space	
14				
15				

Figure 25-1: The CELLTYP function returns a string that describes the contents of a cell.



A workbook that demonstrates the CELLTYP function is available on the companion CD-ROM.

A Multifunctional Function

This section demonstrates a technique that may be helpful in some situations – the technique of making a single worksheet function act like multiple functions. The following VBA function, named STATFUNCTION, takes two arguments – the range (rng) and the operation (op). Depending on the value of op, the function returns a value computed by using any of the following worksheet functions: AVERAGE, COUNT, MAX, MEDIAN, MIN, MODE, STDEV, SUM, or VAR. For example, you can use this function in your worksheet:

```
=STATFUNCTION(B1:B24,A24)
```

The result of the formula depends on the contents of cell A24, which should be a string, such as *Average*, *Count*, *Max*, and so on. You can adapt this technique for other types of functions.

```
Function STATFUNCTION(rng, op)
    Select Case UCase(op)
        Case "SUM"
            STATFUNCTION = Application.Sum(rng)
        Case "AVERAGE"
            STATFUNCTION = Application.Average(rng)
        Case "MEDIAN"
            STATFUNCTION = Application.Median(rng)
        Case "MODE"
            STATFUNCTION = Application.Mode(rng)
        Case "COUNT"
            STATFUNCTION = Application.Count(rng)
        Case "MAX"
            STATFUNCTION = Application.Max(rng)
        Case "MIN"
            STATFUNCTION = Application.Min(rng)
        Case "VAR"
            STATFUNCTION = Application.Var(rng)
        Case "STDEV"
            STATFUNCTION = Application.StDev(rng)
        Case Else
            STATFUNCTION = CVErr(xlErrNA)
    End Select
End Function
```

Figure 25-2 shows the STATFUNCTION function that is used in conjunction with a drop-down list generated by Excel's Data → Validation command. The formula in cell C14 is:

```
=STATFUNCTION(C1:C12,B14)
```



The workbook shown in Figure 25-2 is available on the companion CD-ROM.

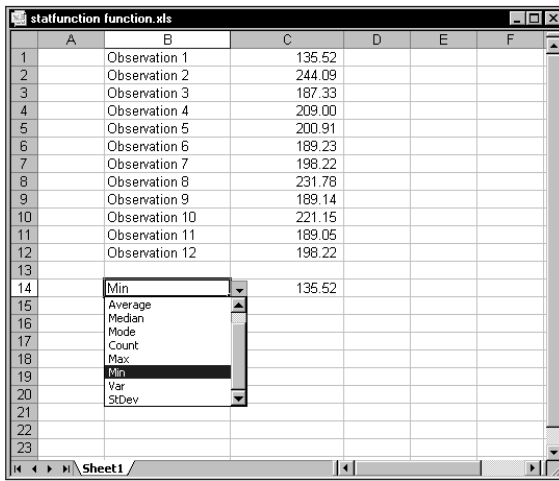


Figure 25-2: Selecting an operation from the list displays the result in cell B14.

The following `STATFUNCTION2` function is a much simpler approach that works exactly like the `STATFUNCTION` function. It uses the `Evaluate` method to evaluate an expression.

```
Function STATFUNCTION2(rng, op)
    STATFUNCTION2 = Evaluate(Op & "(" & _
        rng.Address(external:=True) & ")")
End Function
```

For example, assume that the `rng` argument is `C1:C12`, and the `op` argument is the string `SUM`. The expression that is used as an argument for the `Evaluate` method is:

```
SUM(C1:C12)
```

The `Evaluate` method evaluates its argument and returns the result. In addition to being much shorter, a benefit of this version of `STATFUNCTION` is that it's not necessary to list all of the possible functions.

Generating Random Numbers

This section presents two functions that deal with random numbers. One generates random numbers that don't change. The other selects a cell at random from a range.

Generating Random Numbers That Don't Change

You can use Excel's RAND function to quickly fill a range of cells with random values. But, as you may have discovered, the RAND function generates a new random number whenever the worksheet is recalculated. If you prefer to generate random numbers that don't change with each recalculation, use the following STATICRAND Function procedure:

```
Function STATICRAND()  
' Returns a random number that doesn't  
' change when recalculated  
    STATICRAND = Rnd  
End Function
```

The STATICRAND function uses VBA's Rnd function, which, like Excel's RAND function, returns a random number between 0 and 1. When you use STATICRAND, however, the random numbers don't change when the sheet is calculated.



Pressing F9 does not generate new values from the STATICRAND function, but pressing Ctrl+Alt+F9 (Excel's "global recalc" key combination) does.

If you want to generate a series of random integers between 1 and 1000, you can use a formula such as

```
=INT(STATICRAND()*1000)+1
```

Selecting a Cell at Random

The following function, named DRAWONE, randomly chooses one cell from an input range and returns the cell's contents.

```
Function DRAWONE(rng)  
' Chooses one cell at random from a range  
    DRAWONE = rng(Int((rng.Count) * Rnd + 1))  
End Function
```

If you use this function, you'll find that it is not recalculated when the worksheet is calculated. In other words, the function is not a volatile function (for more information about controlling recalculation, see the sidebar, "Controlling Function Recalculation," later in this chapter). You can make the function volatile by adding the following statement:

```
Application.Volatile True
```

Controlling Function Recalculation

When you use a custom function in a worksheet formula, when is it recalculated?

Custom functions behave like Excel's built-in worksheet functions. Normally, a custom function is recalculated only when it needs to be recalculated — that is, when you modify any of a function's arguments — but you can force functions to recalculate more frequently. Adding the following statement to a Function procedure makes the function recalculate whenever any cell changes:

```
Application.Volatile True
```

The `Volatile` method of the `Application` object has one argument (either `True` or `False`). Marking a function procedure as "volatile" forces the function to be calculated whenever calculation occurs in *any* cell in the worksheet.

For example, the custom `STATICRAND` function presented in this chapter can be changed to emulate Excel's `RAND()` function by using the `Volatile` method, as follows:

```
Function NONSTATICRAND()  
' Returns a random number that  
' changes when the sheet is recalculated  
Application.Volatile True  
NONSTATICRAND = Rnd  
End Function
```

Using the `False` argument of the `Volatile` method causes the function to be recalculated only when one or more of its arguments change (if a function has no arguments, this method has no effect). By default, all functions work as if they include an `Application.Volatile False` statement.

After doing so, the `DRAWONE` function displays a new random cell value whenever the sheet is calculated.



I present two additional functions that deal with randomization later in this chapter (see "Advanced Function Techniques").

Calculating Sales Commissions

Sales managers often need to calculate the commissions earned by their sales forces. The calculations in the function example presented here are based on a sliding scale: Employees who sell more earn a higher commission rate (see Table 25-1).

For example, a salesperson with sales between \$10,000 and \$19,999 qualifies for a commission rate of 10.5 percent.

TABLE 25-1 COMMISSION RATES FOR MONTHLY SALES

Monthly Sales	Commission Rate
Less than \$10,000	8.0%
\$10,000 - \$19,999	10.5%
\$20,000 - \$39,999	12.0%
\$40,000 or more	14.0%

You can calculate commissions for various sales amounts entered into a worksheet in several ways. You can use a complex formula with nested IF functions, such as the following.

```
=IF(A1<0,0,IF(A1<10000,A1*0.08,
IF(A1<20000,A1*0.105,
IF(A1<40000,A1*0.12,A1*0.14))))
```

This may not be the best approach for a couple of reasons. First, the formula is overly complex, thus making it difficult to understand. Second, the values are hard-coded into the formula, thus making the formula difficult to modify. And if you have more than seven commission rates, you run up against Excel's limit on nested functions.

A better approach is to use a lookup table function to compute the commissions. For example:

```
=VLOOKUP(A1,Table,2)*A1
```

Using VLOOKUP is a good alternative, but it may not work if the commission structure is more complex. (See the following subsection, "A Function for a More Complex Commission Structure.") Yet another approach is to create a custom function.

A Function for a Simple Commission Structure

The following COMMISSION function accepts a single argument (Sales) and computes the commission amount.

```
Function COMMISSION(Sales) As Single
'   Calculates sales commissions
  Const Tier1 As Double = 0.08
  Const Tier2 As Double = 0.105
  Const Tier3 As Double = 0.12
  Const Tier4 As Double = 0.14
  Select Case Sales
    Case Is >= 40000
      COMMISSION2 = Sales * Tier4
    Case Is >= 20000
      COMMISSION2 = Sales * Tier3
    Case Is >= 10000
      COMMISSION2 = Sales * Tier2
    Case Is < 10000
      COMMISSION2 = Sales * Tier1
  End Select
End Function
```

The following worksheet formula, for example, returns 3,000 (the sales amount – 25,000 – qualifies for a commission rate of 12 percent):

```
=COMMISSION(25000)
```

This function is very easy to understand and maintain. It uses constants to store the commission rates, and a Select Case structure to determine which commission rate to use.



When a Select Case structure is evaluated, program control exits the Select Case structure when the first true Case is encountered.

A Function for a More Complex Commission Structure

If the commission structure is more complex, you may need to use additional arguments for your COMMISSION function. Imagine that the aforementioned sales manager implements a new policy to help reduce turnover: The total commission paid increases by 1 percent for each year that a salesperson stays with the company.

The following is a modified COMMISSION function (named COMMISSION2). This function now takes two arguments: The monthly sales (Sales) and the number of years employed (Years).

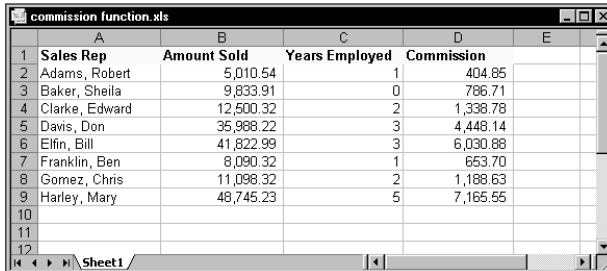
```

Function COMMISSION2(Sales, Years) As Single
'   Calculates sales commissions based on
'   years in service
Const Tier1 As Double = 0.08
Const Tier2 As Double = 0.105
Const Tier3 As Double = 0.12
Const Tier4 As Double = 0.14
Select Case Sales
    Case Is >= 40000
        COMMISSION2 = Sales * Tier4
    Case Is >= 20000
        COMMISSION2 = Sales * Tier3
    Case Is >= 10000
        COMMISSION2 = Sales * Tier2
    Case Is < 10000
        COMMISSION2 = Sales * Tier1
End Select
COMMISSION2 = COMMISSION2 + (COMMISSION2 * Years / 100)
End Function

```

Figure 25-3 shows the COMMISSION2 function in use. The formula in cell D2 is

=COMMISSION2(B2,C2)



	A	B	C	D	E
1	Sales Rep	Amount Sold	Years Employed	Commission	
2	Adams, Robert	5,010.54	1	404.85	
3	Baker, Sheila	9,833.91	0	786.71	
4	Clarke, Edward	12,500.32	2	1,338.78	
5	Davis, Don	35,968.22	3	4,448.14	
6	Elfin, Bill	41,822.99	3	6,030.88	
7	Franklin, Ben	8,090.32	1	653.70	
8	Gomez, Chris	11,098.32	2	1,188.63	
9	Harley, Mary	48,745.23	5	7,165.55	
10					
11					
12					

Figure 25-3: Calculating sales commissions based on sales amount and years employed



The workbook shown in Figure 25-3 is available on the companion CD-ROM.

Text Manipulation Functions

Text strings can be manipulated with functions in a variety of ways, including reversing the display of a text string, scrambling the characters in a text string, or extracting specific characters from a text string. This section offers a number of function examples that manipulate text strings.



The companion CD-ROM contains a workbook that demonstrates all of the functions in this section.

Reversing a String

The following REVERSETEXT function returns the text in a cell backwards.

```
Function REVERSETEXT(text) As String
' Returns its argument, reversed
  REVERSETEXT = StrReverse(text)
End Function
```

This function simply uses VBA's StrReverse function. The following formula, for example, returns *tfosorciM*.

```
=REVERSETEXT("Microsoft")
```

The StrReverse function is not available with versions of Excel prior to Excel 2000. Therefore, if you need this functionality with an earlier version of Excel, you'll need to "roll your own." The following REVERSETEXT2 function works just like the REVERSETEXT function.

```
Function REVERSETEXT2(text) As String
' Returns its argument, reversed
' For use with versions prior to Excel 2000
  Dim TextLen As Integer
  Dim i As Integer
  TextLen = Len(text)
  For i = TextLen To 1 Step -1
    REVERSETEXT2 = REVERSETEXT2 & Mid(text, i, 1)
  Next i
End Function
```

The function uses a For-Next loop with a negative Step value. The letters are concatenated (using &t, which is the concatenation operator) to form the string in reverse order.

Scrambling Text

The following function returns the contents of its argument with the characters randomized. For example, using *Microsoft* as the argument may return *oficMorts*, or some other random permutation.

```
Function SCRAMBLE(text)
'   Scrambles its single-cell argument
  Dim TextLen As Integer
  Dim i As Integer
  Dim RandPos As Integer
  Dim Char As String * 1
  Set text = text.Range("A1")
  TextLen = Len(text)
  For i = 1 To TextLen
    Char = Mid(text, i, 1)
    RandPos = Int((TextLen - 1 + 1) * Rnd + 1)
    Mid(text, i, 1) = Mid(text, RandPos, 1)
    Mid(text, RandPos, 1) = Char
  Next i
  SCRAMBLE = text
End Function
```

This function loops through each character, and then swaps it with another character in a randomly selected position.

You may be wondering about the use of Mid. Note that when Mid is used on the right side of an assignment statement, it is a function. But when Mid is used on the left side of the assignment statement, it is a statement. Consult the online help for more information about Mid.

Returning an Acronym

The ACRONYM function returns the first letter (in uppercase) of each word in its argument. For example, the following formula returns *IBM*.

```
=ACRONYM("International Business Machines")
```

The listing for the ACRONYM Function procedure follows:

```
Function ACRONYM(text) As String
'   Returns an acronym for text
  Dim TextLen As Integer
```

```

Dim i As Integer
text = Application.Trim(text)
TextLen = Len(text)
ACRONYM = Left(text, 1)
For i = 2 To TextLen
    If Mid(text, i, 1) = " " Then
        ACRONYM = ACRONYM & Mid(text, i + 1, 1)
    End If
Next i
ACRONYM = UCase(ACRONYM)
End Function

```

This function uses Excel's TRIM function to remove any extra spaces from the argument. The first character in the argument is always the first character in the result. The For-Next loop examines each character. If the character is a space, then the character *after* the space is appended to the result. Finally, the result converts to uppercase by using VBA's UCase function.

Does the Text Match a Pattern?

The following function returns TRUE if a string matches a pattern composed of text and wildcard characters. The ISLIKE function is remarkably simple, and is essentially a *wrapper* for VBA's useful Like operator.

```

Function ISLIKE(text As String, pattern As String) As Boolean
    Returns true if the first argument is like the second
    ISLIKE = text Like pattern
End Function

```

The supported wildcard characters are as follows:

- ? Matches any single character
- * Matches zero or more characters
- # Matches any single digit (0–9)
- [list] Matches any single character in the list
- [!list] Matches any single character not in the list

The following formula returns TRUE because the question mark (?) matches any single character. If the first argument were "Unit12," then the function would return FALSE.

```
=ISLIKE("Unit1","Unit?")
```

The ISLIKE function also works with values. The following formula, for example, returns TRUE if cell A1 contains a value that begins with 1 and has exactly three numeric digits.

```
=ISLIKE(A1,"1###")
```

The following formula returns TRUE because the first argument is a single character contained in the list of characters specified in the second argument.

```
=ISLIKE("a","[aeiou]")
```

If the character list begins with an exclamation point (!), then the comparison is made with characters *not* in the list. For example, the following formula returns TRUE because the first argument is a single character that does not appear in the second argument's list.

```
=ISLIKE("g","[!aeiou]")
```

The Like operator is very versatile. For complete information about VBA's Like operator, consult the online help.

Does a Cell Contain Text?

Chapter 5 describes how a number of Excel's worksheet functions are at times unreliable when dealing with text in a cell. The following CELLHASTEXT function returns TRUE if the cell argument contains text or contains a value formatted as Text.

```
Function CELLHASTEXT(cell) As Boolean
' Returns TRUE if cell contains a string
' or cell is formatted as Text
Dim UpperLeft as Range
CELLHASTEXT = False
Set UpperLeft = cell.Range("A1")
If UpperLeft.NumberFormat = "@" Then
    CELLHASTEXT = True
    Exit Function
End If
If Not IsNumeric(UpperLeft) Then
    CELLHASTEXT = True
    Exit Function
End If
End Function
```

The following formula returns TRUE if cell A1 contains a text string or if the cell is formatted as Text.

```
=CELLHASTEXT(A1)
```

Extracting the *n*th Element from a String

The `EXTRACTELEMENT` function is a custom worksheet function that extracts an element from a text string based on a specified separator character. Assume that cell A1 contains the following text:

```
123-456-789-9133-8844
```

For example, the following formula returns the string `9133`, which is the fourth element in the string. The string uses a hyphen (-) as the separator.

```
=EXTRACTELEMENT(A1,4,"-")
```

The `EXTRACTELEMENT` function uses three arguments:

- ◆ *Txt*: The text string from which you're extracting. This can be a literal string or a cell reference.
- ◆ *n*: An integer that represents the element to extract.
- ◆ *Separator*: A single character used as the separator.



If you specify a space as the Separator character, then multiple spaces are treated as a single space (almost always what you want). If *n* exceeds the number of elements in the string, the function returns an empty string.

The VBA code for the `EXTRACTELEMENT` function follows:

```
Function EXTRACTELEMENT(Txt, n, Separator) As String
' Returns the nth element of a text string, where the
' elements are separated by a specified separator character
Dim AllElements As Variant
AllElements = Split(Txt, Separator)
EXTRACTELEMENT = AllElements(n - 1)
End Function
```

This function uses VBA's Split function, which returns a variant array that contains each element of the text string. This array begins with 0 (not 1), so using n-1 references the desired element.

The Split function was introduced in Excel 2000. If you're using an older version of Excel, then you'll need to use the following function:

```
Function EXTRACTELEMENT2(Txt, n, Separator) As String
' Returns the nth element of a text string, where the
' elements are separated by a specified separator character

    Dim Txt1 As String, TempElement As String
    Dim ElementCount As Integer, i As Integer

    Txt1 = Txt
    ' If space separator, remove excess spaces
    If Separator = Chr(32) Then Txt1 = Application.Trim(Txt1)

    ' Add a separator to the end of the string
    If Right(Txt1, Len(Txt1)) <> Separator Then _
        Txt1 = Txt1 & Separator

    ' Initialize
    ElementCount = 0
    TempElement = ""

    ' Extract each element
    For i = 1 To Len(Txt1)
        If Mid(Txt1, i, 1) = Separator Then
            ElementCount = ElementCount + 1
            If ElementCount = n Then
                ' Found it, so exit
                EXTRACTELEMENT2 = TempElement
                Exit Function
            Else
                TempElement = ""
            End If
        Else
            TempElement = TempElement & Mid(Txt1, i, 1)
        End If
    Next i
    EXTRACTELEMENT2 = ""
End Function
```

Spelling Out a Number

The SPELLDOLLARS function returns a number spelled out in text – as on a check. For example, the following formula returns the string *One hundred twenty-three and 45/100 dollars*.

```
=SPELLDOLLARS(123.45)
```

Figure 25-4 shows some additional examples of the SPELLDOLLARS function. Column C contains formulas that use the function. For example, the formula in C1 is:

```
=SPELLDOLLARS(A1)
```

Note that negative numbers are spelled out and enclosed in parentheses.

	A	B	C	D	E
1	32		Thirty-Two and 00/100 Dollars		
2	37.56		Thirty-Seven and 56/100 Dollars		
3	-32		(Thirty-Two and 00/100 Dollars)		
4	-26.44		(Twenty-Six and 44/100 Dollars)		
5	-4		(Four and 00/100 Dollars)		
6	1.56		One and 56/100 Dollars		
7	1		One and 00/100 Dollars		
8	6.56		Six and 56/100 Dollars		
9	12.12		Twelve and 12/100 Dollars		
10	1000000		One Million and 00/100 Dollars		
11	10000000000		Ten Billion and 00/100 Dollars		
12					
13					
14					

Figure 25-4: Examples of the SPELLDOLLARS function



The SPELLDOLLARS function is too lengthy to list here, but you can view the complete listing in the workbook on the companion CD-ROM.

Counting and Summing Functions

Chapter 7 contains many formula examples to count and sum cells based on various criteria. If you can't arrive at a formula-based solution for a counting or summing problem, then you can probably create a custom function. This section contains three functions that perform counting or summing.



The companion CD-ROM contains a workbook that demonstrates the functions in this section.

Counting Cells Between Two Values

Assume that you need to count the number of values between 6 and 12 in the range A1:A100. The following formula will do the job:

```
=COUNTIF(A1:A100, "<=12")-COUNTIF(A1:A100, "<6")
```

This formula works well, but setting it up can be confusing. The formula actually counts the number of cells that are less than or equal to 12 and then subtracts the number of cells that are less than 6.

The following COUNTBETWEEN function is essentially a “wrapper” for this type of formula:

```
Function COUNTBETWEEN(rng, num1, num2)
    COUNTBETWEEN = Application.CountIf(rng, "<=" & num2) _
        - Application.CountIf(rng, "<" & num1)
End Function
```

The COUNTBETWEEN function accepts three arguments:

- ◆ *rng*: A range reference
- ◆ *num1*: The lower limit
- ◆ *num2*: The upper limit

The function uses Excel’s COUNTIF function, and returns the number of cells in *rng* that are greater than or equal to *num1* and less than or equal to *num2*.

Counting Visible Cells in a Range

The following COUNTVISIBLE function accepts a range argument and returns the number of non-empty visible cells in the range. A cell is not visible if it resides in a hidden row or a hidden column.

```
Function COUNTVISIBLE(rng)
' Counts visible cells
Dim CellCount As Long
Dim cell As Range
Application.Volatile
```



```

CellCount = 0
Set rng = Intersect(rng.Parent.UsedRange, rng)
For Each cell In rng
    If Not IsEmpty(cell) Then
        If Not cell.EntireRow.Hidden And _
            Not cell.EntireColumn.Hidden Then _
            CellCount = CellCount + 1
        End If
    Next cell
COUNTVISIBLE = CellCount
End Function

```

This function loops through each cell in the range, checking first to see if the cell is empty. If the cell is not empty, then this function checks the Hidden properties of the cell's row and column. If either the row or column is hidden, then the CellCount variable increments.

If you're working with AutoFilters or outlines, you may prefer to use Excel's SUBTOTAL function (with a first argument of 2 or 3). The SUBTOTAL function, however, does not work properly if cells are hidden manually by using the Format → Row → Hide or Format → Column → Hide commands. In such a case, the COUNTVISIBLE function is the only alternative.

Summing Visible Cells in a Range

The SUMVISIBLE function is based on the COUNTVISIBLE function discussed in the previous section. This function accepts a range argument and returns the sum of the visible cells in the range. A cell is not visible if it resides in a hidden row or a hidden column.

```

Function SUMVISIBLE(rng)
' Sums only visible cells
Dim CellSum As Double
Dim cell As Range
Application.Volatile
CellSum = 0
Set rng = Intersect(rng.Parent.UsedRange, rng)
For Each cell In rng
    If IsNumeric(cell) Then
        If Not cell.EntireRow.Hidden And _
            Not cell.EntireColumn.Hidden Then _
            CellSum = CellSum + cell
        End If
    Next cell
SUMVISIBLE = CellSum
End Function

```

Hiding and unhiding rows and columns don't trigger a worksheet recalculation. Therefore, you may need to press Ctrl+Alt+F9 to force a complete recalculation.

Excel's SUBTOTAL function (with a first argument of 9) is also useful for summing visible cells in an AutoFiltered list. The SUBTOTAL function, however, does not work properly if cells are hidden in a non-filtered list.

Date Functions

Chapter 6 presents a number of useful Excel functions and formulas for calculating dates, times, and time periods by manipulating date and time serial values. This section presents additional functions that deal with dates.



The companion CD-ROM contains a workbook that demonstrates the Date functions presented in this section.

Calculating the Next Monday

The following NEXTMONDAY function accepts a date argument and returns the date of the following Monday.

```
Function NEXTMONDAY(d As Date) As Date
    NEXTMONDAY = d + 8 - WeekDay(d, vbMonday)
End Function
```

This function uses VBA's WeekDay function, which returns an integer that represents the day of the week for a date (1 = Sunday, 2 = Monday, and so on). It also uses a predefined constant, vbMonday.

The following formula returns 12/31/2001, which is the first Monday after Christmas Day, 2001 (which is a Tuesday):

```
=NEXTMONDAY(DATE(2001,12,25))
```



The function returns a date serial number. You will need to change the number format of the cell to display this serial number as an actual date.

If the argument passed to the NEXTMONDAY function is a Monday, the function will return the *following* Monday. If you prefer the function to return the same Monday, use this modified version:

```
Function NEXTMONDAY2(d As Date) As Date
    If WeekDay(d) = 2 Then
        NEXTMONDAY2 = d
    Else
        NEXTMONDAY2 = d + 8 - WeekDay(d, vbMonday)
    End If
End Function
```

Calculating the Next Day of the Week

The following NEXTDAY function is a variation on the NEXTMONDAY function. This function accepts two arguments: A date and an integer between 1 and 7 that represents a day of the week (1 = Sunday, 2 = Monday, and so on). The NEXTDAY function returns the date for the next specified day of the week.

```
Function NEXTDAY(d As Date, day As Integer) As Variant
    ' Returns the next specified day
    ' Make sure day is between 1 and 7
    If day < 1 Or day > 7 Then
        NEXTDAY = CVErr(xlErrNA)
    Else
        NEXTDAY = d + 8 - WeekDay(d, day)
    End If
End Function
```

The NEXTDAY function uses an If statement to ensure that the day argument is valid (that is, between 1 and 7). If the day argument is not valid, the function returns #N/A. Because the function can return a value other than a date, it is declared as type variant.

Which Week of the Month?

The following MONTHWEEK function returns an integer that corresponds to the week of the month for a date.

```
Function MONTHWEEK(d As Date) As Variant
    ' Returns the week of the month for a date
    Dim FirstDay As Integer

    ' Check for valid date argument
```

```
If Not IsDate(d) Then
    MONTHWEEK = CVErr(xlErrNA)
    Exit Function
End If

' Get first day of the month
FirstDay = WeekDay(DateSerial(Year(d), Month(d), 1))

' Calculate the week number
MONTHWEEK = Application.RoundUp((FirstDay + day(d) - 1) / 7, 0)
End Function
```

Working with Dates Before 1900

Many users are surprised to discover that Excel can't work with dates prior to the year 1900. To correct this deficiency, I created an add-in called "Extended Date Functions." This add-in enables you to work with dates in the years 0100 through 9999.



The companion CD-ROM contains a copy of the Extended Date Functions add-in.

When installed, the Extended Date Function add-in gives you access to eight new worksheet functions:

- ◆ **XDATE(y,m,d,fmt)**: Returns a date for a given year, month, and day. As an option, you can provide a date formatting string.
- ◆ **XDATEADD(xdate1,days,fmt)**: Adds a specified number of days to a date. As an option, you can provide a date formatting string.
- ◆ **XDATEDIF(xdate1,xdate2)**: Returns the number of days between two dates.
- ◆ **XDATEYEARDIF(xdate1,xdate2)**: Returns the number of full years between two dates (useful for calculating ages).
- ◆ **XDATEYEAR(xdate1)**: Returns the year of a date.
- ◆ **XDATEMONTH(xdate1)**: Returns the month of a date.
- ◆ **XDATEDAY(xdate1)**: Returns the day of a date.
- ◆ **XDATEDOW(xdate1)**: Returns the day of the week of a date (as an integer between 1 and 7).



These functions don't make any adjustments for changes made to the calendar in 1582. Consequently, working with dates prior to October 15, 1582, may not yield correct results.

Returning the Last Nonempty Cell in a Column or Row

This section presents two useful functions: `LASTINCOLUMN`, which returns the contents of the last nonempty cell in a column, and `LASTINROW`, which returns the contents of the last nonempty cell in a row. Chapter 13 presents array formulas for this task, but you may prefer to use a custom function.



The companion CD-ROM contains a workbook that demonstrates the functions presented in this section.

Each of these functions accepts a range as its single argument. The range argument can be a column reference (for `LASTINCOLUMN`) or a row reference (for `LASTINROW`). If the supplied argument is not a complete column or row reference (such as `3:3` or `D:D`), the function uses the column or row of the upper-left cell in the range. For example, the following formula returns the contents of the last nonempty cell in column B:

```
=LASTINCOLUMN(B5)
```

The following formula returns the contents of the last nonempty cell in row 7:

```
=LASTINROW(C7:D9)
```

The `LASTINCOLUMN` Function

The following is the `LASTINCOLUMN` function:

```
Function LASTINCOLUMN(rng As Range)
' Returns the contents of the last non-empty cell in a column
  Dim LastCell As Range
  Application.Volatile
  With rng.Parent
    With .Cells(.Rows.Count, rng.Column)
```

```
    If Not IsEmpty(.Value) Then
        LASTINCOLUMN = .Value
    ElseIf IsEmpty(.End(xlUp)) Then
        LASTINCOLUMN = ""
    Else
        LASTINCOLUMN = .End(xlUp).Value
    End If
End With
End With
End Function
```

Notice the references to the Parent of the range. This is done in order to make the function work with arguments that refer to a different worksheet or workbook.

The LASTINROW Function

The following is the LASTINROW function:

```
Function LASTINROW(rng As Range)
' Returns the contents of the last non-empty cell in a row
Application.Volatile
With rng.Parent
    With .Cells(rng.Row, .Columns.Count)
        If Not IsEmpty(.Value) Then
            LASTINROW = .Value
        ElseIf IsEmpty(.End(xlToLeft)) Then
            LASTINROW = ""
        Else
            LASTINROW = .End(xlToLeft).Value
        End If
    End With
End With
End Function
```

Multisheet Functions

You may need to create a function that works with data contained in more than one worksheet within a workbook. This section contains two VBA functions that enable you to work with data across multiple sheets, including a function that overcomes an Excel limitation when copying formulas to other sheets.



The companion CD-ROM contains a workbook that demonstrates the multi-sheet functions presented in this section.

Returning the Maximum Value Across All Worksheets

If you need to determine the maximum value in a cell (for example, B1) across a number of worksheets, use a formula like this one:

```
=MAX(Sheet1:Sheet4!B1)
```

This formula returns the maximum value in cell B1 for Sheet1, Sheet4, and all of the sheets in between. But what if you add a new sheet (Sheet5) after Sheet4? Your formula does not adjust automatically, so you need to edit it to include the new sheet reference:

```
=MAX(Sheet1:Sheet5!B1)
```

The following function accepts a single-cell argument, and returns the maximum value in that cell across all worksheets in the workbook. For example, the following formula returns the maximum value in cell B1 for all sheets in the workbook.

```
=MAXALLSHEETS(B1)
```

If you add a new sheet, you don't need to edit the formula.

```
Function MAXALLSHEETS(cell as Range)
    Dim MaxVal As Double
    Dim Addr As String
    Dim Wksht As Object
    Application.Volatile
    Addr = cell.Range("A1").Address
    MaxVal = -9.9E+307
    For Each Wksht In cell.Parent.Parent.Worksheets
        If Wksht.Name = cell.Parent.Name And _
            Addr = Application.Caller.Address Then
            ' avoid circular reference
        Else
            If IsNumeric(Wksht.Range(Addr)) Then
                If Wksht.Range(Addr) > MaxVal Then _
                    MaxVal = Wksht.Range(Addr).Value
            End If
        End If
    Next Wksht
End Function
```

```

        End If
    End If
Next Wksht
If MaxVal = -9.9E+307 Then MaxVal = 0
MAXALLSHEETS = MaxVal
End Function

```

The For Each statement uses the following expression to access the workbook:

```
cell.Parent.Parent.Worksheets
```

The parent of the cell is a worksheet, and the parent of the worksheet is the workbook. Therefore, the For Each-Next loop cycles among all worksheets in the workbook. The first If statement inside of the loop performs a check to see if the cell being checked is the cell that contains the function. If so, that cell is ignored to avoid a circular reference error.



You can easily modify the MAXALLSHEETS function to perform other cross-worksheet calculations: Minimum, Average, Sum, and so on.

The SHEETOFFSET Function

A recurring complaint about Excel (including Excel 2002) is its poor support for relative sheet references. For example, suppose that you have a multisheet workbook, and you enter a formula like the following on Sheet2:

```
=Sheet1!A1+1
```

This formula works fine. However, if you copy the formula to the next sheet (Sheet3), the formula continues to refer to Sheet1. Or, if you insert a sheet between Sheet1 and Sheet2, the formula continues to refer to Sheet1 (most likely, you want it to refer to the newly inserted sheet). In fact, you can't create formulas that refer to worksheets in a relative manner. However, you can use the SHEETOFFSET function to overcome this limitation.

THE SHEETOFFSET FUNCTION: TAKE ONE

Following is a VBA Function procedure named SHEETOFFSET.

```

Function SHEETOFFSET(offset As Integer, Ref As Range)
' Returns cell contents at Ref, in sheet offset
    Dim WksIndex As Integer
    Application.Volatile

```



```

WksIndex = WorksheetIndex(Application.Caller.Parent)
SHEETOFFSET = Worksheets(WksIndex + offset).Range(Ref.Address)
End Function

```

The SHEETOFFSET function accepts two arguments:

- ◆ *offset*: The sheet offset, which can be positive, negative, or 0.
- ◆ *ref*: A single-cell reference. If the *offset* argument is 0, the cell reference must not be the same as the cell that contains the formula. If so, you get a circular reference error.

The following formula returns the value in cell A1 of the sheet before the sheet that contains the formula:

```
=SHEETOFFSET(-1,A1)
```

The following formula returns the value in cell A1 of the sheet after the sheet that contains the formula:

```
=SHEETOFFSET(1,A1)
```

This function works fine in most cases. For example, you can copy the formula to other sheets and the relative referencing will be in effect in all of the copied formulas. And, if you insert a worksheet, the sheet reference adjusts automatically.

This function, however, has a problem: If your workbook contains non-worksheet sheets (that is, chart sheets or Excel 5 dialog sheets), the function may fail because it attempts to reference a cell on a sheet that is not a worksheet.

THE SHEETOFFSET FUNCTION: TAKE TWO

You can, nevertheless, use an improved version of the SHEETOFFSET function. This version of the function uses a second function named `WorksheetIndex`. The `WorksheetIndex` function returns the worksheet index for a `Worksheet` object passed as an argument. It then uses the value to identify another worksheet. The following is a version of SHEETOFFSET, which essentially ignores any non-worksheet sheets in the workbook.

```

Function SHEETOFFSET(offset as Integer, Ref as Range)
' Returns cell contents at Ref, in sheet offset
  Dim WksIndex As Integer
  Application.Volatile
  WksIndex = WorksheetIndex(Application.Caller.Parent)
  SHEETOFFSET = Worksheets(WksIndex + offset).Range(Ref.Address)
End Function

```

```
Private Function WorksheetIndex(x As Worksheet) As Integer
' Returns the Worksheets (not Sheets) Index
Dim Wks As Worksheet, WksNum As Integer
WksNum = 1
For Each Wks In x.Parent.Worksheets
    If x.Name = Wks.Name Then
        WorksheetIndex = WksNum
        Exit Function
    End If
    WksNum = WksNum + 1
Next Wks
End Function
```

Notice that because the `WorksheetIndex` function is not designed for use in a formula, it is declared with the `Private` keyword. Doing so prevents it from appearing in the Paste Function dialog box.

Advanced Function Techniques

In this section, I explore some even more advanced functions. The examples in this section demonstrate some special techniques that you can use with your custom functions.

- ◆ Returning an error value from a function
- ◆ Returning an array from a function
- ◆ Using optional function arguments
- ◆ Using an indefinite number of function arguments
- ◆ Using Windows API functions

Returning an Error Value

In some cases, you may want your custom function to return a particular error value. Consider the `REVERSETEXT` function, which I presented earlier in this chapter.

```
Function REVERSETEXT(text) As String
' Returns its argument, reversed
REVERSETEXT = StrReverse(text)
End Function
```

This function reverses the contents of its single-cell argument (which can be text or a value). If the argument is a multicell range, the function returns #VALUE!

Assume that you want this function to work only with strings. If the argument does not contain a string, you want the function to return an error value (#N/A). You may be tempted to simply assign a string that looks like an Excel formula error value. For example:

```
REVERSETEXT = "#N/A"
```

Although the string *looks* like an error value, it is not treated as such by other formulas that may reference it. To return a *real* error value from a function, use VBA's CVer function, which converts an error number to a real error.

Fortunately, VBA has built-in constants for the errors that you want to return from a custom function. These constants are listed here:

- ◆ xlErrDiv0
- ◆ xlErrNA
- ◆ xlErrName
- ◆ xlErrNull
- ◆ xlErrNum
- ◆ xlErrRef
- ◆ xlErrValue

The following is the revised REVERSETEXT function:

```
Function REVERSETEXT(text) As Variant
' Returns its argument, reversed
If Application.ISNONTEXT(text) Then
    REVERSETEXT = CVer(xlErrNA)
Else
    REVERSETEXT = StrReverse(text)
End If
End Function
```

This function uses Excel's ISNONTEXT function to determine whether the argument is not a text string. If the argument is not a text string, the function returns the #N/A error. Otherwise, it returns the characters in reverse order.



The data type for the original REVERSETEXT function was String, because the function returned a text string. In this revised version, the function is declared as a variant because it can now return something other than a string.

Returning an Array from a Function

Most functions that you develop with VBA return a single value. It's possible, however, to write a function that returns multiple values in an array.



Part III deals with arrays and array formulas. Specifically, these chapters provide examples of a single formula that returns multiple values in separate cells. As you'll see, you can also create custom functions that return arrays.

VBA includes a useful function called "Array." The Array function returns a variant that contains an array. It's important to understand that the array returned is not the same as a normal array composed of elements of the variant type. In other words, a variant array is not the same as an array of variants.

If you're familiar with using array formulas in Excel, then you have a head start understanding VBA's Array function. You enter an array formula into a cell by pressing Ctrl+Shift+Enter. Excel inserts brackets around the formula to indicate that it's an array formula. See Chapter 12 for more details on array formulas.



The lower bound of an array created by using the Array function is, by default, 0. However, the lower bound can be changed if you use an Option Base statement.

The following MONTHNAMES function demonstrates how to return an array from a Function procedure.

```
Function MONTHNAMES() As Variant
    MONTHNAMES = Array( _
        "Jan", "Feb", "Mar", "Apr", _
        "May", "Jun", "Jul", "Aug", _
        "Sep", "Oct", "Nov", "Dec")
End Function
```

Figure 25-5 shows a worksheet that uses the MONTHNAMES function. You enter the function by selecting A4:L4, and then entering the following formula:

```
{=MONTHNAMES() }
```

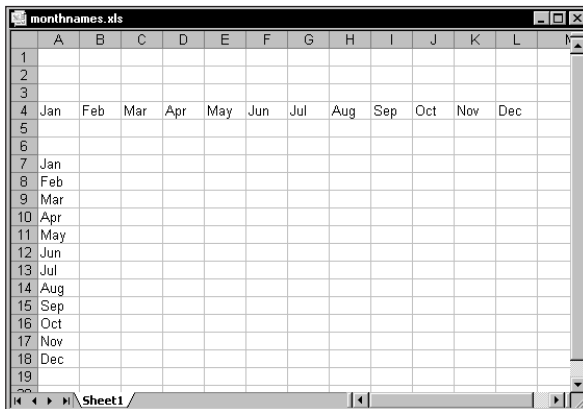


Figure 25-5: The MONTHNAMES function entered as an array formula



As with any array formula, you must press Ctrl+Shift+Enter to enter the formula. Don't enter the brackets — Excel inserts the brackets for you.

The MONTHNAMES function, as written, returns a horizontal array in a single row. To display the array in a vertical range in a single column (as in A7:A18 in Figure 25-5), select the range and enter the following formula:

```
{=TRANSPOSE(MONTHNAMES())}
```

Alternatively, you can modify the function to do the transposition. The following function uses Excel's TRANSPOSE function to return a vertical array.

```
Function VMONTHNAMES() As Variant
    VMONTHNAMES = Application.Transpose(Array( _
        "Jan", "Feb", "Mar", "Apr", _
        "May", "Jun", "Jul", "Aug", _
        "Sep", "Oct", "Nov", "Dec"))
End Function
```



A workbook that demonstrates MONTHNAMES and VMONTHNAMES is available on the companion CD-ROM.

Returning an Array of Nonduplicated Random Integers

The RANDOMINTEGERS function returns an array of nonduplicated integers. This function is intended for use in a multicell array formula. Figure 25-6 shows a worksheet that uses the following formula in the range A1:D10.

```
{=RANDOMINTEGERS() }
```

	A	B	C	D	E	F	G
1	28	19	33	38			
2	23	32	18	13			
3	36	24	26	1			
4	30	29	9	10			
5	16	22	3	25			
6	21	12	40	7			
7	17	2	39	5			
8	8	6	35	14			
9	15	4	27	31			
10	34	20	11	37			
11							

Figure 25-6: An array formula generates nonduplicated consecutive integers, arranged randomly.

This formula was entered into the entire range by using Ctrl+Shift+Enter. The formula returns an array of nonduplicated integers, arranged randomly. Because 40 cells contain the formula, the integers range from 1 to 40. The following is the code for RANDOMINTEGERS:

```
Function RANDOMINTEGERS()
    Dim FuncRange As Range

    Dim V() As Integer, ValArray() As Integer
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer
    Randomize
    ' Create Range object
    Set FuncRange = Application.Caller

    ' Return an error if FuncRange is too large
    CellCount = FuncRange.Count
    If CellCount > 1000 Then
        RANDOMINTEGERS = CVErr(xlErrNA)
        Exit Function
    End If
```

```
' Assign variables
RCount = FuncRange.Rows.Count
CCount = FuncRange.Columns.Count
ReDim V(1 To RCount, 1 To CCount)
ReDim ValArray(1 To 2, 1 To CellCount)

' Fill array with random numbers
' and consecutive integers
For i = 1 To CellCount
    ValArray(1, i) = Rnd
    ValArray(2, i) = i
Next i

' Sort ValArray by the random number dimension
For i = 1 To CellCount
    For j = i + 1 To CellCount
        If ValArray(1, i) > ValArray(1, j) Then
            Temp1 = ValArray(1, j)
            Temp2 = ValArray(2, j)
            ValArray(1, j) = ValArray(1, i)
            ValArray(2, j) = ValArray(2, i)
            ValArray(1, i) = Temp1
            ValArray(2, i) = Temp2
        End If
    Next j
Next i

' Put the randomized values into the V array
i = 0
For r = 1 To RCount
    For c = 1 To CCount
        i = i + 1
        V(r, c) = ValArray(2, i)
    Next c
Next r
RANDOMINTEGERS = V
End Function
```



A workbook containing the RANDOMINTEGERS function is available on the companion CD-ROM.

Randomizing a Range

The following RANGERANDOMIZE function accepts a range argument and returns an array that consists of the input range in random order.

```
Function RANGERANDOMIZE(rng)
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer
    Randomize

    ' Return an error if rng is too large
    CellCount = rng.Count
    If CellCount > 1000 Then
        RANGERANDOMIZE = CVErr(xlErrNA)
        Exit Function
    End If

    ' Assign variables
    RCount = rng.Rows.Count
    CCount = rng.Columns.Count
    ReDim V(1 To RCount, 1 To CCount)
    ReDim ValArray(1 To 2, 1 To CellCount)

    ' Fill ValArray with random numbers
    ' and values from rng
    For i = 1 To CellCount
        ValArray(1, i) = Rnd
        ValArray(2, i) = rng(i)
    Next i

    ' Sort ValArray by the random number dimension
    For i = 1 To CellCount
        For j = i + 1 To CellCount
            If ValArray(1, i) > ValArray(1, j) Then
                Temp1 = ValArray(1, j)
                Temp2 = ValArray(2, j)
                ValArray(1, j) = ValArray(1, i)
                ValArray(2, j) = ValArray(2, i)
                ValArray(1, i) = Temp1
                ValArray(2, i) = Temp2
            End If
        Next j
    Next i
End Function
```



```

        Next j
    Next i

    ' Put the randomized values into the V array
    i = 0
    For r = 1 To RCount
        For c = 1 To CCount
            i = i + 1
            V(r, c) = ValArray(2, i)
        Next c
    Next r
    RANGERANDOMIZE = V
End Function

```

The code closely resembles the code for the `RANDOMINTEGERS` function. Figure 25-7 shows the function in use. The array formula in C2:C11 is:

```
{=RANGERANDOMIZE(A2:A11)}
```

	A	B	C	D	E	F
1	Original		Randomized			
2	Alan		Jack			
3	Bob		Hilda			
4	Cindy		Alan			
5	Dave		Frank			
6	Ellen		Dave			
7	Frank		Ellen			
8	Ginny		Cindy			
9	Hilda		Irvin			
10	Irvin		Bob			
11	Jack		Ginny			
12						
13						

Figure 25-7: The `RANGERANDOMIZE` function returns the contents of a range, in random order.

This formula returns the contents of A2:A11, but in random order.



The workbook containing the `RANGERANDOMIZE` function is available on the companion CD-ROM.

Using Optional Arguments

Many of Excel's built-in worksheet functions use optional arguments. For example, the LEFT function returns characters from the left side of a string. Its official syntax is:

```
LEFT(text,num_chars)
```

The first argument is required, but the second is optional. If you omit the optional argument, Excel assumes a value of 1.

Custom functions that you develop in VBA can also have optional arguments. You specify an optional argument by preceding the argument's name with the keyword *Optional*. The following is a simple function that returns the user's name.

```
Function USER()  
    USER = Application.UserName  
End Function
```

Suppose that, in some cases, you want the user's name to be returned in uppercase letters. The following function uses an optional argument.

```
Function USER(Optional UpperCase As Boolean)  
    If IsMissing(UpperCase) Then UpperCase = False  
    If UpperCase = True Then  
        USER = Ucase(Application.UserName)  
    Else  
        USER = Application.UserName  
    End If  
End Function
```

If the argument is FALSE or omitted, the user's name is returned without any changes. If the argument is TRUE, the user's name converts to uppercase (using VBA's Ucase function) before it is returned. Notice that the first statement in the procedure uses VBA's IsMissing function to determine whether the argument was supplied. If the argument is missing, the statement sets the UpperCase variable to FALSE (the default value).

All of the following formulas are valid (and the first two have the same effect):

```
=USER()  
=USER(False)  
=USER(True)
```

Using an Indefinite Number of Arguments

Some of Excel's worksheet functions take an indefinite number of arguments. A familiar example is the SUM function, which has the following syntax:

```
SUM(number1,number2...)
```

The first argument is required, but you can have as many as 29 additional arguments. Here's an example of a formula that uses the SUM function with four range arguments:

```
=SUM(A1:A5,C1:C5,E1:E5,G1:G5)
```

You can mix and match the argument types. For example, the following example uses three arguments – a range, followed by a value, and finally an expression.

```
=SUM(A1:A5,12,24*3)
```

You can create function procedures that have an indefinite number of arguments. The trick is to use an array as the last (or only) argument, preceded by the keyword *ParamArray*.



ParamArray can apply only to the *last* argument in the procedure. It is always a variant data type, and it is always an optional argument (although you don't use the Optional keyword).

A SIMPLE EXAMPLE OF INDEFINITE ARGUMENTS

The following is a Function procedure that can have any number of single-value arguments. It simply returns the sum of the arguments.

```
Function SIMPLESUM(ParamArray arglist() As Variant) As Double
    Dim arg as Variant
    For Each arg In arglist
        SIMPLESUM = SIMPLESUM + arg
    Next arg
End Function
```

The following formula returns the sum of the single-cell arguments:

```
=SIMPLESUM(A1,A5,12)
```

The most serious limitation of the SIMPLESUM function is that it does not handle multicell ranges. This improved version does:

```
Function SIMPLESUM(ParamArray arglist() As Variant) As Double
    Dim arg as Variant
    Dim cell as Range
    For Each arg In arglist
        If TypeName(arg) = "Range" Then
            For Each cell In arg
                SIMPLESUM = SIMPLESUM + cell
            Next cell
        Else
            SIMPLESUM = SIMPLESUM + arg
        End If
    Next arg
End Function
```

This function checks each entry in the Arglist array. If the entry is a range, then the code uses a For Each-Next loop to sum the cells in the range.

Even this improved version is certainly no substitute for Excel's SUM function. Try it out by using various types of arguments and you'll see that it fails unless each argument is a value or a range reference. Also, if an argument consists of an entire column, you'll find that the function is *very* slow because it evaluates every cell – even the empty ones.

EMULATING EXCEL'S SUM FUNCTION

This section presents a Function procedure called "MYSUM." Unlike the SIMPLESUM function listed in the previous section, MYSUM emulates Excel's SUM function perfectly.

Before you look at the code for the MYSUM function, take a minute to think about Excel's SUM function. This very versatile function can have any number of arguments (even "missing" arguments), and the arguments can be numerical values, cells, ranges, text representations of numbers, logical values, and even embedded functions. For example, consider the following formula:

```
=SUM(A1,5,"6",,TRUE,SQRT(4),B1:B5)
```

This formula – which is a valid formula – contains all of the following types of arguments, listed here in the order of their presentation:

- ◆ A single cell reference (A1)
- ◆ A literal value (5)
- ◆ A string that looks like a value ("6")
- ◆ A missing argument

- ◆ A logical value (TRUE)
- ◆ An expression that uses another function (SQRT)
- ◆ A range reference (B1:B5)

The following is the listing for the MYSUM function that handles all of these argument types.

```
Function MySum(ParamArray args() As Variant) As Variant
' Emulates Excel's SUM function

' Variable declarations
Dim i As Variant
Dim TempRange As Range, cell As Range
Dim ECode As String
MySum = 0

' Process each argument
For i = 0 To UBound(args)
' Skip missing arguments
If Not IsMissing(args(i)) Then
' What type of argument is it?
Select Case TypeName(args(i))
Case "Range"
' Create temp range to handle full row or column ranges
Set TempRange = _
Intersect(args(i).Parent.UsedRange, args(i))
For Each cell In TempRange
If IsError(cell) Then
MySum = cell ' return the error
Exit Function
End If
If cell = True Or cell = False Then
MySum = MySum + 0
Else
If IsNumeric(cell) Or IsDate(cell) Then _
MySum = MySum + cell
End If
Next cell
Case "Null" 'ignore it
Case "Error" 'return the error
MySum = args(i)
Exit Function
Case "Boolean"
' Check for literal TRUE and compensate
```

```
    If args(i) = "True" Then MySum = MySum + 1
  Case "Date"
    MySum = MySum + args(i)
  Case Else
    MySum = MySum + args(i)
End Select
End If
Next i
End Function
```



A workbook containing the MYSUM function is available on the companion CD-ROM.

As you study the code for MYSUM, keep the following points in mind:

- ◆ Missing arguments (determined by the IsMissing function) are simply ignored.
- ◆ The procedure uses VBA's TypeName function to determine the type of argument (Range, Error, or something else). Each argument type is handled differently.
- ◆ For a range argument, the function loops through each cell in the range and adds its value to a running total.
- ◆ The data type for the function is variant because the function needs to return an error if any of its arguments is an error value.
- ◆ If an argument contains an error (for example, #DIV0!), the MYSUM function simply returns the error—just like Excel's SUM function.
- ◆ Excel's SUM function considers a text string to have a value of 0 unless it appears as a literal argument (that is, as an actual value, not a variable). Therefore, MySum adds the cell's value only if it can be evaluated as a number (VBA's IsNumeric function is used for this).
- ◆ Dealing with Boolean arguments is tricky. For MySum to emulate SUM exactly, it needs to test for a literal TRUE in the argument list and compensate for the difference (that is, add 2 to -1 to get 1).
- ◆ For range arguments, the function uses the Intersect method to create a temporary range that consists of the intersection of the range and the sheet's used range. This handles cases in which a range argument consists of a complete row or column, which would take forever to evaluate.

You may be curious about the relative speeds of `SUM` and `MySum`. `MySum`, of course, is much slower, but just how much slower depends on the speed of your system and the formulas themselves. On my system, a worksheet with 1,000 `SUM` formulas recalculated instantly. After I replaced the `SUM` functions with `MySum` functions, it took about 12 seconds. `MySum` may be improved a bit, but it can never come close to `SUM`'s speed.

By the way, I hope you understand that the point of this example is not to create a new `SUM` function. Rather, it demonstrates how to create custom worksheet functions that look and work like those built into Excel.

Summary

This chapter presented many examples of custom VBA Function procedures that you can use in your worksheet formulas. You can use many of these functions as is. You may need to adapt others to suit your specific needs.

Appendix A

Working with Imported 1-2-3 Files

LOTUS 1-2-3 USED TO be the leading spreadsheet. That distinction, of course, now belongs to Excel, which commands more than 90 percent of the spreadsheet market. Some users, of course, continue to use 1-2-3, so you may be in a position in which you need to import a file generated by 1-2-3. If so, the information in this appendix may be helpful to you.

About 1-2-3 Files

Many versions of 1-2-3 have surfaced over the years, and 1-2-3 files exist in several formats. Table A-1 describes the 1-2-3 files you may encounter.

TABLE A-1 LOTUS 1-2-3 FILE TYPES

File Extension	Description
WKS	Generated by 1-2-3 for DOS Release 1.0 and 1.0a. These files consist of a single sheet. Excel can read and write these files.
WK1	Generated by 1-2-3 for DOS Release 2.x. These files consist of a single sheet, and may have a companion *.FMT or *.ALL file that contains formatting information. Excel can read these files, but saves only the active sheet.
WK3	Generated by 1-2-3 for DOS Release 3.x and 1-2-3 for Windows Release 1.0. These files may contain multiple sheets, and may have a companion *.FM3 file that contains formatting information. Excel can read and write these files.
WK4	Generated by 1-2-3 for Windows Release 4.0. These files may contain multiple sheets. Excel can read and write these files.
123	Generated by 1-2-3 for Windows Release 5 and Millenium Edition. Excel can neither read nor write these files.

Got a Case of File Bloat?

When you import a 1-2-3 file and save it as an Excel file, you may find that the file becomes very large, making it very slow to open and save. The most likely cause is that the imported 1-2-3 file contains entire columns that are preformatted. When Excel imports such a file, it converts all formatted cells — even if they're empty. The solution is to select all blank rows below the last used cell in your worksheet, then delete those rows. Resave the workbook and it should be a more manageable size.

When importing or exporting 1-2-3 files, do not expect a perfect translation. Excel's online help describes the limitations.



Excel supports file links to 1-2-3 workbooks. However, this feature is limited to WKS, WK1, WK3, and WK4 files (not the more recent 123 file format). In some cases, you may need to explicitly update the links. To do so, use Edit → Links, and click the Update Now button.

Lotus 1-2-3 Formulas

In some cases, you may find that the formulas in an imported 1-2-3 file work perfectly in Excel. In other cases, some formulas may not convert correctly and you may need to do some tweaking or rewriting.

Excel evaluates some formulas differently than 1-2-3. These formulas fall into three categories:

- ◆ Those that use text in calculation
- ◆ Those that use logical value (TRUE and FALSE)
- ◆ Those that use database criteria

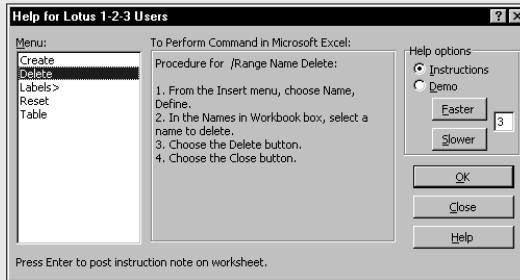
To force Excel to use 1-2-3's method of evaluating formulas, select Tools → Options. In the Options dialog box, click the Transition tab and place a check mark next to the Transition formula evaluation option.



When you open a 1-2-3 file, the Transition formula evaluation check box is selected automatically for that sheet to ensure that Excel calculates the formulas according to Lotus 1-2-3 rules.

Let Excel Teach You

If you're moving up from an older DOS version of 1-2-3, you may be surprised to know that Excel can help you with the transition. The secret lies in the Help → Lotus 1-2-3 Help command. Selecting this command displays a dialog box with the 1-2-3 commands listed along the left side (see the accompanying figure).



Select the 1-2-3 command sequence, and Excel displays instructions, or even demonstrates the corresponding menu command. For example, if you're a veteran 1-2-3 for DOS user, you know that you use /rmd (for Range Name Delete) to delete a name. If you enter this command sequence in Excel's Help for Lotus 1-2-3 Users dialog box, you see instructions that describe how to perform that operation in Excel.

If you plan to make extensive use of an imported 1-2-3 file, you might want to consider translating any formulas that aren't evaluated correctly and turning off the Transition formula evaluation option. Doing so helps to avoid confusion among users unfamiliar with 1-2-3.

The following sections provide some tips on how to convert your 1-2-3 formulas so they work properly in Excel (without the Transition formula evaluation setting).

Calculation Order

Excel and 1-2-3 differ in how they calculate formulas. When calculating a formula, 1-2-3 evaluates the exponentiation operator (^) before the negation operator (-). In Excel, this order is reversed. Consider the following formula:

=-3^2

1-2-3 returns the value -9, but Excel returns +9. To obtain the same results as in 1-2-3, you need to change the formula to:

=-(3^2)

Text in Calculations

In 1-2-3, cells that contain text are considered to have a value of 0 when the cell is used in a formula that uses mathematical operators. Excel, on the other hand, returns an error.



If the Transition formula evaluation option is set, Excel considers text to have a value of 0.

The following formula is perfectly valid in 1-2-3 (and it returns 12). In Excel, the formula returns a #VALUE! error.

```
= "Dog" + 12
```

Similarly, if cell A1 contains the text *Dog*, and cell A2 contains the value 12, the following formula is valid in 1-2-3, but returns an error in Excel:

```
= A1 + A2
```

Excel, however, does permit references to text cells in function arguments, and it ignores such references. For example, the following formula works fine in both 1-2-3 and Excel, even if the range A1:A10 contains text:

```
= SUM(A1:A10)
```

You can take advantage of this fact to convert a 1-2-3 formula such as =A1+A2 to the following:

```
= SUM(A1, A2)
```

Logical Values

Boolean expressions in 1-2-3 are evaluated to 1 or 0. Excel displays these values as TRUE or FALSE. TRUE is equivalent to 1-2-3's 1, and FALSE is equivalent to 1-2-3's 0.



If the Transition formula evaluation option is set, Excel displays 0 for FALSE and 1 for TRUE.

In 1-2-3, for example, the following formula displays either 1 or 0, depending on the contents of cells A1 and A2. In Excel, the formula returns either TRUE or FALSE.

```
=A1<A2
```

This distinction may be important if your worksheet uses IF functions that check for 0 or 1. For example, the following formula has different results in 1-2-3 and Excel:

```
=IF(A1<A2=1,B1,B2)
```

To fix this formula so it works properly in Excel, change it to:

```
=IF(A1<A2,B1,B2)
```

Lotus 1-2-3 uses the following logical operators: #AND#, #NOT#, and #OR#. Excel uses logical functions (AND, NOT, and OR) in place of these. For example, the following 1-2-3 formula returns the string *yes* if cell A1=12 and cell A2=12, and the string *no* if both cells are not equal to 12:

```
@IF(A1=12#AND#A2=12,"yes","no")
```

The equivalent 1-2-3 formula is:

```
=IF(AND(A1=12,A2=12),"yes","no")
```

Date Problems

If the imported 1-2-3 worksheet contains dates that use a hyphen (for example, 12-31-98), these dates may not be recognized by Excel. For example, Excel will interpret the cell as a formula:

```
=12-31-98
```

These cells will need to be edited so they display properly.

Database Criteria

If your imported worksheet uses database criteria ranges (for example, advanced filtering), be especially careful. Database criteria ranges are evaluated differently when you extract data, find data, and use database functions. For example, the criteria “Ben” finds only rows where the value *Ben* is contained in the cell. In Excel, the criteria “Ben” finds rows in which the contents of the cell begins with *Ben*—including *Benjamin*, *Benny*, and *Benito*.

What About 1-2-3 Macros?

Excel can execute some 1-2-3 macros – the keystroke macros developed using early versions of 1-2-3. These macros are stored directly in a worksheet and represent keystrokes sent to the interface. Don't expect perfect compatibility, however.

Typically, these keystroke macros are given a range name such as \t. This macro is executed by typing Ctrl+T. These special names are valid if they are contained in an imported file. But, you'll find that you cannot create such a name in Excel.

If you convert 1-2-3 files to Excel, the best approach is to recreate the macros using VBA. You'll get much better performance and the macros will be easier to maintain.



If the Transition formula evaluation option is set, Excel works exactly like 1-2-3 in using database criteria.

Lotus 1-2-3 Function Compatibility

Most of the worksheet functions in 1-2-3 have equivalents in Excel. In some cases, however, the correspondence is not perfect. Fortunately, Excel's online help provides a thorough description of the differences between the worksheet functions available in 1-2-3 and in Excel.

Function Equivalents

Table A-2 lists 1-2-3 functions that have equivalent Excel functions. It's important to understand that in some cases the correspondence is not exact. Also, for some Excel functions, you must enter the arguments in a different order.

TABLE A-2 EXCEL EQUIVALENTS FOR 1-2-3 FUNCTIONS

Lotus 1-2-3 Function	Equivalent Excel Function
@	INDIRECT
@@	INDIRECT
@ABS	ABS
@ACCRUED	ACCRINT

Lotus 1-2-3 Function	Equivalent Excel Function
@ACOS	ACOS
@ACOSH	ACOSH
@ASIN	ASIN
@ASINH	ASINH
@ATAN	ATAN
@ATAN2	ATAN2
@ATANH	ATANH
@AVEDEV	AVEDEV
@AVG	AVERAGE, AVERAGEA
@BESSELI	BESSELI
@BESSELJ	BESSELJ
@BESSELK	BESSELK
@BESSELY	BESSELY
@BIN2DEC	BIN2DEC
@BIN2HEX	BIN2HEX
@BIN2OCT	BIN2OCT
@BINOMIAL	BINOMDIST
@CELL	CELL
@CELLPOINTER	CELL
@CHAR	CHAR
@CHIDIST	CHIINV, CHIDIST
@CHOOSE	CHOOSE
@CLEAN	CLEAN
@CODE	CODE
@COLS	COLUMNS
@COLUMN	COLUMN
@COMBIN	COMBIN

Continued

TABLE A-2 EXCEL EQUIVALENTS FOR 1-2-3 FUNCTIONS *(Continued)*

Lotus 1-2-3 Function	Equivalent Excel Function
@CONFIDENCE	CONFIDENCE
@CONVERT	CONVERT
@CORREL	CORREL, PEARSON
@COS	COS
@COSH	COSH
@COUNT	COUNTA
@COUNTBLANK	COUNTBLANK
@COUNTIF	COUNTIF
@COUPDAYBS	COUPDAYBS
@COUPDAYS	COUPDAYS
@COUPDAYSNC	COUPDAYSNC
@COUPNCD	COUPNCD
@COUPNUM	COUPNUM
@COUPPCD	COUPPCD
@COV	COVAR
@CRITBINOMIAL	CRITBINOM
@CTERM	NPER
@D360	DAYS360
@DATE	DATE
@DATEDIF	DATEDIF
@DATEVALUE	DATEVALUE
@DAVG	DAVERAGE
@DAY	DAY
@DAYS	DAYS360
@DAYS360	DAYS360
@DB	DB

Lotus 1-2-3 Function	Equivalent Excel Function
@DCOUNT	DCOUNTA
@DDB	DDB
@DEC2BIN	DEC2BIN
@DEC2FRAC	DOLLARFR
@DEC2HEX	DEC2HEX
@DEC2OCT	DEC2OCT
@DEGTORAD	RADIANS
@DEVSQ	DEVSQ
@DGET	DGET
@DISC	DISC
@DMAX	DMAX
@DMIN	DMIN
@DPURECOUNT	DCOUNT, DCOUNTA
@DSTD	DSTDEVP
@DSTDS	DSTDEV
@DSUM	DSUM
@DURATION	DURATION
@DVAR	DVARP
@DVAR	DVAR
@ERF	ERF
@ERFC	ERFC
@EVEN	EVEN
@EXACT	EXACT
@EXP	EXP
@EXPONDIST	EXPONDIST
@FACT	FACT
@FALSE	FALSE
@FDIST	FINV, FDIST

Continued

TABLE A-2 EXCEL EQUIVALENTS FOR 1-2-3 FUNCTIONS *(Continued)*

Lotus 1-2-3 Function	Equivalent Excel Function
@FIND	FIND
@FISHER	FISHER
@FISHERINV	FISHERINV
@FORECAST	FORECAST
@FRAC2DEC	DOLLARDE
@FTEST	FTEST
@FV	FV
@FVAL	FV
@GAMMALN	GAMMALN
@GEOMEAN	GEOMEAN
@HARMEAN	HARMEAN
@HEX2BIN	HEX2BIN
@HEX2DEC	HEX2DEC
@HEX2OCT	HEX2OCT
@HLOOKUP	HLOOKUP
@HOUR	HOUR
@HYPGEOMDIST	HYPGEOMDIST
@IF	IF
@INDEX	INDEX
@INFO	INFO
@INT	TRUNC
@INTRATE	INTRATE
@IPAYMT	CUMIPMT, IMPT
@IRATE	RATE
@IRR	IRR
@ISEMPTY	ISBLANK
@ISERR	ISERR, ISERROR

Lotus 1-2-3 Function	Equivalent Excel Function
@ISNUMBER	ISNONTEXT, ISNUMBER
@ISRANGE	ISREF
@ISSTRING	ISTEXT
@KURTOSIS	KURT
@LARGE	LARGE
@LEFT	LEFT
@LENGTH	LEN
@LN	LN
@LOG	LOG, LOG10
@LOGINV	LOGINV
@LOGNORMDIST	LOGNORMDIST
@LOWER	LOWER
@MATCH	MATCH
@MAX	MAXA
@MDURATION	MDURATION
@MEDIAN	MEDIAN
@MID	MID
@MIN	MINA
@MINUTE	MINUTE
@MIRR	MIRR
@MOD	MOD
@MODE	MODE
@MONTH	MONTH
@N	N
@NA	NA
@NEGBINOMDIST	NEGBINOMDIST
@NETWORKDAYS	NETWORKDAYS

Continued

TABLE A-2 EXCEL EQUIVALENTS FOR 1-2-3 FUNCTIONS *(Continued)*

Lotus 1-2-3 Function	Equivalent Excel Function
@NEXTMONTH	EOMONTH, EDATE
@NORMAL	NORMINV, NORMDIST, NORMSDIST
@NORMSINV	NORMSINV
@NOW	NOW
@NPER	NPER
@NPV	NPV
@OCT2BIN	OCT2BIN
@OCT2DEC	OCT2DEC
@OCT2HEX	OCT2HEX
@ODD	ODD
@PAYMT	PMT
@PERCENTILE	PERCENTILE
@PERMUT	PERMUT
@PI	PI
@PMT	PMT
@POISSON	POISSON
@PPAYMT	CUMPRINC, PPMT
@PRANK	PERCENTRANK
@PRICE	PRICE
@PRICEDISC	PRICEDISC
@PRICEMAT	PRICEMAT
@PROB	PROB
@PRODUCT	PRODUCT
@PROPER	PROPER
@PUREAVG	AVERAGE, AVERAGEA
@PURECOUNT	COUNT, COUNTA
@PUREMAX	MAX, MAXA

Lotus 1-2-3 Function	Equivalent Excel Function
@PUREMIN	MIN, MINA
@PURESTD	STDEVP, STDEVPA
@PURESTDS	STDEV, STDEVA
@PUREVAR	VARP, VARPA
@PUREVARS	VAR, VARA
@PV	PV
@PVAL	PV
@QUARTILE	QUARTILE
@QUOTIENT	QUOTIENT
@RADTODEG	DEGREES
@RAND	RAND
@RANDBETWEEN	RANDBETWEEN
@RANK	RANK
@RATE	RATE
@RECEIVED	RECEIVED
@REGRESSION	INTERCEPT
@REPEAT	REPT
@REPLACE	REPLACE
@RIGHT	RIGHT
@ROUND	ROUND
@ROUNDDOWN	INT, ROUNDDOWN
@ROUNDM	CEILING, FLOOR
@ROUNDUP	ROUNDUP
@ROW	ROW
@ROWS	ROWS
@RSQ	RSQ

Continued

TABLE A-2 EXCEL EQUIVALENTS FOR 1-2-3 FUNCTIONS *(Continued)*

Lotus 1-2-3 Function	Equivalent Excel Function
@S	T
@SECOND	SECOND
@SERIESSUM	SERIESSUM
@SIGN	SIGN
@SIN	SIN
@SINH	SINH
@SKEWNESS	SKEW
@SLN	SLN
@SMALL	SMALL
@SQRT	SQRT
@SQRTPI	SQRTPI
@STANDARDIZE	STANDARDIZE
@STD	STDEVPA
@STDS	STDEVA
@STEYX	STEYX
@STRING	FIXED, TEXT
@SUM	SUM
@SUMIF	SUMIF
@SUMPRODUCT	SUMPRODUCT
@SUMSQ	SUMSQ
@SUMX2MY2	SUMX2MY2
@SUMX2PY2	SUMX2PY2
@SUMXMY2	SUMXMY2
@SYD	SYD
@TAN	TAN
@TANH	TANH
@TBILLEQ	TBILLEQ

Lotus 1-2-3 Function	Equivalent Excel Function
@TBILLPRICE	TBILLPRICE
@TBILLYIELD	TBILLYIELD
@TDIST	TDIST, TINV
@TERM	NPER
@TIME	TIME
@TIMEVALUE	TIMEVALUE
@TODAY	TODAY
@TRIM	TRIM
@TRIMMEAN	TRIMMEAN
@TRUE	TRUE
@TRUNC	TRUNC
@TTEST	TTEST
@UPPER	UPPER
@VALUE	VALUE
@VAR	VARPA
@VARS	VARA
@VDB	VDB
@VLOOKUP	VLOOKUP
@WEEKDAY	WEEKDAY
@WEIBULL	WEIBULL
@WORKDAY	WORKDAY
@YEAR	YEAR
@YEARFRAC	YEARFRAC
@YIELD	YIELD
@YIELDDISC	YIELDDISC
@YIELDMAT	YIELDMAT
@ZTEST	ZTEST

Converting Database Functions

There's a common problem with 1-2-3's database functions (for example, @DSUM and @DCOUNT). Lotus 1-2-3 enables you to specify your criteria as an argument. Refer to Figure A-1, which shows a 1-2-3 file imported into Excel. The following formula (in cell E4) was not translated correctly and displays a #NAME? error:

```
=DCOUNTA(A1:C17,"Product",AND(PRODUCT="Widget",MONTH="January"))
```

	A	B	C	D	E	F	G	H
1	Product	Sold By	Month					
2	Widget	Jones	January					
3	Sprocket	Smith	January					
4	Sprocket	Jones	January		#NAME?	Records qualify		
5	Widget	Kendell	January					
6	Widget	Kendell	January					
7	Sprocket	Kendell	January					
8	Widget	Jones	January					
9	Sprocket	Smith	January					
10	Widget	Smith	February					
11	Widget	Jones	February					
12	Sprocket	Kendell	February					
13	Sprocket	Kendell	February					
14	Sprocket	Jones	February					
15	Widget	Smith	February					
16	Widget	Smith	February					
17	Sprocket	Kendell	February					
18								

Figure A-1: This imported 1-2-3 file uses an @DCOUNT function with a criterion argument not supported by Excel.

The original 1-2-3 formula was written to return the count of records in which the Product is *Widget* and the Month is *January*. The original 1-2-3 formula (before conversion by Excel) was:

```
@DCOUNT(A1:C17,"Product",PRODUCT="Widget" #AND#MONTH="January")
```

Unfortunately, Excel's database functions do not allow you to specify the criteria as an argument. Rather, you need to do one of the following:

- ◆ Set up a special criteria range for the DCOUNTA function
- ◆ Use a different function – in this case, the COUNTIF function

Figure A-2 shows a criteria range in E1:F2. The following formula returns the count of records in which the Product is *Widget* and the Month is *January*:

```
=DCOUNTA(A1:C17,"Product",E1:F2)
```

The screenshot shows an Excel spreadsheet window titled 'sales.wk4'. The spreadsheet has columns A through G and rows 1 through 18. Columns A, B, and C contain data for 'Product', 'Sold By', and 'Month' respectively. Columns E and F contain a filtered view of the data. A formula bar at the bottom shows the formula '=DCOUNT(A2:A17, "Widget", C2:C17, "January")' and the result '4 Records qualify' is displayed in cell E5.

	A	B	C	D	E	F	G
1	Product	Sold By	Month		Product	Month	
2	Widget	Jones	January		Widget	January	
3	Sprocket	Smith	January				
4	Sprocket	Jones	January		4 Records qualify		
5	Widget	Kendell	January				
6	Widget	Kendell	January				
7	Sprocket	Kendell	January				
8	Widget	Jones	January				
9	Sprocket	Smith	January				
10	Widget	Smith	February				
11	Widget	Jones	February				
12	Sprocket	Kendell	February				
13	Sprocket	Kendell	February				
14	Sprocket	Jones	February				
15	Widget	Smith	February				
16	Widget	Smith	February				
17	Sprocket	Kendell	February				
18							

Figure A-2: Using a criteria range for the DCOUNT formula

Alternatively, you can use an array formula, which doesn't require a criteria range. The following formula is the Excel equivalent of the incorrectly translated @DCOUNT formula:

```
{=SUM((A2:A17="Widget")*(C2:C17="January"))}
```



Enter an array formula using Ctrl+Shift+Enter. Do not type the brackets; Excel inserts them for you. For more information about array formulas, see Part III. For additional counting techniques, see Chapter 7.

Appendix B

Excel Function Reference

THIS APPENDIX CONTAINS A complete listing of Excel's worksheet functions. The functions are arranged alphabetically in tables by categories used by the Insert Function dialog box.

Excel Functions by Category

Tables B-1 through B-10 present the following 10 categories of Excel functions: database, date and time, engineering, financial, information, logical, lookup and reference, math and trig, statistical, and text. Some of these functions are available only if you install the Analysis ToolPak add-in program.

For more information about a particular function, including its arguments, select the function in the Insert Function dialog box and press F1.

Table B-1 lists Excel's database functions.

TABLE B-1 DATABASE CATEGORY FUNCTIONS

Function	What It Does
DAVERAGE	Returns the average of selected database entries
DCOUNT	Counts the cells containing numbers from a specified database and criteria
DCOUNTA	Counts nonblank cells from a specified database and criteria
DGET	Extracts from a database a single record that matches the specified criteria
DMAX	Returns the maximum value from selected database entries
DMIN	Returns the minimum value from selected database entries
DPRODUCT	Multiplies the values in a particular field of records that match the criteria in a database
DSTDEV	Estimates the standard deviation based on a sample of selected database entries
DSTDEVP	Calculates the standard deviation based on the entire population of selected database entries

Continued

TABLE B-1 DATABASE CATEGORY FUNCTIONS *(Continued)*

Function	What It Does
DSUM	Adds the numbers in the field column of records in the database that match the criteria
DVAR	Estimates variance based on a sample from selected database entries
DVARP	Calculates variance based on the entire population of selected database entries

Table B-2 lists Excel's date and time functions.

TABLE B-2 DATE AND TIME CATEGORY FUNCTIONS

Function	What It Does
DATE	Returns the serial number of a particular date
DATEDIF	Calculates the number of days, months, or years between two dates. This function is documented only in Excel 2000.
DATEVALUE	Converts a date in the form of text to a serial number
DAY	Converts a serial number to a day of the month
DAYS360	Calculates the number of days between two dates, based on a 360-day year
EDATE*	Returns the serial number of the date that is the indicated number of months before or after the start date
EOMONTH*	Returns the serial number of the last day of the month before or after a specified number of months
HOUR	Converts a serial number to an hour
MINUTE	Converts a serial number to a minute
MONTH	Converts a serial number to a month
NETWORKDAYS*	Returns the number of whole workdays between two dates
NOW	Returns the serial number of the current date and time
SECOND	Converts a serial number to a second
TIME	Returns the serial number of a particular time
TIMEVALUE	Converts a time in the form of text to a serial number

Function	What It Does
TODAY	Returns the serial number of today's date
WEEKDAY	Converts a serial number to a day of the week
WEEKNUM*	Returns the week number in the year
WORKDAY*	Returns the serial number of the date before or after a specified number of workdays
YEAR	Converts a serial number to a year
YEARFRAC*	Returns the year fraction representing the number of whole days between start_date and end_date

*Available only when the Analysis ToolPak add-in is installed.

Table B-3 lists Excel's engineering functions.

TABLE B-3 ENGINEERING CATEGORY FUNCTIONS AVAILABLE IN THE ANALYSIS TOOLPAK ADD-IN

Function	What It Does
BESSELI	Returns the modified Bessel function $I_n(x)$
BESSELJ	Returns the Bessel function $J_n(x)$
BESSELK	Returns the modified Bessel function $K_n(x)$
BESSELY	Returns the Bessel function $Y_n(x)$
BIN2DEC	Converts a binary number to decimal
BIN2HEX	Converts a binary number to hexadecimal
BIN2OCT	Converts a binary number to octal
COMPLEX	Converts real and imaginary coefficients into complex numbers
CONVERT	Converts a number from one measurement system to another
DEC2BIN	Converts a decimal number to binary
DEC2HEX	Converts a decimal number to hexadecimal
DEC2OCT	Converts a decimal number to octal
DELTA	Tests whether two values are equal

Continued

**TABLE B-3 ENGINEERING CATEGORY FUNCTIONS AVAILABLE IN THE ANALYSIS
TOOLPAK ADD-IN** *(Continued)*

Function	What It Does
ERF	Returns the error function
ERFC	Returns the complementary error function
FACTDOUBLE	Returns the double factorial of a number
GESTEP	Tests whether a number is greater than a threshold value
HEX2BIN	Converts a hexadecimal number to binary
HEX2DEC	Converts a hexadecimal number to decimal
HEX2OCT	Converts a hexadecimal number to octal
IMABS	Returns the absolute value (modulus) of a complex number
IMAGINARY	Returns the imaginary coefficient of a complex number
IMARGUMENT	Returns the argument theta – an angle expressed in radians
IMCONJUGATE	Returns the complex conjugate of a complex number
IMCOS	Returns the cosine of a complex number
IMDIV	Returns the quotient of two complex numbers
IMEXP	Returns the exponential of a complex number
IMLN	Returns the natural logarithm of a complex number
IMLOG10	Returns the base-10 logarithm of a complex number
IMLOG2	Returns the base-2 logarithm of a complex number
IMPOWER	Returns a complex number raised to an integer power
IMPRODUCT	Returns the product of two complex numbers
IMREAL	Returns the real coefficient of a complex number
IMSIN	Returns the sine of a complex number
IMSQRT	Returns the square root of a complex number
IMSUB	Returns the difference of two complex numbers
IMSUM	Returns the sum of complex numbers
OCT2BIN	Converts an octal number to binary
OCT2DEC	Converts an octal number to decimal
OCT2HEX	Converts an octal number to hexadecimal

Table B-4 lists Excel's financial functions.

TABLE B-4 FINANCIAL CATEGORY FUNCTIONS

Function	What It Does
ACCRINT*	Returns the accrued interest for a security that pays periodic interest
ACCRINTM*	Returns the accrued interest for a security that pays interest at maturity
AMORDEGRC*	Returns the depreciation for each accounting period
AMORLINC*	Returns the depreciation for each accounting period
COUPDAYBS*	Returns the number of days from the beginning of the coupon period to the settlement date
COUPDAYS*	Returns the number of days in the coupon period that contains the settlement date
COUPDAYSNC*	Returns the number of days from the settlement date to the next coupon date
COUPNCD*	Returns the next coupon date after the settlement date
COUPNUM*	Returns the number of coupons payable between the settlement date and maturity date
COUPPCD*	Returns the previous coupon date before the settlement date
CUMIPMT*	Returns the cumulative interest paid between two periods
CUMPRINC*	Returns the cumulative principal paid on a loan between two periods
DB	Returns the depreciation of an asset for a specified period, using the fixed-declining balance method
DDB	Returns the depreciation of an asset for a specified period, using the double-declining balance method or some other method that you specify
DISC*	Returns the discount rate for a security
DOLLARDE*	Converts a dollar price (expressed as a fraction) into a dollar price (expressed as a decimal number)
DOLLARFR*	Converts a dollar price (expressed as a decimal number) into a dollar price (expressed as a fraction)
DURATION*	Returns the annual duration of a security with periodic interest payments

Continued

TABLE B-4 FINANCIAL CATEGORY FUNCTIONS (Continued)

Function	What It Does
EFFECT*	Returns the effective annual interest rate
FV	Returns the future value of an investment
FVSCHEDULE*	Returns the future value of an initial principal after applying a series of compound interest rates
INTRATE*	Returns the interest rate for a fully invested security
IPMT	Returns the interest payment for an investment for a given period
IRR	Returns the internal rate of return for a series of cash flows
ISPMT	Returns the interest associated with a specific loan payment
MDURATION*	Returns the Macauley modified duration for a security with an assumed par value of \$100
MIRR	Returns the internal rate of return where positive and negative cash flows are financed at different rates
NOMINAL*	Returns the annual nominal interest rate
NPER	Returns the number of periods for an investment
NPV	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate
ODDFPRICE*	Returns the price per \$100 face value of a security with an odd first period
ODDFYIELD*	Returns the yield of a security with an odd first period
ODDLPRICE*	Returns the price per \$100 face value of a security with an odd last period
ODDLYIELD*	Returns the yield of a security with an odd last period
PMT	Returns the periodic payment for an annuity
PPMT	Returns the payment on the principal for an investment for a given period
PRICE*	Returns the price per \$100 face value of a security that pays periodic interest
PRICEDISC*	Returns the price per \$100 face value of a discounted security
PRICEMAT*	Returns the price per \$100 face value of a security that pays interest at maturity

Function	What It Does
PV	Returns the present value of an investment
RATE	Returns the interest rate per period of an annuity
RECEIVED*	Returns the amount received at maturity for a fully invested security
SLN	Returns the straight-line depreciation of an asset for one period
SYD	Returns the sum-of-years' digits depreciation of an asset for a specified period
TBILLEQ*	Returns the bond-equivalent yield for a Treasury bill
TBILLPRICE*	Returns the price per \$100 face value for a Treasury bill
TBILLYIELD*	Returns the yield for a Treasury bill
VDB	Returns the depreciation of an asset for a specified or partial period using a declining balance method
XIRR*	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic
XNPV*	Returns the net present value for a schedule of cash flows that is not necessarily periodic
YIELD*	Returns the yield on a security that pays periodic interest
YIELDDISC*	Returns the annual yield for a discounted security; for example, a Treasury bill
YIELDMAT*	Returns the annual yield of a security that pays interest at maturity

**Available only when the Analysis ToolPak add-in is installed.*

Table B-5 lists Excel's information functions.

TABLE B-5 INFORMATION CATEGORY FUNCTIONS

Function	What It Does
CELL	Returns information about the formatting, location, or contents of a cell
ERROR.TYPE	Returns a number corresponding to an error type
INFO	Returns information about the current operating environment

Continued

TABLE B-5 INFORMATION CATEGORY FUNCTIONS *(Continued)*

Function	What It Does
ISBLANK	Returns TRUE if the value is blank
ISERR	Returns TRUE if the value is any error value except #N/A
ISERROR	Returns TRUE if the value is any error value
ISEVEN*	Returns TRUE if the number is even
ISLOGICAL	Returns TRUE if the value is a logical value
ISNA	Returns TRUE if the value is the #N/A error value
ISNONTEXT	Returns TRUE if the value is not text
ISNUMBER	Returns TRUE if the value is a number
ISODD*	Returns TRUE if the number is odd
ISREF	Returns TRUE if the value is a reference
ISTEXT	Returns TRUE if the value is text
N	Returns a value converted to a number
NA	Returns the error value #N/A
TYPE	Returns a number indicating the data type of a value

**Available only when the Analysis ToolPak add-in is installed.*

Table B-6 lists Excel's logical functions.

TABLE B-6 LOGICAL CATEGORY FUNCTIONS

Function	What It Does
AND	Returns TRUE if all of its arguments are TRUE
FALSE	Returns the logical value FALSE
IF	Specifies a logical test to perform
NOT	Reverses the logic of its argument
OR	Returns TRUE if any argument is TRUE
TRUE	Returns the logical value TRUE

Table B-7 lists Excel's lookup and reference functions.

TABLE B-7 LOOKUP AND REFERENCE CATEGORY FUNCTIONS

Function	What It Does
ADDRESS	Returns a reference as text to a single cell in a worksheet
AREAS	Returns the number of areas in a reference
CHOOSE	Chooses a value from a list of values
COLUMN	Returns the column number of a reference
COLUMNS	Returns the number of columns in a reference
GETPIVOTDATA	Returns data stored in a pivot table
HLOOKUP	Looks in the top row of an array and returns the value of the indicated cell
HYPERLINK	Creates a shortcut that opens a document on your hard drive, a server, or the Internet
INDEX	Uses an index to choose a value from a reference or array
INDIRECT	Returns a reference indicated by a text value
LOOKUP	Looks up values in a vector or array
MATCH	Looks up values in a reference or array
OFFSET	Returns a reference offset from a given reference
ROW	Returns the row number of a reference
ROWS	Returns the number of rows in a reference
RTD*	Retrieves real-time data from a program that supports COM automation
TRANSPOSE	Returns the transpose of an array
VLOOKUP	Looks in the first column of an array and moves across the row to return the value of a cell

**This function is available only in Excel 2002.*

Table B-8 lists Excel's mathematical and trigonometric functions.

TABLE B-8 MATH AND TRIG CATEGORY FUNCTIONS

Function	What It Does
ABS	Returns the absolute value of a number
ACOS	Returns the arccosine of a number
ACOSH	Returns the inverse hyperbolic cosine of a number
ASIN	Returns the arcsine of a number
ASINH	Returns the inverse hyperbolic sine of a number
ATAN	Returns the arctangent of a number
ATAN2	Returns the arctangent from x and y coordinates
ATANH	Returns the inverse hyperbolic tangent of a number
CEILING	Rounds a number to the nearest integer or to the nearest multiple of significance
COMBIN	Returns the number of combinations for a given number of objects
COS	Returns the cosine of a number
COSH	Returns the hyperbolic cosine of a number
DEGREES	Converts radians to degrees
EVEN	Rounds a number up to the nearest even integer
EXP	Returns e raised to the power of a given number
FACT	Returns the factorial of a number
FLOOR	Rounds a number down, toward 0
GCD*	Returns the greatest common divisor
INT	Rounds a number down to the nearest integer
LCM*	Returns the least common multiple
LN	Returns the natural logarithm of a number
LOG	Returns the logarithm of a number to a specified base
LOG10	Returns the base-10 logarithm of a number
MDETERM	Returns the matrix determinant of an array
MINVERSE	Returns the matrix inverse of an array

Function	What It Does
MMULT	Returns the matrix product of two arrays
MOD	Returns the remainder from division
MROUND*	Returns a number rounded to the desired multiple
MULTINOMIAL*	Returns the multinomial of a set of numbers
ODD	Rounds a number up to the nearest odd integer
PI	Returns the value of pi
POWER	Returns the result of a number raised to a power
PRODUCT	Multiplies its arguments
QUOTIENT*	Returns the integer portion of a division
RADIANS	Converts degrees to radians
RAND	Returns a random number between 0 and 1
RANDBETWEEN*	Returns a random number between the numbers that you specify
ROMAN	Converts an Arabic number to Roman, as text
ROUND	Rounds a number to a specified number of digits
ROUNDDOWN	Rounds a number down, toward 0
ROUNDUP	Rounds a number up, away from 0
SERIESSUM*	Returns the sum of a power series based on the formula
SIGN	Returns the sign of a number
SIN	Returns the sine of the given angle
SINH	Returns the hyperbolic sine of a number
SQRT	Returns a positive square root
SQRTPI*	Returns the square root of (<i>number</i> * π)
SUBTOTAL	Returns a subtotal in a list or database
SUM	Adds its arguments
SUMIF	Adds the cells specified by a given criteria
SUMPRODUCT	Returns the sum of the products of corresponding array components

Continued

TABLE B-8 MATH AND TRIG CATEGORY FUNCTIONS (Continued)

Function	What It Does
SUMSQ	Returns the sum of the squares of the arguments
SUMX2MY2	Returns the sum of the difference of squares of corresponding values in two arrays
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two arrays
SUMXMY2	Returns the sum of squares of differences of corresponding values in two arrays
TAN	Returns the tangent of a number
TANH	Returns the hyperbolic tangent of a number
TRUNC	Truncates a number to an integer

**Available only when the Analysis ToolPak add-in is installed.*

Table B-9 lists Excel's statistical functions.

TABLE B-9 STATISTICAL CATEGORY FUNCTIONS

Function	What It Does
AVEDEV	Returns the average of the absolute deviations of data points from their mean
AVERAGE	Returns the average of its arguments
AVERAGEA	Returns the average of its arguments and includes evaluation of text and logical values
BETADIST	Returns the cumulative beta probability density function
BETAINV	Returns the inverse of the cumulative beta probability density function
BINOMDIST	Returns the individual term binomial distribution probability
CHIDIST	Returns the one-tailed probability of the chi-squared distribution
CHIINV	Returns the inverse of the one-tailed probability of the chi-squared distribution
CHITEST	Returns the test for independence

Function	What It Does
CONFIDENCE	Returns the confidence interval for a population mean
CORREL	Returns the correlation coefficient between two data sets
COUNT	Counts how many numbers are in the list of arguments
COUNTA	Counts how many values are in the list of arguments
COUNTBLANK	Counts the number of blank cells in the argument range
COUNTIF	Counts the number of cells that meet the criteria you specify in the argument
COVAR	Returns covariance – the average of the products of paired deviations
CRITBINOM	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value
DEVSQ	Returns the sum of squares of deviations
EXPONDIST	Returns the exponential distribution
FDIST	Returns the F probability distribution
FINV	Returns the inverse of the F probability distribution
FISHER	Returns the Fisher transformation
FISHERINV	Returns the inverse of the Fisher transformation
FORECAST	Returns a value along a linear trend
FREQUENCY	Returns a frequency distribution as a vertical array
FTEST	Returns the result of an F-test
GAMMADIST	Returns the gamma distribution
GAMMAINV	Returns the inverse of the gamma cumulative distribution
GAMMALN	Returns the natural logarithm of the gamma function, $G(x)$
GEOMEAN	Returns the geometric mean
GROWTH	Returns values along an exponential trend
HARMEAN	Returns the harmonic mean
HYPGEOMDIST	Returns the hypergeometric distribution
INTERCEPT	Returns the intercept of the linear regression line

Continued

TABLE B-9 STATISTICAL CATEGORY FUNCTIONS (Continued)

Function	What It Does
KURT	Returns the kurtosis of a data set
LARGE	Returns the <i>k</i> th largest value in a data set
LINEST	Returns the parameters of a linear trend
LOGEST	Returns the parameters of an exponential trend
LOGINV	Returns the inverse of the lognormal distribution
LOGNORMDIST	Returns the cumulative lognormal distribution
MAX	Returns the maximum value in a list of arguments, ignoring logical values and text
MAXA	Returns the maximum value in a list of arguments, including logical values and text
MEDIAN	Returns the median of the given numbers
MIN	Returns the minimum value in a list of arguments, ignoring logical values and text
MINA	Returns the minimum value in a list of arguments, including logical values and text
MODE	Returns the most common value in a data set
NEGBINOMDIST	Returns the negative binomial distribution
NORMDIST	Returns the normal cumulative distribution
NORMINV	Returns the inverse of the normal cumulative distribution
NORMSDIST	Returns the standard normal cumulative distribution
NORMSINV	Returns the inverse of the standard normal cumulative distribution
PEARSON	Returns the Pearson product moment correlation coefficient
PERCENTILE	Returns the <i>k</i> th percentile of values in a range
PERCENTRANK	Returns the percentage rank of a value in a data set
PERMUT	Returns the number of permutations for a given number of objects
POISSON	Returns the Poisson distribution
PROB	Returns the probability that values in a range are between two limits
QUARTILE	Returns the quartile of a data set
RANK	Returns the rank of a number in a list of numbers

Function	What It Does
RSQ	Returns the square of the Pearson product moment correlation coefficient
SKEW	Returns the skewness of a distribution
SLOPE	Returns the slope of the linear regression line
SMALL	Returns the <i>k</i> th smallest value in a data set
STANDARDIZE	Returns a normalized value
STDEV	Estimates standard deviation based on a sample, ignoring text and logical values
STDEVA	Estimates standard deviation based on a sample, including text and logical values
STDEVP	Calculates standard deviation based on the entire population, ignoring text and logical values
STDEVPA	Calculates standard deviation based on the entire population, including text and logical values
STEYX	Returns the standard error of the predicted <i>y</i> -value for each <i>x</i> in the regression
TDIST	Returns the student's <i>t</i> -distribution
TINV	Returns the inverse of the student's <i>t</i> -distribution
TREND	Returns values along a linear trend
TRIMMEAN	Returns the mean of the interior of a data set
TTEST	Returns the probability associated with a student's <i>t</i> -test
VAR	Estimates variance based on a sample, ignoring logical values and text
VARA	Estimates variance based on a sample, including logical values and text
VARP	Calculates variance based on the entire population, ignoring logical values and text
VARPA	Calculates variance based on the entire population, including logical values and text
WEIBULL	Returns the Weibull distribution
ZTEST	Returns the two-tailed <i>P</i> -value of a <i>z</i> -test

Table B-10 lists Excel's text functions.

TABLE B-10 TEXT CATEGORY FUNCTIONS

Function	What It Does
CHAR	Returns the character specified by the code number
CLEAN	Removes all nonprintable characters from text
CODE	Returns a numeric code for the first character in a text string
CONCATENATE	Joins several text items into one text item
DOLLAR	Converts a number to text, using currency format
EXACT	Checks to see whether two text values are identical
FIND	Finds one text value within another (case-sensitive)
FIXED	Formats a number as text with a fixed number of decimals
LEFT	Returns the leftmost characters from a text value
LEN	Returns the number of characters in a text string
LOWER	Converts text to lowercase
MID	Returns a specific number of characters from a text string, starting at the position that you specify
PROPER	Capitalizes the first letter in each word of a text value
REPLACE	Replaces characters within text
REPT	Repeats text a given number of times
RIGHT	Returns the rightmost characters from a text value
SEARCH	Finds one text value within another (not case-sensitive)
SUBSTITUTE	Substitutes new text for old text in a text string
T	Converts its arguments to text
TEXT	Formats a number and converts it to text
TRIM	Removes spaces from text
UPPER	Converts text to uppercase
VALUE	Converts a text argument to a number

Appendix C

Using Custom Number Formats

THE ABILITY TO CREATE custom number formats is one of Excel's most powerful features. Although Excel provides a good variety of built-in number formats, you may find that none of these suit your needs. This appendix describes how to create custom number formats and provides many examples.

About Number Formatting

By default, all cells use the General number format. This is basically a “what you type is what you get” format. But if the cell is not wide enough to show the entire number, the General format rounds numbers with decimals and uses scientific notation for large numbers. In many cases, you may want to format a cell using something other than the General number format.

The key thing to remember about number formatting is that it affects only how a value is *displayed*. The actual number remains intact, and any formulas that use a formatted number use the actual number.



An exception to this rule occurs if you specify the Precision as displayed option in the Calculation tab of the Options dialog box. If that option is in effect, formulas will use the values that are actually displayed in the cells.

Automatic Number Formatting

Excel is smart enough to perform some formatting for you automatically. For example, if you enter 12.3% into a cell, Excel knows that you want to use a percentage format and applies it for you automatically. If you use commas to separate thousands (such as 123,456), Excel applies comma formatting for you. And if you precede your value with a dollar sign, Excel formats the cell for currency.



Beginning with Excel 2000, you have an option when it comes to entering values into cells formatted as a percentage. Select Tools → Options, and click the Edit tab in the Options dialog box. If the check box labeled Enable automatic percent entry is checked, you can simply enter a normal value into a cell formatted to display as a percent (for example, enter **12.5** for 12.5%). If this check box is not checked, you must enter the value as a decimal (for example, **.125** for 12.5%).

Excel automatically applies a built-in number format to a cell based on the following criteria:

- ◆ If a number contains a slash (/), it may be converted to a date format or a fraction format.
- ◆ If a number contains a hyphen (-), it may be converted to a date format.
- ◆ If a number contains a colon (:), or is followed by a space and the letter A or P, it may be converted to a time format.
- ◆ If a number contains the letter E (in either uppercase or lowercase), it may be converted to scientific notation or exponential format.



To avoid automatic number formatting when you enter a value, preformat the cell with the desired number format, or precede your entry with an apostrophe (this makes the entry text).

Formatting Numbers Using Toolbar Buttons

The Formatting toolbar contains several buttons that enable you to quickly apply common number formats. When you click one of these buttons, the selected cells take on the specified number format. Table C-1 summarizes the formats that these Formatting toolbar buttons perform in the U.S. English version of Excel.



These five toolbar buttons actually apply predefined styles to the selected cells. Access Excel's styles by using the Format → Style command.

TABLE C-1 NUMBER-FORMATTING BUTTONS ON THE FORMATTING TOOLBAR

Button Name	Formatting Applied
Currency Style	Adds a dollar sign to the left, separates thousands with a comma, and displays the value with two digits to the right of the decimal point
Percent Style	Displays the value as a percentage, with no decimal places
Comma Style	Separates thousands with a comma and displays the value with two digits to the right of the decimal place
Increase Decimal	Increases the number of digits to the right of the decimal point by one
Decrease Decimal	Decreases the number of digits to the right of the decimal point by one

Using Shortcut Keys to Format Numbers

Another way to apply number formatting is to use shortcut keys. Table C-2 summarizes the shortcut key combinations that you can use to apply common number formatting to the selected cells or range.

TABLE C-2 NUMBER-FORMATTING KEYBOARD SHORTCUTS

Key Combination	Formatting Applied
Ctrl+Shift+~	General number format (i.e., unformatted values)
Ctrl+Shift+\$	Currency format with two decimal places (negative numbers appear in parentheses)
Ctrl+Shift+%	Percentage format, with no decimal places
Ctrl+Shift+^	Scientific notation number format, with two decimal places
Ctrl+Shift+#	Date format with the day, month, and year
Ctrl+Shift+@	Time format with the hour, minute, and AM or PM
Ctrl+Shift+!	Two decimal places, thousands separator, and a hyphen for negative values

Using the Format Cells Dialog Box to Format Numbers

For optimal control of number formatting, use the Number tab of the Format Cells dialog box. Select the cells to format, and then choose Format → Cells. The Number tab of the Format Cells dialog box displays 12 categories of number formats from which to choose. When you select a category from the list box, the right side of the dialog box changes to display appropriate options. For example, Figure C-1 shows how the dialog box looks when you click the Currency category, and select \$ in the Symbol field.

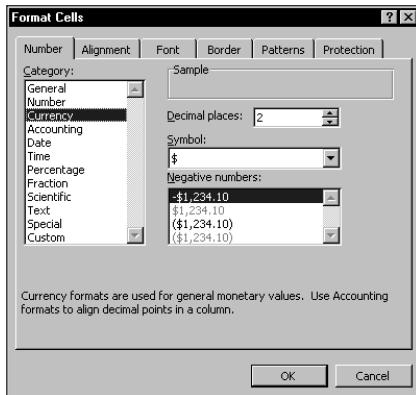


Figure C-1: Options for the Currency category

Following is a list of the number-format categories, along with some general comments:

- ◆ **General:** The default format; it displays numbers as integers, decimals, or in scientific notation if the value is too wide to fit in the cell.
- ◆ **Number:** Enables you to specify the number of decimal places, whether to use your system thousands separator (for example, a comma) to separate thousands, and how to display negative numbers.
- ◆ **Currency:** Enables you to specify the number of decimal places, choose a currency symbol, and display negative numbers. This format always uses the system thousands separator symbol (for example, a comma) to separate thousands.
- ◆ **Accounting:** Differs from the Currency format in that the currency symbols always line up vertically.
- ◆ **Date:** Enables you to choose from a variety of date formats. Excel 2002 also lets you select the locale for your date formats.
- ◆ **Time:** Enables you to choose from a number of time formats. Excel 2002 also lets you select the locale for your time formats.

- ◆ **Percentage:** Enables you to choose the number of decimal places; always displays a percent sign.
- ◆ **Fraction:** Enables you to choose from among nine fraction formats.
- ◆ **Scientific:** Displays numbers in exponential notation (with an E):
2.00E+05 = 200,000. 2.05E+05 = 205,000. You can choose the number of decimal places to display to the left of E.
- ◆ **Text:** When applied to a value, causes Excel to treat the value as text (even if it looks like a value). This feature is useful for items such as numerical part numbers.
- ◆ **Special:** Contains additional number formats. The list varies, depending on the Locale you choose. For the English (United States) locale, the formatting options are Zip Code, Zip Code +4, Phone Number, and Social Security Number.
- ◆ **Custom:** Enables you to define custom number formats not included in any of the other categories.



If the cell displays a series of hash marks (such as #####), it usually means that the column is not wide enough to display the value by using the number format that you selected. Either make the column wider or change the number format. A series of hash marks also can mean that the cell contains an invalid date or time.

Formatting Numbers in Charts

When you create a chart, the number formatting on the chart is linked to the worksheet cells that contain the numbers. For example, the values displayed on the chart's value axis or data labels use the same number format as the values used to create the chart. If you like, you can apply number formats (including custom number formats) to values that appear in charts.

Generally, you can double-click any part of a chart that displays a number. This brings up the appropriate Format dialog box. Click the Number tab and specify the desired number format. You can choose from a built-in format, or use a custom number format.

To reestablish links between a chart's number formats and the worksheet number formatting, select the Linked to source check box on the Number tab of the Format dialog box.

Creating a Custom Number Format

Figure C-2 shows how the Format Cells dialog box looks when you select the Custom category. This category enables you to create number formats not included in any of the other categories. As you can see, Excel gives you a great deal of flexibility in creating custom number formats.

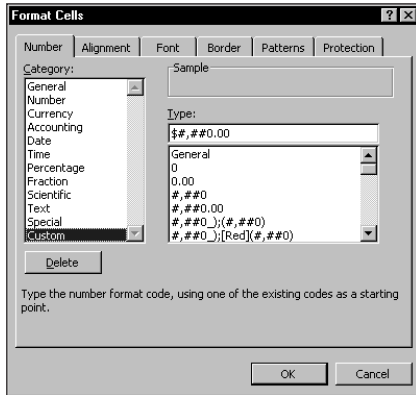


Figure C-2: The Custom category enables you to create custom number formats.



Custom number formats are stored with the worksheet. To make the custom format available in a different workbook, you can just copy a cell that uses the custom format to the other workbook.

About Custom Number Formats

You construct a number format by specifying a series of codes as a *number format string*. You enter this code sequence in the Type field after you select the Custom category on the Number tab of the Format Cells dialog box. Here's an example of a simple number format code:

0.000

This code consists of placeholders and a decimal point; it tells Excel to display the value with three digits to the right of the decimal place. Here's another example:

00000

This custom number format has five placeholders and displays the value with five digits (no decimal point). This is a good format to use when the cell holds a zip code (in fact, this is the code actually used by the ZIP Code format in the Special category). When you format the cell with this number format and then enter a zip code such as 06604 (Bridgeport, CT), the value is displayed with the leading zero. If you enter this number into a cell with the General number format, it displays 6604 (no leading zero).

Scroll through the list of number formats in the Custom category in the Format Cells dialog box to see many more examples. In many cases, you can use one of these codes as a starting point, and only slight customization will be needed.

Parts of a Number Format String

A custom format string enables you to specify different format codes for positive numbers, negative numbers, zero values, and text. You do so by separating the codes with a semicolon. The codes are arranged in the following structure:

```
Positive format; Negative format; Zero format; Text format
```

If you use only one section, the format string applies to all values. If you use two sections, the first section applies to positive values and zeros, the second to negative values. If you use three sections, the first section applies to positive values, the second to negative values, and the third to zeros.

The following is an example of a custom number format that specifies a different format for each of these types:

```
[Green]General;[Red]General;[Black]General;[Blue]General
```

This example takes advantage of the fact that colors have special codes. A cell formatted with this custom number format displays its contents in a different color, depending on the value. When a cell is formatted with this custom number format, a positive number is green, a negative number is red, a zero is black, and text is blue.



If you want to apply cell formatting automatically (such as text or background color) based on the cell's contents, a better solution is to use Excel's Conditional Formatting feature (available in Excel 97 or later). Chapter 19 discusses this feature.

Preformatting Cells

Usually, you'll apply number formats to cells that already contain values. You also can format cells with a specific number format *before* you make an entry. Then, when you enter information, it takes on the format that you specified. You can preformat specific cells, entire rows or columns, or even the entire worksheet.

Rather than preformat an entire worksheet, however, you can change the number format for the Normal style (unless you specify otherwise, all cells use the Normal style). Change the Normal style by selecting Format → Style. In the Style dialog box, click the Modify button and then choose the new number format for the Normal style.

Custom Number Format Codes

Table C-3 lists the formatting codes available for custom formats, along with brief descriptions. I use most of these codes in examples later in this appendix.

TABLE C-3 CODES USED TO CREATE CUSTOM NUMBER FORMATS

Code	Comments
General	Displays the number in General format
#	Digit placeholder
0 (zero)	Digit placeholder
?	Digit placeholder
.	Decimal point
%	Percentage
,	Thousands separator
E- E+ e- e+	Scientific notation
\$ - + / () : space	Displays this character
\	Displays the next character in the format
*	Repeats the next character, to fill the column width
_ (underscore)	Leaves a space equal to the width of the next character
"text"	Displays the text inside the double quotation marks
@	Text placeholder

Code	Comments
[<i>color</i>]	Displays the characters in the color specified
[COLOR <i>n</i>]	Displays the corresponding color in the color palette, where <i>n</i> is a number from 0 to 56
[condition value]	Enables you to set your own criteria for each section of a number format

Table C-4 lists the codes used to create custom formats for dates and times.

TABLE C-4 CODES USED IN CREATING CUSTOM FORMATS FOR DATES AND TIMES

Code	Comments
m	Displays the month as a number without leading zeros (1–12)
mm	Displays the month as a number with leading zeros (01–12)
mmm	Displays the month as an abbreviation (Jan–Dec)
mmmm	Displays the month as a full name (January–December)
mmmmm	Displays the first letter of the month (J–D)
d	Displays the day as a number without leading zeros (1–31)
dd	Displays the day as a number with leading zeros (01–31)
ddd	Displays the day as an abbreviation (Sun–Sat)
dddd	Displays the day as a full name (Sunday–Saturday)
yy or yyyy	Displays the year as a two-digit number (00–99) or as a four-digit number (1900–9999)
h or hh	Displays the hour as a number without leading zeros (0–23) or as a number with leading zeros (00–23)
m or mm	Displays the minute as a number without leading zeros (0–59) or as a number with leading zeros (00–59)
s or ss	Displays the second as a number without leading zeros (0–59) or as a number with leading zeros (00–59)
[]	Displays hours greater than 24 or minutes or seconds greater than 60
AM/PM	Displays the hour using a 12-hour clock; if no AM/PM indicator is used, the hour uses a 24-hour clock

Where Did Those Number Formats Come From?

Excel may create custom number formats without you realizing it. When you use the Increase Decimal or Decrease Decimal button on the Formatting toolbar, new number formats are created that appear on the Number tab of the Format Cells dialog box. (To access this dialog box, click Cells on the Format menu.) For example, if you click the Increase Decimal button five times, the following custom number formats are created:

0.0
0.000
0.0000
0.000000

A format string for two decimal places is not created because that format string is built-in.

Custom Number Format Examples

The remainder of this appendix consists of useful examples of custom number formats. You can use most of these format codes as-is. Others may require slight modification to meet your needs.

Scaling Values

You can use a custom number format to scale a number. For example, if you work with very large numbers, you may want to display the numbers in thousands (i.e., displaying 1,000,000 as 1,000). The actual number, of course, will be used in calculations that involve that cell. The formatting affects only how it is displayed.

DISPLAYING VALUES IN THOUSANDS

The following format string displays values without the last three digits to the left of the decimal place, and no decimal places. In other words, the value appears as if it's divided by 1,000 and rounded to no decimal places.

`#,###,`

A variation of this format string follows. A value with this number format appears as if it's divided by 1,000 and rounded to two decimal places.

`#,###.00,`

Table C-5 shows examples of these number formats:

TABLE C-5 EXAMPLES OF DISPLAYING VALUES IN THOUSANDS

Value	Number Format	Display
123456	#,###,	123
1234565	#,###,	1,235
-323434	#,###,	-323
123123.123	#,###,	123
499	#,###,	(blank)
500	#,###,	1
123456	#,###.00,	123.46
1234565	#,###.00,	1,234.57
-323434	#,###.00,	-323.43
123123.123	#,###.00,	123.12
499	#,###.00,	.50
500	#,###.00,	.50

DISPLAYING VALUES IN HUNDREDS

The following format string displays values in hundreds, with two decimal places. A value with this number format appears as if it's divided by 100, and rounded to two decimal places.

0" . "00

Table C-6 shows examples of these number formats:

TABLE C-6 EXAMPLES OF DISPLAYING VALUES IN HUNDREDS

Value	Number Format	Display
546	0""00	5.46
100	0""00	1.00

Continued

TABLE C-6 EXAMPLES OF DISPLAYING VALUES IN HUNDREDS (Continued)

Value	Number Format	Display
9890	0:"00	98.90
500	0:"00	5.00
-500	0:"00	-5.00
0	0:"00	0.00

DISPLAYING VALUES IN MILLIONS

The following format string displays values in millions, with no decimal places. A value with this number appears as if it's divided by 1,000,000, and rounded to no decimal places.

```
#,###,.
```

A variation of this format string follows. A value with this number appears as if it's divided by 1,000,000, and rounded to two decimal places.

```
#,###.00,.
```

Another variation follows. This adds the letter M to the end of the value.

```
#,###,.,M
```

The following format string is a bit more complex. It adds the letter M to the end of the value – and also displays negative values in parentheses as well as displaying zeros.

```
#,###.0,., "M" ); ( #,###.0,., "M" ); 0.0 "M" )
```

Table C-7 shows examples of these format strings:

TABLE C-7 EXAMPLES OF DISPLAYING VALUES IN MILLIONS

Value	Number Format	Display
123456789	#,###,.	123
1.23457E+11	#,###,.	123,457

Value	Number Format	Display
1000000	#,###,.	1
5000000	#,###,.	5
-5000000	#,###,.	-5
0	#,###,.	(blank)
123456789	#,###.00,.	123.46
1.23457E+11	#,###.00,.	123,456.7
1000000	#,###.00,.	1.00
5000000	#,###.00,.	5.00
-5000000	#,###.00,.	-5.00
0	#,###.00,.	.00
123456789	#,###,,"M"	123M
1.23457E+11	#,###,,"M"	123,457M
1000000	#,###,,"M"	1M
5000000	#,###,,"M"	5M
-5000000	#,###,,"M"	-5M
0	#,###,,"M"	M
123456789	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	123.5M
1.23457E+11	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	123,456.8M
1000000	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	1.0M
5000000	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	5.0M
-5000000	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	(5.0M)
0	#,###.0,,"M"); (#,###.0,,"M");0.0"M"_)	0.0M

ADDING ZEROS TO A VALUE

The following format string displays a value with three additional zeros and no decimal places. A value with this number format appears as if it's rounded to no decimal places and then multiplied by 1,000.

```
#",000"
```

Examples of this format string, plus a variation that adds six zeros, are shown in Table C-8.

TABLE C-8 EXAMPLES OF DISPLAYING A VALUE WITH EXTRA ZEROS

Value	Number Format	Display
1	#",000"	1,000
1.5	#",000"	2,000
43	#",000"	43,000
-54	#",000"	-54,000
5.5	#",000"	6,000
0.5	#",000,000"	1,000,000
0	#",000,000"	,000,000
1	#",000,000"	1,000,000
1.5	#",000,000"	2,000,000
43	#",000,000"	43,000,000
-54	#",000,000"	-54,000,000
5.5	#",000,000"	6,000,000
0.5	#",000,000"	1,000,000

Hiding Zeros

In the following format string, the third element of the string is empty, which causes zero value cells to display as blank:

```
General;General;:@
```

This format string uses the General format for positive and negative values. You can, of course, substitute any other format codes.

Displaying Leading Zeros

To display leading zeros, create a custom number format that uses the 0 character. For example, if you want all numbers to display with 10 digits, use the number format string that follows. Values with fewer than 10 digits will display with leading zeros.

```
0000000000
```

You also can force all numbers to display with a fixed number of leading zeros. The format string that follows, for instance, appends three zeros to the beginning of each number:

```
"000"#
```

In the following example, the format string uses the repeat character code (an asterisk) to apply leading zeros to fill the entire width of the cell:

```
*00
```

Formatting Percentages

Using a percent symbol (%) in a format string causes the cell to display in percentage format. Note that the percent sign also appears in the formula bar.

The following format string formats values less than or equal to 1 in Percentage format. Values greater than 1 and text are formatted using the General format.

```
[<=1]0.00%;General
```

When you mix cells with percent and normal formatting in a column, you may prefer to see the nonpercent values indented from the right so the values line up properly. To do so, apply the following number format to nonpercent cells. This format string uses an underscore followed by the percent symbol. The result is a space equal to the width of the percent symbol.

```
#.00_%
```

Figure C-3 shows a worksheet that uses this number format for the nonpercent cells (the range C6:C12).

	A	B	C	D
1	Value	Number Format	Display	
2	0.4355	0.00%	43.55%	
3	0.015	0.00%	1.50%	
4	0.0024	0.00%	0.24%	
5	0.1991	0.00%	19.91%	
6	146.36	[<=1]0.0%;General	146.36	
7	226.84	[<=1]0.0%;General	226.84	
8	204.908	[<=1]0.0%;General	204.91	
9	66.6	[<=1]0.0%;General	66.60	
10	40.6	[<=1]0.0%;General	40.60	
11	98.4229	[<=1]0.0%;General	98.42	
12	43.5	[<=1]0.0%;General	43.50	

Figure C-3: Use a custom number format to align numbers.

Displaying Fractions

Excel supports quite a few built-in fraction number formats (select the Fraction category). For example, to display the value .125 as a fraction with 8 as the denominator, select As eighths (4/8) from the Type list (see Figure C-4).

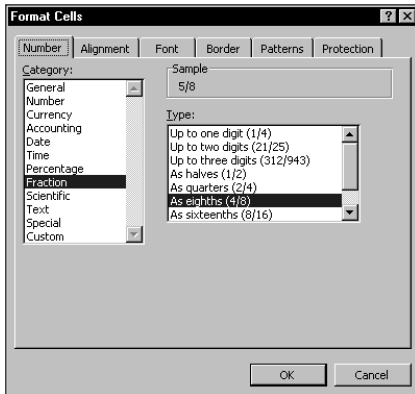


Figure C-4: Selecting a number format to display a value as a fraction

You can use a custom format string to create other fractional formats. For example, the following format string displays a value in 50ths:

```
# ??/50
```

The following format string displays a value in terms of fractional dollars. For example, the value 154.87 is displayed as *154 and 87/100 Dollars*.

```
0 "and "??/100 "Dollars"
```

The following example displays the value in sixteenths, with a quotation mark appended to the right. This format string is useful when you deal with inches (for example, 2/16").

```
# ??/16\"
```

Displaying N/A for Text

The following number format string uses General formatting for all cell entries except text. Text entries appear as N/A.

```
General;General;General;N/A
```

You can, of course, modify the format string to display specific formats for values. The following variation displays values with one decimal place:

```
0.0;0.0;0.0;N/A
```

Displaying Text in Quotes

The following format string displays numbers normally, but surrounds text with quotation marks:

```
General;General;General;"@"
```

Repeating Text

The number format string displays the contents of the cell three times. For example, if the cell contains the text *Budget*, the cell displays *Budget Budget Budget*.

```
:::@ @ @
```

Testing Custom Number Formats

When you create a custom number format, don't overlook the Sample box in the Number tab of the Format Cells dialog box. This box displays the value in the active cell using the format string in the Type box.

It's a good idea to test your custom number formats using the following data: a positive value, a negative value, a zero value, and text. Often, creating a custom number format takes several attempts. Each time you edit a format string, it is added to the list. When you finally get the correct format string, access the Format Cells dialog box one more time and delete your previous attempts.

Displaying a Negative Sign on the Right

The following format string displays negative values with the negative sign to the right of the number. Positive values have an additional space on the right, so both positive and negative numbers align properly on the right.

```
0.00_ - ;0.00-
```

Figure C-5 shows this format string in use.

	A	B	C	D
1	Value	Number Format	Display	
2	1.5	0.00_ - ;0.00-	1.50	
3	0	0.00_ - ;0.00-	0.00	
4	-3.4	0.00_ - ;0.00-	3.40-	
5	9	0.00_ - ;0.00-	9.00	
6	-0.5	0.00_ - ;0.00-	0.50-	
7	-2.5	0.00_ - ;0.00-	2.50-	
8	-2.25	0.00_ - ;0.00-	2.25-	
9	0	0.00_ - ;0.00-	0.00	
10	5.8	0.00_ - ;0.00-	5.80	
11	-5.8	0.00_ - ;0.00-	5.80-	
12	2	0.00_ - ;0.00-	2.00	
13				

Figure C-5: Using a custom number format that displays the negative sign on the right

Conditional Number Formatting

Conditional formatting refers to formatting that is applied based on the contents of a cell. Excel's Conditional Formatting feature provides the most efficient way to perform conditional formatting, but you also can use custom number formats.



Conditional formatting is limited to three conditions — two of them explicit, and the third one implied (that is, everything else). The conditions are enclosed in square brackets and must be simple numeric comparisons.

The following format string uses a different format, depending on the value in the cell. This format string essentially separates the numbers into three groups: less than or equal to 4, greater than or equal to 8, and other. Figure C-6 shows an example of this format string in use.

```
[<=4]"Low"* 0;[>=8]"High"* 0;"Medium"* 0
```

	A	B	C	D
	Value	Number Format	Display	
1	1	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Low	1
2	2	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Low	2
3	3	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Low	3
4	4	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Low	4
5	5	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Medium	5
6	6	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Medium	6
7	7	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	Medium	7
8	8	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	High	8
9	9	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	High	9
10	10	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	High	10
11	11	[<=4]"Low"* 0; >=8]"High"* 0;"Medium"* 0	High	11
12				
13				

Figure C-6: Cells in column C use a conditional number format.

The following number format string displays values less than 1 with a cent symbol on the right (for example, .54¢). Otherwise, values display with a dollar sign (for example, \$3.54).

```
[<1].00¢; $0.00_¢
```

The following number format is useful for telephone numbers. Values greater than 9999999 (that is, numbers with area codes) are displayed as (xxx) xxx-xxxx. Other values (numbers without area codes) are displayed as xxx-xxxx.

```
[>9999999](000) 000-0000; 000-0000
```

For Zip codes, you might want to use the format string that follows. This displays Zip codes using five digits. But if the number is greater than 99999, it uses the “Zip plus four” format (xxxxx-xxxx).

```
[>99999]00000-0000; 00000
```

Coloring Values

Custom number format strings can display the cell contents in various colors. The following format string, for example, displays positive numbers in red, negative numbers in green, zero values in black, and text in blue:

```
[Red]General; [Green]-General; [Black]General; [Blue]General
```

Following is another example of a format string that uses colors. Positive values are displayed normally; negative numbers and text display the text Error! in red.

```
General; [Red]"Error!"; 0; [Red]"Error!"
```

Using the following format string, values that are less than 2 are displayed in red. Values greater than 4 are displayed in green. Everything else (text, or values between 2 and 4) displays in black.

```
[Red][<2]General;[Green][>4]General;[Black]General
```

As seen in the preceding examples, Excel recognizes color names such as [Red] and [Blue]. It also can use other colors from the color palette, indexed by a number. The following format string, for example, displays the cell contents using the sixteenth color in the color palette:

```
[Color16]General
```



You cannot change cells that are colored using a number format string by using normal cell formatting commands.

Formatting Dates and Times

When you enter a date into a cell, Excel formats the date using the system short date format. You can change this format using the Windows Control Panel (Regional Settings).

Excel provides many useful built-in date and time formats. The following table shows some other date and time formats that you may find useful. The first column of the table shows the date/time serial number.

Value	Number Format	Display
36676	mmm d, yyyy (dddd)	May 30, 2000 (Tuesday)
36676	"It's" dddd!	It's Tuesday!
36676	dddd, mm/dd/yyyy	Tuesday, 05/30/2000
36676	"Month: "mmm	Month: May
36676	General (m/d/yyyy)	36676 (5/30/2000)
0.345	h "Hours"	8 Hours
0.345	h:mm o'clock	8:16 o'clock
0.345	h:mm a/p"m"	8:16 am
0.78	h:mm a/p".m."	6:43 p.m.



See Chapter 6 for more information about Excel's date and time serial number system.

Displaying Text with Numbers

The ability to display text with a value is one of the most useful benefits of using a custom number format. To add text, just create the number format string as usual and put the text within quotation marks. The following number format string, for example, displays a value with the text (*US Dollars*) added to the end:

```
#,##0.00 "(US Dollars)"
```

Here's another example that displays text before the number:

```
"Average: "0.00
```

If you use the preceding number format, you'll find that the negative sign appears before the text for negative values. To display number signs properly, use this variation:

```
"Average: "0.00;"Average: "-0.00
```

The following format string displays a value with the words *Dollars and Cents*. For example, the number 123.45 displays as *123 Dollars and .45 Cents*.

```
0 "Dollars and" .00 "Cents"
```

Displaying a Zero with Dashes

The following number format string displays zero values as a series of dashes:

```
#,##0.0;-###0.0;-----
```

You can, of course, create lots of variations. For example, you can replace the six hyphens with any of the following:

```
<0>
```

```
-0-
```

```
~~
```

```
<NULL>
```

```
"[NULL]"
```

Formatting Numbers Using the TEXT Function

Excel's TEXT function accepts a number format string as its second argument. For example, the following formula displays the contents of cell A1 using a custom number format that displays a fraction:

```
=TEXT(A1, "# ??/50")
```

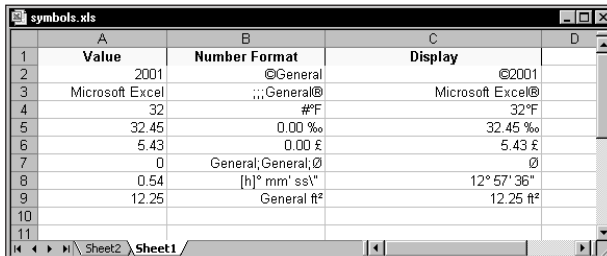
Not all formatting codes function, however. For example, colors and repeating characters are ignored. The following formula does not display the contents of cell A1 in red:

```
=TEXT(A1, "[Red]General")
```

Note that, when using square brackets, you must place them within quotation marks.

Using Special Symbols

Your number format strings can use special symbols, such as the copyright symbol, degrees symbol, and so on. Figure C-7 shows some special symbols used in number format strings.



	A	B	C	D
	Value	Number Format	Display	
1				
2	2001	@General	@2001	
3	Microsoft Excel	:::General@	Microsoft Excel@	
4	32	#°F	32°F	
5	32.45	0.00 %	32.45 %	
6	5.43	0.00 £	5.43 £	
7	0	General;General;@	@	
8	0.54	[h]* mm' ss\''	12° 57' 36"	
9	12.25	General ft²	12.25 ft²	
10				
11				

Figure C-7: Using special symbols in number format strings

To enter a symbol, you need to know the Alt+ keyboard sequence required to create the symbol. For example, you can produce the copyright symbol by pressing Alt+0169 (make sure you use the numeric keypad to enter the digits).



If you're using Excel 2002, you can determine these codes by using the Insert → Symbol command (see Figure C-8). For earlier versions of Excel, use the Windows Character Map program.

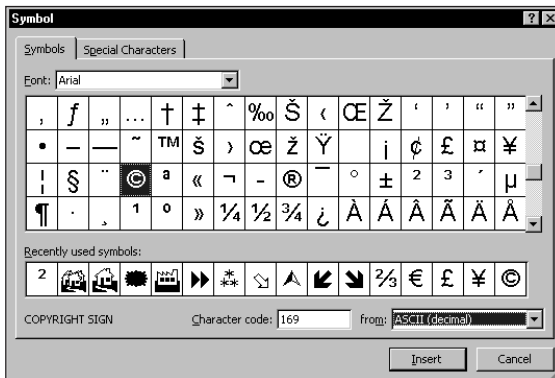


Figure C-8: Use the Insert Symbol dialog box (in Excel 2002 only) to determine the codes for special symbols.

Another use for special symbols is to display superscript characters, such as exponents. For example, Alt+0178 produces a “squared” symbol and Alt+0179 generates a “cubed” symbol.

You can use these special characters to display currency symbols. For example, you might want to display the symbol for the Japanese yen (Alt+0165) or the British pound (Alt+0162).

Suppressing Certain Types of Entries

You can use number formatting to hide certain types of entries. For example, the following format string displays text, but not values:

```
::
```

This format string displays values, but not text or zeros:

```
0.0;-0.0;;
```

This format string displays everything except zeros:

```
0.0;-0.0;;@
```

You can use the following format string to completely hide the contents of a cell:

```
:::
```

Note that when the cell is activated, however, the cell’s contents are visible on the formula bar. A better way to hide a cell’s contents is to select the Hidden option on the Protection tab of the Format Cells dialog box and protect the sheet.

Displaying a Number Format String in a Cell

Excel doesn't have a worksheet function that displays the number format for a specified cell. You can, however, create your own function using VBA. Insert the following function procedure into a VBA module:

```
Function NumberFormat(cell) As String
    ' Returns the number format string for a cell
    Application.Volatile True
    NumberFormat = cell.Range("A1").NumberFormat
End Function
```

Then you can create a formula such as the following:

```
=NumberFormat(C4)
```

This formula returns the number format for cell C4.

This function can be useful in formulas that calculate a conditional sum. For example, you can create a formula that sums only the cells that use a particular number format. See Chapter 7 for information about computing conditional sums.



If the cell contains more than 1,024 characters, the ;;; format string does not hide the contents.



Refer to Part VI for more information about creating custom worksheet functions using VBA.

Filling a Cell with a Repeating Character

The asterisk (*) symbol specifies a repeating character in a number format string. The repeating character completely fills the cell and adjusts if the column width changes. The following format string, for example, displays the contents of a cell padded on the right with dashes:

```
General*-;General*-;General*-;General*-
```

Figure C-9 shows several examples of number format strings that use an asterisk to repeat a character.

	A	B	C
1	Value	NumberFormat	Display
2	45.3	General*;General*;General*;General*	45.3-----
3	-45.43	General*;General*;General*;General*	45.43-----
4	Excel	General*;General*;General*;General*	Excel-----
5	45.3	*-General*;-General*;-General*;-General	-----45.3
6	-45.43	*-General*;-General*;-General*;-General	-----45.43
7	Excel	*-General*;-General*;-General*;-General	-----Excel
8	1434.55	\$\$,##0.00*	\$1,434.55-----
9	1545.98	\$\$,##0.00**	\$1,545.98*****
10	509.54	**	*****
11	Excel	*+	Excel
12	545.98	*+	+++++
13	12.83	*\$General	\$\$\$\$\$\$\$\$\$12.83
14			

Figure C-9: Examples of number formats that use a repeating character

Displaying Leading Dots

The following custom number format is a variation on the accounting format. Using this number format displays the dollar sign on the left and the value on the right. The space in between is filled with dots.

```
_$* .# ,##0.00_) ;_($* .(# ,##0.00) ;_($* "-"??_) ;_(@_)
```


Appendix D

Additional Excel Resources

IF I'VE DONE MY job, the information provided in this book will be very useful to you. The book, however, cannot cover every conceivable topic. Therefore, I've compiled a list of additional resources that you may find helpful. I classify these resources into three categories: Microsoft technical support, Internet newsgroups, and Internet Web sites.

By the way, don't forget about Excel's online help system. This help system seems to get better with each release.

Microsoft Technical Support

Technical support is the common term for assistance provided by a software vendor. In this case, I'm talking about assistance that comes directly from Microsoft. Microsoft's technical support is available in several different forms.

Support Options

To find out your support options, choose the Help → About Microsoft Excel command, and then click the Tech Support button. This opens a help file that lists all the support options offered by Microsoft, including both free and fee-based support.

Through my experience, I suggest you use vendor *standard telephone* support only as a last resort. Chances are, you'll run up a big phone bill (assuming you can even get through) and spend lots of time on hold, but you may or may not find an answer to your question.

The truth is, the people who answer the phone are equipped to answer only the most basic questions. And the answers to these basic questions are usually readily available elsewhere.

Microsoft Knowledge Base

Your best bet for solving a problem may be the Microsoft Knowledge Base. This is the primary Microsoft product information source – an extensive, searchable database that consists of tens of thousands of detailed articles containing technical

information, bug lists, fix lists, and more. You have free and unlimited access to the Knowledge Base via the Internet. The URL is:

<http://search.support.microsoft.com/kb/>

Microsoft Excel Home Page

The official home page of Excel is at:

<http://www.microsoft.com/office/excel>

Microsoft Office Tools on the Web

For information about Office 2002 (including Excel), try this site:

<http://office.microsoft.com>

You'll find product updates, add-ins, examples, and lots of other useful information.



As you know, the Internet is a dynamic entity that changes rapidly. Web sites are often reorganized, so a particular URL listed in this appendix may not be available when you try to access it.

Internet Newsgroups

Usenet is an Internet service that provides access to several thousand special interest groups that enable you to communicate with people who share common interests. A newsgroup works like a public bulletin board. You can post a message or questions and (usually) others reply to your message.

There are thousands of newsgroups covering virtually every topic you can think of (and many that you haven't thought of). Typically, questions posed on a newsgroup are answered within 24 hours – assuming, of course, that you ask the questions in a manner that makes others want to reply.



Besides an Internet connection, you need special newsreader software to access newsgroups. Microsoft Outlook Express (free) is a good choice. This product is part of Internet Explorer.

Spreadsheet Newsgroups

The primary Usenet newsgroup for general spreadsheet users is:

`comp.apps.spreadsheets`

This newsgroup is intended for users of any spreadsheet brand, but about 90 percent of the postings deal with Excel.

Microsoft Newsgroups

Microsoft maintains an extensive list of newsgroups, including quite a few devoted to Excel. If your Internet service provider doesn't carry the Microsoft newsgroups, you can access them directly from Microsoft's news server. You need to configure your newsreader software to access Microsoft's news server at this address:

`msnews.microsoft.com`

Table D-1 lists the key Excel newsgroups found on Microsoft's news server.

TABLE D-1 MICROSOFT.COM'S EXCEL-RELATED NEWSGROUPS

Newsgroup	Topic
<code>microsoft.public.excel.programming</code>	Programming Excel with VBA or XLM macros
<code>microsoft.public.excel.123quattro</code>	Converting 1-2-3 or Quattro Pro sheets into Excel sheets
<code>microsoft.public.excel.worksheet.functions</code>	Worksheet functions
<code>microsoft.public.excel.charting</code>	Building charts with Excel
<code>microsoft.public.excel.printing</code>	Printing with Excel
<code>microsoft.public.excel.queryDAO</code>	Using Microsoft Query and Data Access Objects (DAO) in Excel
<code>microsoft.public.excel.datamap</code>	Using the Data Map feature in Excel
<code>microsoft.public.excel.crashesGPFs</code>	Help with General Protection Faults or system failures

Continued

TABLE D-1 MICROSOFT.COM'S EXCEL-RELATED NEWSGROUPS (Continued)

Newsgroup	Topic
microsoft.public.excel.misc	General topics that do not fit one of the other categories
microsoft.public.excel.links	Using links in Excel
microsoft.public.excel.macintosh	Excel issues on the Macintosh operating system
microsoft.public.excel.interopdedde	OLE, DDE, and other cross-application issues
microsoft.public.excel.setup	Setting up and installing Excel
microsoft.public.excel.templates	Spreadsheet Solutions templates and other XLT files
microsoft.public.excel.sdk	Issues regarding the Excel Software Development Kit

Searching Newsgroups

Many people don't realize that you can perform a keyword search on past newsgroup postings. Often, this is an excellent alternative to posting a question to the newsgroup because you can get the answer immediately. The best source for searching newsgroup postings is Google.com, at the following Web address:

<http://groups.google.com>



Formerly, newsgroup searches were performed at the Deja.com Web site. That site has closed down, and the newsgroup archives were purchased by Google.

How does searching work? Suppose you have a problem identifying unique values in a range of cells. You can perform a search using the following keywords: **Excel**, **Range**, and **Unique**. The Google search engine probably will find dozens of newsgroup postings that deal with these topics. It may take a while to sift through the messages, but you have an excellent chance of finding an answer to your question.

Tips for Posting to a Newsgroup

1. Make sure that your question has not already been answered. Check the FAQ (if one exists) and also perform a Google.com search (see "Searching Newsgroups" in this appendix).
2. Make the subject line descriptive. Postings with a subject line such as "Help me!" and "Excel Question" are less likely to be answered than postings with a more specific subject, such as "Need Help With Custom Worksheet Function."
3. Specify the spreadsheet product and version that you use. In many cases, the answer to your question depends on your version of Excel.
4. Make your question as specific as possible.
5. Keep your question brief and to the point, but provide enough information so someone can answer it adequately.
6. Indicate what you've done to try to answer your own question.
7. Post in the appropriate newsgroup, and don't cross-post to other groups unless the question applies to multiple groups.
8. Don't type in all uppercase or all lowercase, and check your grammar and spelling.
9. Don't include a file attachment.
10. Avoid posting in HTML format.
11. If you request an e-mail reply in addition to a newsgroup reply, don't use an "anti-spam" e-mail address that requires the responder to modify your address. Why cause extra work for someone doing *you* a favor?

Internet Web Sites

If you have access to the World Wide Web (WWW), you can find some very useful Web sites devoted to Excel. I list a few of my favorites here.

The Spreadsheet Page

This is my own Web site, which contains files to download, developer tips, instructions for accessing Excel Easter eggs, spreadsheet jokes, an extensive list of links to other Excel sites, and information about my books. The URL is:

<http://www.j-walk.com/ss>



This site also contains a list of errors that I've found in each of my books, including the book you're reading now. (Yes, a few errors have been known to creep into these pages.)

Excel Web Source

This site, maintained by Chip Pearson, contains dozens of useful examples of VBA and clever formula techniques. The URL is:

<http://www.cpearson.com/excel.htm>

Stephen Bullen's Excel Page

Stephen's Web site contains some fascinating examples of Excel code, including a section titled "They Said It Couldn't Be Done." The URL is:

<http://www.bmsltd.co.uk/excel>

Spreadsheet FAQ

Many newsgroups have a *FAQ*—a list of frequently asked questions. The purpose of providing a list of FAQs is to prevent the same questions from being asked over and over. The FAQ for the comp.apps.spreadsheets newsgroup is available at:

<http://www.faqs.org/faqs/spreadsheets/faq>

Appendix E

What's on the CD-ROM

This appendix describes the contents of the companion CD-ROM.

CD-ROM Overview

The CD-ROM consists of four components:

- ◆ **Chapter Examples:** Excel workbooks that are discussed in the chapters of this book.
- ◆ **Power Utility Pak 2000:** A 30-day trial version of the author's popular Excel add-in (works with Excel 97 or later). Use the coupon in this book to order the latest version of PUP at a significant discount. The complete VBA source code is also available for a small fee.
- ◆ **Sound-Proof 2000:** The demo version of the author's audio proofreader add-in.
- ◆ Complete, searchable version of the book in PDF format (use Acrobat Reader to access these files).
- ◆ The latest version of Acrobat Reader from Adobe.



All CD-ROM files are read-only. Therefore, if you open a file from the CD-ROM and make any changes to it, you need to save it to your hard drive. Also, if you copy a file from the CD-ROM to your hard drive, the file retains its read-only attribute. To change this attribute after copying a file, right-click the filename or icon and select Properties from the shortcut menu. In the Properties dialog box, click the General tab and remove the check mark from the Read-only check box.

Chapter Examples

Most of the chapters in this book refer to workbooks that are available on the CD-ROM. Each chapter has its own subdirectory on the CD-ROM. For example, you can find the files for Chapter 5 in the following directory:

```
chapters\chap05\
```

Following is a list of the chapter examples with a brief description of each. Note that not all chapters have example files.

Chapter 5

character set.xls	Displays all characters for a selected font. Requires Excel 97 or later.
identifying text in cells.xls	Examples of three functions (ISTEXT, CELL, and TYPE) that are supposed to identify the type of data in a cell.
text formula examples.xls	Contains the example formulas described in the chapter.
text histogram.xls	Displays a histogram using text characters rather than a chat.

Chapter 6

calendar array.xls	A single array formula that displays a monthly calendar.
day of the week count.xls	Counts the number of each day of the week for a particular year.
holidays.xls	Formulas that calculate the dates of various holidays.
jogging log.xls	Formulas to keep track of jogging data.
ordinal dates.xls	Formulas that express a date as an ordinal number (such as June 13th, 1999).
time sheet.xls	A workbook (with VBA macros) to keep track of hours worked in a week. This example is not discussed in this chapter.
work days.xls	Demonstrates the NETWORKDAYS function.

Chapter 7

adjustable bins.xls	Demonstrates a histogram with adjustable bins.
basic counting.xls	Formulas that demonstrate basic counting techniques.
conditional summing.xls	Formulas that demonstrate various ways to calculate conditional sums.
count unique.xls	Formulas to count the number of unique entries in a range.
counting text in a range.xls	Formulas that demonstrate various ways to count text in a range.
cumulative sum.xls	Formulas to calculate a cumulative sum.
frequency distribution.xls	Creates a frequency distribution using the FREQUENCY function, the Analysis ToolPak, and formulas.
multiple criteria counting.xls	The workbook used in the multiple criteria counting examples.

Chapter 8

basic lookup examples.xls	Demonstrates four basic lookup techniques.
closest match.xls	Demonstrates how to perform a lookup using the closest matching value.
gpa.xls	Calculates a grade point average with multiple formulas or a single array formula.
grade lookup.xls	Uses a lookup table to determine letter grades.
interpolated lookup.xls	Demonstrates how to perform a lookup using linear interpolation.
lookup address.xls	Demonstrates how to determine the cell address of a lookup item.
lookup to the left.xls	Demonstrates how to perform a lookup when the index column is not the leftmost column in the lookup table.
multiple lookup tables.xls	Demonstrates how to use the IF function to work with multiple lookup tables.

two-column lookup.xls	Demonstrates how to perform a lookup using two columns from the lookup table.
two-way lookup.xls	Demonstrates how to perform a two-way lookup (by columns and by rows).

Chapter 9

data table summary.xls	Demonstrates how to use a one-way and two-way data table to summarize information in a list.
nested subtotals.xls	Demonstrates the use of the SUBTOTAL function.
real estate database.xls	A workbook that contains real estate listing information. Used to demonstrate advanced filtering.

Chapter 10

simultaneous equations.xls	Formulas to solve simultaneous equations with two or three variables.
solve right triangle.xls	Formulas to calculate various parts of a right triangle; gives two known parts.
unit conversion tables.xls	Contains conversion tables for a variety of measurement units.

Chapter 11

Example 01-07 (Simple Accumulations).xls	Examples 1-7.
Examples 08-11 (Complex Accumulations).xls	Examples 8-11.
Examples 12-19 (Simple Discounting).xls	Examples 12-19.
Examples 20-25 (Amortization).xls	Examples 20-25.
Examples 26-29 (Effective Cost of Loans).xls	Examples 26-29.

Examples 30-31 (IPMT, PPMT, CUMIPMT, CUMPRINC).xls	Examples 30-31.
Examples 32-33 (Interest and Payment Frequencies).xls	Examples 32-33.
Examples 34-35 (Non Standard Uses of Functions).xls	Examples 34-35.
Interest Conversion Functions Demo.xls	Examples that demonstrate the use of the VBA interest conversion functions.

Chapter 12

depreciation.xls	Formulas that demonstrate the use of Excel's depreciation functions.
Example 01-09 (NPV).xls	Examples 1-9.
Example 10-12 (IRR).xls	Examples 10-12.
Example 13-15 (MIRR).xls	Examples 13-15.
Example 16 (FVSCHEDULE).xls	Example 16.

Chapter 13

accumulation schedule.xls	An example accumulation schedule.
creating indices.xls	Demonstrates how to create indices.
credit card calculations.xls	Formulas to perform credit card calculation.
detailed loan amortization schedule.xls	A detailed loan amortization schedule.
discounted cash flow schedule.xls	A discounted cash flow schedule.
loan data tables.xls	Demonstrates how to use Excel's Data Table feature.
simple loan amortization schedule.xls	A simple loan amortization schedule.

variable rate analysis.xls	A variable rate analysis example.
variable rate loan amortization schedule.xls	A variable rate loan amortization schedule.
XIRR and XNPV functions.xls	Demonstrates the use of Excel's XIRR and XNPV functions.

Chapter 15

calendar array.xls	An array formula that displays a calendar.
logical functions.xls	Demonstrates how to use logical functions in an array formula.
multi-cell array formulas.xls	Examples of array formulas that occupy multiple cells.
single-cell array formulas.xls	Examples of array formulas that occupy a single cell.
sorted function.xls	A custom VBA function that returns a sorted range.
sum every nth.xls	An array formula to sum every nth value in a range.

Chapter 16

all-time high.xls	Demonstrates how to keep track of the highest value ever entered into a range.
circular reference.xls	The introductory circular reference example file.
net profit (circular).xls	Formulas to calculate net profit using a circular reference.
net profit (not circular).xls	Formulas to calculate net profit without using a circular reference.
recursive equations.xls	Demonstrates how to use a circular reference to solve recursive equations.
simultaneous equations.xls	Demonstrates how to use a circular reference to solve simultaneous equations.

time stamp.xls	Demonstrates how to time-stamp a cell using a circular reference.
unique random integers.xls	Demonstrates how to generate unique random integers by using circular references.

Chapter 17

animated shapes.xls	Demonstrates animated objects (including charts).
autoupdate chart.xls	Demonstrates a technique to plot new data as it's added to the worksheet.
box plot.xls	Demonstrates a box plot to summarize data across groups.
bullen function plotter.xls	Demonstrates a technique developed by Stephen Bullen that plots a function automatically.
chart data in active row.xls	Demonstrates a charting technique that uses the data in the row of the active cell.
chart from combo box.xls	Uses a combo box to select data to be plotted.
circle.xls	Demonstrates how to plot a circle in a chart.
clock chart vba.xls	An alternate version of the analog clock that uses VBA rather than formulas.
clock chart.xls	Displays an analog clock in a chart.
comparative histogram.xls	Demonstrates a comparative histogram (population pyramid).
gantt chart.xls	Demonstrates a simple Gantt (timeline) chart.
gauge chart.xls	Demonstrates how to create a chart that resembles a gauge.
hypocycloid.xls	A chart that generates interesting geometric designs.
linear trendline.xls	Demonstrates a linear trendline.
linked picture.xls	Demonstrates how to use a linked picture of a range in a chart.
multiple charts.xls	Demonstrates how to display multiple charts on a single chart sheet.
nonlinear trendline.xls	Demonstrates nonlinear trendlines.

plot every nth data point.xls	Demonstrates a technique to plot every <i>n</i> th data point.
plot last n data points.xls	Demonstrates a technique to plot only the most recent data.
surface chart.xls	Demonstrates the use of a surface chart to plot a function with two variables.
thermometer chart.xls	Demonstrates a chart that displays progress toward a goal.
xy sketch.xls	Interactive drawing on a chart.

Chapter 18

bank accounts.xls	A worksheet database used in several examples.
calculated field and item.xls	Demonstrates creating calculated fields and items in a pivot table.
sales by date.xls	Demonstrates grouping pivot table items by date.

Chapter 19

conditional formatting.xls	Contains the conditional formatting examples.
data validation.xls	Contains the data validation examples.

Chapter 20

credit card validation.xls	Demonstrates a megaformula to determine whether a credit card number is valid.
no middle name.xls	Demonstrates three ways to remove the middle name from a full name (formulas, a megaformula, and a custom VBA function).
position of last space.xls	Demonstrates a megaformula to return the character position of the last space character in a string.
total interest.xls	Introductory megaformula example.

Chapter 25

\xdate directory	A directory that holds the files for the Extended Date Functions add-in.
celltype function.xls	Demonstrates the CELLTYPE function.
commission function.xls	Demonstrates the COMMISSION function.
counting and summing functions.xls	Demonstrates the counting and summing functions.
date functions.xls	Demonstrates the date functions.
last nonempty cell.xls	Demonstrates the LASTINCOLUMN and LASTINROW functions.
monthnames.xls	Demonstrates the MONTHNAMES function.
multisheet functions.xls	Demonstrates the MAXALLSHEETS and SHEETOFFSET functions.
mysum function.xls	Demonstrates the MYSUM function.
random integers function.xls	Demonstrates the RANOMINTEGERS function.
range randomize function.xls	Demonstrates the RANGERANDOMIZE function.
simple functions.xls	A workbook that demonstrates simple VBA functions.
spelledollars function.xls	Demonstrates the SPELLDOLLARS function.
statfunction function.xls	Demonstrates the STATFUNCTION function.
text manipulation functions.xls	Demonstrates the text manipulation functions.

Power Utility Pak

Power Utility Pak is a collection of Excel add-ins that I developed. The companion CD-ROM contains a copy of the trial version of this product. The trial version can be used for 30 days.

Registering Power Utility Pak

The normal registration fee for Power Utility Pak is \$39.95. You can use the coupon in this book, however, to get a free copy of the latest version of Power Utility Pak (you pay shipping and handling only). In addition, you can purchase the complete VBA source code for only \$20.00.

Installing the trial version

To install the trial version of Power Utility Pak, follow these steps:

1. Make sure that Excel is not running.
2. Locate the PUP2000T.EXE file on the CD-ROM. This file is located in the PUP\ directory.
3. Double-click PUP2000T.EXE. This expands the files to a directory that you specify on your hard drive.
4. Start Excel.
5. Select Tools → Add-Ins, and click the Browse button. Locate the PUP2000.XLA file in the directory you specified in Step 3.
6. Make sure that Power Utility Pak 2000 is checked in the add-ins list.
7. Click OK to close the Add-Ins dialog box.

After you install Power Utility Pak, it will be available whenever you start Excel, and Excel will have a new menu: PUP 2000. Access the Power Utility Pak features from the PUP 2000 menu or select the Create a PUP toolbar command to generate a toolbar.

Power Utility Pak includes extensive online help. Select PUP 2000 → Help to view the Help file.

Uninstalling Power Utility Pak

If you decide that you don't want Power Utility Pak, follow these instructions to remove it from Excel's list of add-ins:

1. In Excel, select Tools → Add-Ins.
2. In the Add-Ins dialog box, remove the check mark from Power Utility Pak 2000.
3. Click OK to close the Add-Ins dialog box.

To remove Power Utility Pak from your system after you've followed the preceding steps to uninstall it from Excel, delete the directory into which you originally installed it.

Sound-Proof 2000

Sound-Proof 2000 is an Excel add-in that I developed. It uses Microsoft Agent to read the contents of selected cells. It's the perfect proofreading tool for anyone who does data entry in Excel.



Excel 2002 includes a new text-to-speech feature. However, you'll find that Sound-Proof 2000 is more customizable and has many additional options.

Cells are read back using natural language format. For example, 154.78 is read as "one hundred fifty-four point seven eight." Date values are read as actual dates (for example, "June fourteen, nineteen ninety-eight") and time values are read as actual times (for example, "six forty-five a.m.").

The companion CD-ROM contains a demo version of Sound-Proof 2000. The demo version's only limitation is that it reads no more than 12 cells at a time. The full version is available for \$24.95. Ordering instructions are provided in the online Help file.

Installing the demo version

To install the demo version of Sound-Proof, follow these steps:

1. Make sure that Excel is not running.
2. Locate the SP2000D.EXE file on the CD-ROM. This file is located in the SP\ directory.
3. Double-click SP2000D.EXE. This expands the files to a directory that you specify on your hard drive.
4. Start Excel.
5. Select Tools → Add-Ins and click the Browse button. Locate the SP2000.XLA file in the directory you specified in Step 3.
6. Make sure that Sound-Proof 2000 is checked in the add-ins list.
7. Click OK to close the Add-Ins dialog box.

After you install Sound-Proof 2000, it will be available whenever you start Excel, and Excel will have a new menu command: Tools → Sound-Proof 2000.

Uninstalling Sound-Proof

If you decide that you don't want Sound-Proof 2000, follow these instructions to remove it from Excel's list of add-ins:

1. In Excel, select Tools → Add-Ins.
2. In the Add-Ins dialog box, remove the check mark from Sound-Proof 2000.
3. Click OK to close the Add-Ins dialog box.

After performing these steps, you can reinstall Sound-Proof at any time by placing a check mark next to the Sound-Proof 2000 item in the Add-Ins dialog box.

To remove Sound-Proof from your system after you have performed the preceding steps to uninstall the add-in from Excel, delete the directory into which you originally installed it.

Electronic Version of Excel 2002 Formulas

The complete (and searchable) text of this book is on the CD-ROM in Adobe's Portable Document Format (PDF), readable with the Adobe Acrobat Reader (also included). For more information on Adobe Acrobat Reader, go to www.adobe.com.

Adobe Acrobat Reader

The Adobe Acrobat Reader is a helpful program that enables you to view the searchable version of this book, which is in .pdf format on the CD-ROM. To install and run Adobe Acrobat Reader, follow these steps:

1. Start Windows Explorer or Windows NT Explorer and then open the Acrobat folder on the CD-ROM.
2. In the Reader folder, double-click the .exe file and follow the instructions presented on-screen for installing Adobe Acrobat Reader.

Index

Symbols & Numbers

- & (ampersand) concatenation operator, 36, 38, 119–120, 629
- ' (apostrophe)
 - number as text prefix, 114
 - Visual Basic for Applications (VBA) comment prefix, 622
- * (asterisk)
 - multiplication operator, 36, 38, 629
 - number format code, 740, 756–757
 - number padding with, 124–125
 - wildcard character, 129, 670
- @ (at sign)
 - Lotus formula/function prefix, 14, 30, 101
 - number format code, 740
- @DCOUNT Lotus function, 707, 714–715
- \ (backslash)
 - integer division operator, 60
 - names, in, 60
 - number format code, 740
- ^ (caret) exponentiation operator, 36, 38, 629
- : (colon)
 - number format code, 740
 - range reference operator, 36, 101
 - time values, in, 169
- , (comma)
 - array element separator, 90, 379, 380
 - chart range reference separator, 442
 - function argument separator, 100
 - number format code, 740
 - thousands separator, 298, 733, 735
 - union operator, 36
- { } (curly brackets) array formula delimiters, 383
- \$ (dollar sign)
 - cell reference prefix, 42
 - number format code, 740
- = (equal sign)
 - array prefix, 382
 - assignment operator, 622
 - Define Name dialog box Refers to field prefix, 81
 - formula prefix, 14, 30, 35
 - logical comparison operator, 36, 38
- > (greater than sign) greater than operator, 36, 38
- >= (greater than sign, equal sign) greater than or equal to operator, 36, 38
- # (hash mark)
 - number format code, 740
 - replacement character, 50, 562
 - Visual Basic for Applications (VBA) date/time delimiter, 628
 - wildcard character, 129, 670
 - #AND# Lotus logical operator, 703
 - #DIV/0! division by zero errors, 50, 564
 - #N/A missing data errors, 50, 185–186, 398, 565
 - #NAME? undefined name/range errors, 50, 565
 - #NOT# Lotus logical operator, 703
 - #NULL! range intersection errors, 50, 565–566
 - #NUM! errors, 50, 566
 - #OR# Lotus logical operator, 703
 - #REF! invalid cell reference errors, 50, 566
 - #VALUE! errors, 50, 567, 610
- < (less than sign) less than operator, 36, 38
- <= (less than sign, equal sign) less than or equal to operator, 36, 38
- <> (less than sign, greater than sign) not equal to operator, 36, 38
- (minus sign)
 - formula prefix, 14, 30
 - negation operator, 38, 569
 - number format code, 740
 - subtraction operator, 36, 38, 569, 629
- () (parentheses)
 - entering, 39, 561
 - function argument delimiters, 99, 100
 - megaformulas, in, 551
 - mismatched, debugging, 561
 - nesting, 39–40
 - number format code, 740
 - operator precedence control using, 30, 38–40, 569
 - SERIES formula argument delimiters, 442
- % (percent sign)
 - number format code, 740
 - percent operator, 36, 38
- . (period)
 - number format code, 740
 - Visual Basic for Applications (VBA) worksheet function prefix, 632
- π (pi), returning value of, 727
- + (plus sign)
 - addition operator, 36, 38, 629
 - formula prefix, 14, 30
 - number format code, 740
- ? (question mark)
 - names, in, 60
 - number format code, 740
 - wildcard character, 129, 670
- ; (semicolon)
 - array element separator, 90, 380
 - number format string code separator, 739
- / (slash)
 - division operator, 36, 38, 629
 - number format code, 740
- [] (square brackets)
 - hour delimiters, 146, 170
 - link formulas, in, 44, 532
 - relative cell references, in, 43

~ (tilde) search operator, 129
 _ (underscore)
 name prefix, 60
 number format code, 740
 0 (zero) number format code, 740
 .123 files, 699

A

A1 notation, 43–44
 AAI. *See* Appraisal Institute of America (AAI)
 ABS function, 173, 407, 726
 absolute values, returning, 173, 407
 Accounting number format, 736
 ACCRINT function, 721
 ACCRINTM function, 721
 accumulation calculations
 deposit, original, 298–299, 303
 discounting compared, 304
 Future Value (FV), 297–298, 299, 301–302, 303, 346–347
 growth rate, average, 340–341
 growth rate, average annual, 298, 299–300
 growth rate, average geometric, 346–347
 interest, 297–304, 362, 721
 multi-variable, 301–304
 Net Present Value (NPV), 324, 337–338, 346–347
 Present Value (PV) at different rates, 324, 346–347
 return rate, average annual, 302, 340–341
 schedules, 361–363
 time periods, 298, 303
 accumulation schedule.xls (on the CD-ROM), 769
 ACOS function, 726
 ACOSH function, 726
 Acrobat Reader (on the CD-ROM), 765, 776
 ACRONYM function, 137, 669–670
 acronyms, generating, 137, 669–670
 Add Trendline dialog box, 469–470, 473
 add-ins. *See also specific add-ins*
 creating using Visual Basic for Applications (VBA), 616–618
 date utilities, 148, 159
 formulas referring to missing, 565
 introduced, 21
 names created by, hidden, 64
 versions of Excel, differences in, 618
 addition operators, 36, 38, 568, 629. *See also* summing
 ADDRESS function, 231, 405, 725
 Address property, 647
 adjustable bins.xls (on the CD-ROM), 767
 Adobe Acrobat Reader (on the CD-ROM), 765, 776
 Advanced Filter dialog box, 247

algebra
 array formulas, using, 284
 coefficients, 283
 constants, 283
 equations, linear, 283
 equations, solving simultaneous, 283–284
 matrix, inverse of coefficient , 284
 variables, 283
 .ALL files, 699
 all-time high.xls (on the CD-ROM), 770
 AMORDEGRC function, 721
 AMORLINC function, 721
 amortization, 308
 amortization calculations
 Payment (PMT), 309, 311–313
 Present Value (PV), 309–310
 rates, 311
 rates, variable, 356–358
 schedules, 320, 322, 352–358
 time periods, 310
 Analysis ToolPak add-in
 chart tools, 199
 date tools, 112, 149–150
 described, 21, 24
 engineering tools, 110
 frequency distribution tools, 198–199
 future value tools, 346
 histogram tool, 199
 information tools, 112
 interest calculation tools, 322
 Internal Rate of Return (IRR) tools, 366
 math tools, 112, 285, 287
 measurement unit conversion tools, 269
 Net Present Value (NPV) tools, 366
 principal calculation tools, 322
 statistical tools, 109
 time tools, 112
 trigonometry tools, 112
 Analysis ToolPak functions, specific
 CONVERT, 269
 CUMIPMT, 322–323, 543
 CUMPRINC, 322–323
 DOLLARDE, 285, 288
 DOLLARFR, 285, 288
 EDATE, 149
 EOMONTH, 149
 FVSCHEDULE, 324, 346–347
 MROUND, 285, 287
 NETWORKDAYS, 150, 154–155
 WEEKNUM, 150
 WORKDAY, 150, 156
 XIRR, 325, 366–368
 XNPV, 366
 YEARFRAC, 150, 157
 AND function, 37, 400–401, 531, 724
 #AND# Lotus logical operator, 703
 AND/OR criteria, 189–191, 208–210
 animated shapes.xls (on the CD-ROM), 771
 AnnEff_Effx function, 315, 325, 335

- AnnEff_Nomx function, 316
- ANSI character set, 117
- Apple computers, 4
- application names, returning, 657
- Application object, 664
- Apply Names dialog box, 73
- APPNAME function, 657
- Appraisal Institute of America (AAI), 330
- area calculations, geometric, 280–281
- AREAS function, 111, 725
- Arglist keyword, 603, 695
- arguments. *See* functions, arguments
- arithmetic. *See* math; numbers
- Array function, 687
- arrays
- 1-dimensional, 375
 - 1-dimensional horizontal, 379–380
 - 1-dimensional vertical, 380
 - 2-dimensional, 375, 380–381, 630
 - 3-dimensional, 375
- AND function, using in array formulas, 400
- Arglist arrays, 695
- averages, calculating using, 394, 402–403
- calendars, creating using, 166–167, 417–418
- cell range references, using in lieu of, 395–396
- cell range values, calculating average of non-zero using, 402–403
- cell range values, counting non-zero using, 403
- cell range values, counting unique using, 193–195
- cell range values, determining closest using, 410
- cell range values, determining validity using, 406
- cell range values, finding nth occurrence of using, 405
- cell range values, finding using, 403–404
- cell range values, returning maximum using, 404–405
- cell range values, returning positive-only using, 414–415
- cell range values, returning unique using, 416–417
- cell range values, summing every nth using, 409–410
- cell ranges, counting differences between using, 404
- cell ranges, counting error values in using, 185–186, 398–399
- cell ranges, returning nonempty cells in using, 415
- cell ranges, reversing cell order using, 415–416
- cell ranges, sorting dynamically using, 416
- cell ranges containing, selecting, 383–384
- cells containing, selecting, 383–384
- charts, range reference conversions in, 445
- columns, returning last value in using, 410–411
- constants, 378–379
- constants, creating from range values, 386–387
- constants, naming, 381–383
- constants, summing operations involving, 378–379
- constants, using in calendars, 416
- constants, using in lieu of range references, 395–396
- counting cell contents using, 192–193
- counting cells meeting AND criteria using, 190–191
- counting cells meeting multiple criteria using, 189–190
- counting cells meeting OR criteria using, 190–191
- counting error values using, 185–186, 398–399
- counting invalid items using, 406
- counting logical values using, 185
- counting most frequently occurring text using, 191
- counting nontext cells using, 185
- counting string occurrences using, 193
- counting text cells using, 185
- crosstab tables, creating dynamic using, 413–414
- data type returned by, 687
- database filter criteria formulas, in, 257–258
- database filter criteria ranges, using in place of, 260, 395
- database functions support of, 260
- debugging array formulas, 567
- declaring in Visual Basic for Applications (VBA), 630–631
- deleting, 384
- dynamic, 416, 631
- editing array formulas, 384–385
- elements in, accessing, 382–383
- elements in, performing operations on, 387–388
- entering array formulas, 30, 376, 377, 383, 385
- equations, array formulas in, 284
- error values, counting using, 185–186, 398–399
- formula bar, array formula entry/editing using, 383, 384
- formulas, using in named, 90–91, 381–383
- frequency distribution calculations, array formulas in, 196
- function arguments, arrays as, 103, 388, 694
- functions, returning from custom, 418–421, 687–688

continued

- arrays *continued*
 - functions returning, 391
 - horizontal, 379–380
 - horizontal, converting to vertical, 382–383, 388–389
 - import operation data validity checks, in, 406
 - integers, of consecutive, 389–390, 406
 - integers, of consecutive arranged randomly, 689–690
 - integers, summing digits of using, 406–408
 - intermediate formulas, eliminating, 394–395
 - introduced, 375
 - list comparison using, 406
 - logical functions in array formulas, 400–401
 - lookup formulas, in, 222, 224, 225–226, 229, 395–396
 - Lotus 1-2-3 functions, converting using array formulas, 715
 - matrix determinants, returning, 726
 - matrix inverse, returning, 726
 - matrix product of two, returning, 727
 - megaformulas, in, 548–552, 554–555
 - multicell, changing cell contents in, 384
 - multicell, creating from range values, 386–387
 - multicell, creating with single formula, 376–377
 - multicell, deleting, 384
 - multicell, deleting cells in, 384
 - multicell, editing, 384–385
 - multicell, moving array formula, 384
 - multicell, moving cells in, 384
 - multicell, operations using, 386–390, 414–418
 - multicell, selecting cells containing, 383–384
 - multicell range, contracting, 385
 - multicell range, expanding, 385
 - multicell range, inserting cells in, 384
 - ranking data using, 412–413
 - rows, returning last value in using, 412
 - selecting, 383–384
 - single-cell, operations using, 375, 377, 390–396, 397
 - sorting values dynamically, 416
 - speed considerations, 385
 - summary data tables, in, 264, 359
 - summing largest range values using, 204, 402
 - summing negative values only using, 206
 - summing operations involving array constants, 378–379
 - summing operations returning, 180
 - summing positive values only using, 399
 - summing ranges containing error values using, 398
 - summing rounded values using, 408
 - summing smallest range values using, 204, 392
 - summing values based on a different range using, 206–207
 - summing values based on conditions using, 399–400, 401
 - summing values meeting AND criteria using, 208–210
 - summing values meeting multiple criteria using, 208
 - summing values meeting OR criteria using, 209–210
 - text, removing non-numeric characters from strings using, 410
 - text strings, returning longest in range using, 405
 - transposing, 382–383, 388–389, 688
 - validation operations using, 406
 - vertical, 380
 - vertical, converting to horizontal, 388–389
- ASIN function, 726
- ASINH function, 726
- ATAN function, 726
- ATAN2 function, 726
- ATANH function, 726
- auditing. *See also* debugging
 - background, 44, 578–579
 - cell dependents, tracing, 576–577
 - cell precedents, tracing, 576
 - cells of particular type, identifying, 573
 - circling invalid data, 537
 - error values, tracing, 563, 577
 - Excel Auditor utility, using, 582
 - Formula Auditing toolbar, using, 537, 574, 576–577
 - Formula Evaluator, using, 580
 - Formula view, using, 573–575
 - formula/result display, showing in separate windows, 573–574
 - Go To Special dialog box, using, 573, 576
 - introduced, 24
 - Power Utility Pak (on the CD-ROM) features, 575, 581–582
 - Spreadsheet Detective, using, 582
- AutoCalculate, 182
- AutoCorrect, Formula, 39–40, 562
- AutoFill, 15, 48, 152–153
- AutoFilter, 22, 240–244, 455–456
- AutoFormats, applying to pivot tables, 500
- AutoShapes, 18
- AutoSum, 107
- autoupdate chart.xls (on the CD-ROM), 771
- AVEDEV function, 728
- AVERAGE function
 - array arguments, 394, 403
 - box plots using, 453
 - cells, blank, 402–403
 - described, 728
 - zero values, 402–403

- AVERAGEA function, 728
- averages
- array arguments, 394, 403
 - arrays, calculating using, 394, 402–403
 - box plots using, 453
 - cell range values, non-zero, 402–403
 - cells, blank, 402–403
 - database entries, of, 258, 717
 - grade point averages, 225
 - growth rate, 340–341
 - growth rate, annual, 298, 299–300
 - growth rate, geometric, 346–347
 - lookup formulas, in, 221, 225
 - mean, geometric, 729
 - mean, harmonic, 729
 - payments, time-weighted, 338
 - return, calculating average, 302, 340–341
 - trendline moving averages, 474
 - zero values in, 402–403
- B**
- backsolving, 53
- bank accounts.xls (on the CD-ROM), 772
- bank fees, 317–318
- basic counting.xls (on the CD-ROM), 767
- basic lookup examples.xls (on the CD-ROM), 767
- BESSEL functions, 719, 728
- binary numbers, converting to/from, 719, 720
- BINOMDIST function, 728
- Boolean (TRUE/FALSE) logical values, working with. *See* comparison operations, logical
- borders, 76, 451, 515
- box plots, 453–455. *See also* charts
- box plot.xls (on the CD-ROM), 771
- breakpoints in Visual Basic for Applications (VBA) code, 614, 615, 616
- Bricklin, Dan, 4
- Bullen, Stephen, 463, 764
- bullen function plotter.xls (on the CD-ROM), 771
- Byg Software, 582
- C**
- cache, pivot table, 501
- calculated field and item.xls (on the CD-ROM), 772
- calculation mode, 40–41, 361, 430–431
- calculator, using formula bar as, 35
- calendar array.xls (on the CD-ROM), 770
- calendars, 159, 166–167, 417–418
- case conversion operations, 98, 119, 126–127, 631, 693
- case sensitivity
- comparison operations, 120, 129, 131
 - counting operations, 187, 193
 - lookup formulas, 222
 - names, 60
 - variables, 596
- Visual Basic for Applications (VBA), 596, 603
- cash flows
- Discounted Cash Flow (DCF), 363–364
 - inflows, 329
 - Internal Rate of Return (IRR) calculation, 339–340, 341, 366–367
 - Net Present Value (NPV), affect on, 330
 - Net Present Value (NPV), calculating for flows beginning at end of first period, 332
 - Net Present Value (NPV), calculating for flows in advance, 333–334
 - Net Present Value (NPV), calculating for flows initially positive, 332–333
 - Net Present Value (NPV), calculating for flows series, 331
 - Net Present Value (NPV), calculating for flows with rate time period mismatch, 335–336
 - Net Present Value (NPV), calculating for flows with terminal values, 333–335
 - Net Present Value (NPV), calculating for irregular, 366, 367–368
 - Net Present Value (NPV), calculation of accumulation at different rates, 324, 346–347
 - Net Present Value (NPV), calculation of sums of irregular, 325, 366, 367–368
 - Net Present Value (NPV), calculation of sums of regular, 324, 329–338
 - outflows, 329
 - ownership, 296
 - Point 0, 330
 - rates, calculation of multiple from regular, 324, 342–345
 - rates, calculation of single from irregular, 325, 366–368
 - rates, calculation of single from regular, 324
 - rates, discount, 330, 331
 - rates, hurdle, 330
 - rates, monthly effective, 330
 - rental situations, 333, 336
 - schedules, 324–325, 351
 - schedules, discounted, 363–364
 - signing, 295–296, 337
- CD-ROM with this book. *See also* CD-ROM with this book, example workbooks
- character set macros, 118
 - charts utility, 446
 - Compare Sheets or Ranges utility, 581
 - database resources, 248, 249, 263
 - date utilities, 148, 159, 581
 - error-checking formulas, 301
 - Excel 2002 Formulas* in PDF format, 776

continued

- CD-ROM with this book *continued*
 - Extended Date Functions add-in, 148, 679
 - file guide to, 766–773
 - Financial Functions add-in, 314–316
 - Formula Report utility, 581
 - Insert-A-Date utility, 159
 - JWalk Enhanced Data Form add-in, 248
 - measurement unit conversion tables, 272
 - Name Lister utility, 64, 581
 - Perpetual Calendar utility, 159
 - Power Utility Pak, 21, 765, 773–774
 - Reminder Alarm utility, 159
 - saving files to hard drive, 765
 - Time Tracker utility, 159
 - VBA Project Summary Report utility, 581
 - Workbook Link Report utility, 581
 - Workbook Summary Report utility, 581
 - Worksheet Map utility, 581
- CD-ROM with this book, example workbooks
 - accumulation calculations, 297, 301, 361
 - amortization calculations, 308, 356
 - array formulas, 226, 397
 - cash flow, 331, 363, 366
 - CELLTYPE function, 660
 - charts, animated, 482
 - charts, AutoFiltering, 456
 - charts, circle plotting, 485
 - charts, clock, 484
 - charts, combo boxes for user input, 462
 - charts, comparative histograms, 452
 - charts, drawing in XY, 487
 - charts, function plotting, 463
 - charts, Gantt, 450
 - charts, hypocycloid curves in, 468
 - charts, interactive, 461, 462, 463
 - charts, progress toward goal, 449
 - charts, storing multiple on worksheet, 479
 - charts, trendlines, 471, 475
 - charts with data series automatic update, 457
 - charts with linked pictures, 446
 - circular references, 429, 432, 436
 - COMMISSION function, 667
 - conditional formatting, 521
 - counting formulas, 182, 188, 192, 194
 - credit card, 365, 556
 - data validation, 538
 - date calculation, 155, 161, 162, 163, 167
 - depreciation, 349
 - discounting problem, 304, 307
 - equation, 284, 436
 - frequency distribution, 195
 - histogram, 124, 200
 - indices, 371
 - LASTINCOLUMN function, 680
 - LASTINROW function, 680
 - loan calculations, 316, 320, 370
 - lookup formulas, two-way, 226
 - lookup formulas returning closest match, 230
 - lookup formulas returning item position in range, 230
 - lookup formulas using arrays, 226
 - lookup formulas using INDEX function, 222
 - lookup formulas using linear interpolation, 232
 - lookup formulas using MATCH function, 222
 - lookup formulas using two-columns, 228
 - lookup formulas using VLOOKUP function, 214, 222, 223, 224
 - megaformula, 542, 547, 552, 556
 - MONTHNAMES function, 688
 - MYSUM function, 697
 - Net Present Value (NPV), 331
 - pivot table, 490
 - RANDOMINTEGERS function, 690
 - RANGERANDOMIZE function, 692
 - right triangle formula, 280
 - sales commissions calculation, 667
 - SPELLDOLLARS function, 674
 - STATFUNCTION function, 661
 - summary data table, 263, 358
 - summing formula, 203, 206
 - text manipulation, 547, 668
 - time calculation, 172, 178
 - Visual Basic for Applications (VBA)
 - counting functions, 675
 - Visual Basic for Applications (VBA) date functions, 677
 - Visual Basic for Applications (VBA) multisheet functions, 682
 - Visual Basic for Applications (VBA) simple functions, 654
 - Visual Basic for Applications (VBA) summing functions, 675
 - Visual Basic for Applications (VBA) text functions, 668
 - VMONTHNAMES function, 688
- CEILING function, 285, 287, 726
- CELL function, 110, 111, 117, 723
- cell names. *See also* names
 - case sensitivity, 60
 - changing, 76
 - characters, prohibited, 60
 - checking, 62–63, 64
 - column deletion, behavior during, 77–78
 - column insertion, behavior during, 77
 - copy/paste operations, 32, 64, 68, 78–79
 - creating, 58–64
 - deleting, 75, 565
 - formulas, applying automatically when creating, 74
 - formulas, applying to existing, 73–74
 - formulas, entering in, 32, 69
 - formulas, using in, 69

- function arguments, as, 100
- introduced, 18
- length, maximum, 60
- lists of, creating, 68–69
- multiple for same cell, 59
- natural language formulas *versus*, 71–72
- navigating using, 58
- redefining, 76
- returning, 648
- row deletion, behavior during, 77–78
- row insertion, behavior during, 77
- rules, 60
- scope, 66–67
- spaces in, 60
- unapplying, 74–75
- workbook-level, 67–68, 69
- workbooks, referencing in other, 67
- worksheet deletion, during, 79
- worksheet-level, 67–68, 69
- cell ranges
 - addresses, fixed, 89–90
 - addresses, returning, 405, 647
 - arrays, contracting ranges containing, 385
 - arrays, creating from values in, 386–387
 - arrays, expanding ranges containing, 385
 - arrays, inserting cells in ranges
 - containing, 384
 - arrays, selecting ranges containing, 383–384
 - border display, 76
 - calendars, creating in, 166–167, 417–418
 - cells in, returning nonempty, 415, 680–681
 - cells in, reversing order of, 415–416
 - cells in, summing all, 201–202
 - cells in, summing visible, 210, 676–677
 - cells-used subset, returning, 651–652
 - characters in, counting, 391–392
 - charts, adding data labels by specifying, 446
 - charts, converting range references to arrays in, 445
 - charts, referencing in, 442, 444
 - charts, unlinking from, 444
 - columns in, returning, 649
 - columns in, returning number of, 648
 - columns in, testing for hidden, 649, 654–655
 - combining, 651
 - conditional formatting, applying, 518
 - conditional formatting, applying to duplicate values in, 526
 - conditional formatting, applying to maximum values in, 524
 - conditional formatting, applying to nonsorted values in, 527
 - copy/cut/paste operations, 15, 78, 566
 - data validation lists, specifying ranges for, 536
 - deleting, 75
 - differences between, counting, 404
 - dynamic, 91
 - error values, summing ranges
 - containing, 398
 - font properties, returning, 648, 658
 - hidden, testing for, 654–655
 - INDIRECT function, working with named ranges in, 88–90
 - intersections, 36, 70, 72
 - intersections, returning, 650–651
 - invalid, 64
 - lookup formulas returning item position in, 212, 217, 229–230
 - lookup formulas returning references to, 212
 - lookup formulas returning values from, 212, 216–219
 - looping through using For Each-Next, 643–644
 - multisheet, 65–66
 - names, 18
 - names, applying to existing formulas, 73–74
 - names, as function arguments, 57–58, 100
 - names, automatically created, 61–63, 74
 - names, behavior during column/row deletion, 77–78
 - names, case sensitivity, 60
 - names, changing, 76
 - names, checking, 62–63, 64
 - names, copy/paste operations, 32, 64, 68, 78–79
 - names, creating, 58–64
 - names, deleting, 75, 565
 - names, displaying, 76–77
 - names, entering in formulas, 69
 - names, for noncontiguous, 59
 - names, in macros, 58
 - names, lists of, 68–69
 - names, multiple for same range, 59
 - names, multisheet, 65–66
 - names, natural language formulas *versus*, 71–72
 - names, navigating using, 58
 - names, redefining, 76
 - names, referencing in other workbooks, 67
 - names, returning, 648
 - names, rules for, 60
 - names, scope, 66–67
 - names, unapplying, 74–75
 - names, using in macros, 58
 - names, workbook-level, 67–68, 69
 - names, worksheet-level, 67–68, 69
 - noncontiguous, 59, 442
 - number format, returning, 648, 659
 - object variables, assigning to, 649–650
 - operator, range, 36, 72
 - pictures, linked, 19

continued

- cell ranges *continued*
 - properties, Visual Basic for Applications (VBA), 644–649
 - random number generation in, 433–434, 689–690
 - randomizing, 691–692
 - reference operator, 36, 101
 - references, absolute, 42, 567–568
 - references, color, 567
 - references, combining, 36
 - references, copy/paste operations, 78
 - references, full-column, 101
 - references, full-row, 101
 - references, in SERIES formulas, 442
 - references, in Visual Basic for Applications (VBA), 644–646
 - references, indirect, 88–89
 - references, intersection, 36, 70, 72
 - references, mixed, 42, 88
 - references, natural language formulas versus, 71–72
 - references, operators used for, 36
 - references, relative, 42, 87, 567–568
 - references, returning number of areas in, 725
 - references, returning number of columns in, 725
 - references to ranges in other workbooks, 44–45, 68
 - references to ranges in other worksheets, 44–45
 - references, using array constants in lieu of, 395–396
 - references, using in named formulas, 84–85
 - rows, returning, 649
 - rows, returning number of, 648
 - rows, testing for hidden, 649, 654–655
 - selecting, 14, 60–61
 - text strings, returning longest in, 405
 - validation checks in, 406
 - values in, counting non-zero, 403
 - values in, counting unique, 193–195
 - values in, determining closest, 410
 - values in, determining largest, 432–433
 - values in, finding, 403–404
 - values in, finding nth occurrence of, 405
 - values in, returning average of non-zero, 403
 - values in, returning location of maximum, 404–405
 - values in, returning maximum, 404–405
 - values in, returning maximum using LARGE function, 100
 - values in, returning maximum using MAX function, 98, 433
 - values in, returning most frequently occurring, 191
 - values in, returning positive-only, 414–415
 - values in, returning unique, 416–417
 - values in, sorting dynamically, 416
 - values in, summing based on different ranges, 206–207
 - values in, summing every nth, 409–410
 - values in, summing largest, 204, 402
 - values in, summing smallest, 204, 392
 - values in, summing squares of, 643–644
 - viewing named, 76–77
 - Visual Basic for Applications (VBA), working with in, 643–652
 - worksheets, spanning multiple, 65–66
 - worksheets containing, returning, 647
- cell references. *See also* circular references
 - A1 notation, 43–44
 - absolute, 42
 - color, 567
 - column number of, returning, 725
 - combining, 36
 - conditional formatting formulas, in, 517, 518, 520–521
 - database operations, in, 239, 256
 - entering, 31, 32, 45
 - formula elements, as, 29
 - formulas, copied, 45
 - formulas, using in named, 84–87
 - indirect, 88–89
 - intersection, 36, 70, 72
 - invalid, 50, 566
 - megaformulas, substituting formula text for in, 544
 - mixed, 42, 88
 - multiple, combining into one, 36
 - names, using instead of, 58
 - natural language formulas *versus*, 71–72
 - number format correspondence, 31
 - operators used for, 36
 - R1C1 notation, 43–44
 - ranges, cells in multicell named, 73
 - relative, 42, 85–87
 - Visual Basic for Applications (VBA) formulas, in, 645
 - workbooks, in other, 44–45, 68
 - worksheets, in other, 44–45
- CELLFORMULA function, 654
- CELLHASFORMULA function, 654
- CELLHASTEXT function, 137, 671–672
- CELLSHIDDEN function, 654–655
- cells
 - arrays, changing cell contents in, 384
 - arrays, selecting cells containing, 383–384
 - auditing by identifying cells of particular type, 573
 - characters in, maximum, 113–114
 - charts, linking to, 445–448
 - charts, plotting hidden, 457
 - charts, single-cell, 448
 - color, background, 16, 524, 528–529

- conditional formatting, applying shading using, 515, 517, 525–526
- conditional formatting, applying to cells containing above-average values, 522
- conditional formatting, applying to cells containing formulas, 530–531
- conditional formatting, applying to cells containing invalid data, 532–533
- conditional formatting, applying to cells containing link formulas, 532
- conditional formatting, applying to cells containing more than one word, 528
- conditional formatting, applying to cells containing specific characters, 528
- conditional formatting, applying to cells containing text, 521
- conditional formatting, applying to named, 521
- conditional formatting, applying to specific cells, 518
- conditional formatting, copying cells containing, 519
- conditional formatting, hiding contents using, 524, 528–529
- conditional formatting, locating cells containing, 520
- copy operations, 15, 79
- counting blank, 180, 183–184
- counting cells between two values, 675
- counting cells containing dates, 187, 210
- counting cells containing specific formatting, 210
- counting cells containing specific numeric value, 180
- counting cells meeting AND criteria, 190–191
- counting cells meeting multiple criteria, 189–190
- counting cells meeting OR criteria, 190–191
- counting cells meeting specified criteria, 180, 186–191, 192–193, 210
- counting characters in, 130
- counting nonempty, 180, 184, 411, 646
- counting nontext, 185
- counting numeric, 185
- counting text cells, 185, 187, 392–393
- counting total in range, 183, 644, 647
- counting total in worksheet, 645
- counting visible, 210, 675–676
- counting words in, 135–136
- data stored in, 14
- data type, determining, 110, 115–117, 137, 210, 659–660
- deleting, 75, 566
- dependents, 575, 576–577
- editing contents, 34
- empty, allowing/disallowing using data validation, 537
- empty, determining if, 110
- empty, filling using Paste Special, 245
- empty, in database operations, 241
- empty, in lookup formulas, 219
- empty, in net present value calculations, 339
- empty, in pivot tables, 501
- empty, spaces in, 563
- formatting, stylistic, 16–17
- formatting information, returning, 648, 657–659
- formulas, identifying cells containing, 530–531, 573, 647, 654
- formulas in, returning, 646–647
- line breaks in, 15, 119–120
- locking, 25, 49
- number format, counting cells containing specific, 210
- number format, displaying, 756
- number format, filling with repeating characters using, 756–757
- number format, padding with dashes using, 756–757
- number of, maximum, 9
- precedents, 575–576
- preformatting to avoid automatic number formatting, 734, 740
- returning nonempty, 415, 680–681
- selecting, 14, 16, 60–61
- selecting cells containing arrays, 383–384
- selecting random, 663–664
- shading, 515, 517, 525–526
- summing cells containing specific formatting, 210
- summing visible cells, 210, 676–677
- text, determining if cell contains, 115–117, 137, 671–672
- tracing, 575–577
- wrap, 9, 15, 120–121
- zero values in, 184, 219
- Cells property, 645–646
- CELLSINCOMMON function, 650–651
- CELLTYPE function, 659–660
- celltype function.xls (on the CD-ROM), 773
- CHAR function, 118–119, 732
- character codes, 117–118
- character set.xls (on the CD-ROM), 766
- characters, special, 119
- Chart ⇄ Chart Type, 443
- chart data in active row.xls (on the CD-ROM), 771
- chart from combo box.xls (on the CD-ROM), 771
- Chart ⇄ Location, 479
- Chart Location dialog box, 479
- Chart Options dialog box, 445, 447
- Chart Wizard, 448–449, 450–451, 454–455

- charts. *See also* SERIES formulas
- 3-D, 466–467
 - activating, 20, 444
 - Analysis ToolPak add-in features, 199
 - animation, 481–482
 - arrays, converting data range
 - references to, 445
 - AutoFiltering, 455–456
 - axes labels, 442, 451
 - axes number formats, custom, 452
 - axes scale values, adjusting, 451
 - axes tick marks, removing, 452
 - bar charts, clustered, 452
 - bar charts, creating Gantt charts from, 450
 - bar charts, stacked, 450
 - borders, 451
 - box plots, 453–455
 - category axis titles, 445
 - cell range names, referencing in, 77
 - cells, charts in single, 448
 - cells, linking to, 445–448
 - cells, plotting hidden, 457
 - circles, plotting, 485–486
 - clock charts, 483–485, 486
 - color settings, 443, 455, 467, 515
 - column charts, 448–449
 - columns, plotting data in hidden, 455
 - combo box, data selection from, 461–462
 - COS function in, 483–484
 - COUNTA function, using in, 457, 458
 - curve fitting, 474
 - data labels, adding by specifying range, 446
 - data labels, editing, 446
 - data labels, linking to cells, 446
 - data labels, selecting, 446
 - data points, changing worksheet values by
 - dragging, 480–481
 - data points, displaying most recent, 458–459
 - data points, missing, 444, 454
 - data points, plotting every nth, 455–457
 - data points, plotting last n, 458–459
 - data series, 441
 - data series, deleting from, 444
 - data series, selecting points in, 443
 - data series, updating automatically, 457–458
 - data tables in, 447
 - date format, 451
 - dynamic, 444, 457–458
 - embedded, 10
 - embedded, printing, 443
 - embedded, relocating, 479
 - embedded, removing from printing, 443
 - embedded, selecting, 444
 - embedded, viewing in window, 480
 - EVALUATE function, using in, 465
 - extrapolation, 474
 - fill color, 443
 - font color, 443, 515
 - forecasting using trendlines, 469, 473
 - formulas, working with named, 444, 460–461
 - free-floating, 20
 - frequency distributions, charting, 196–197, 198–200
 - functions, plotting double-variable, 466–467
 - functions, plotting single-variable, 462–466
 - Gantt charts, 449–451
 - gap width, 449, 452
 - gauge charts, 482
 - goal seeking, 481
 - goals, charting progress toward, 448–449
 - histograms, 123–124, 198–200, 451–452
 - hypocycloid curves, 468–469
 - interactive, 460–467
 - interpolation, 454, 474
 - legends, 442, 467
 - line charts, continuing line through points
 - with no data, 444
 - line charts with markers, 454
 - line color, 455
 - location, 479
 - names, using in, 77, 442, 444
 - naming, 80
 - NOW function, updates using, 466
 - number formats, custom, 452
 - OFFSET function, using in, 457, 458–459, 464–465
 - overlap, 452
 - percent values, displaying as data series, 448–449
 - pictures, linking to cells, 447–448
 - pictures, pasting as, 444
 - pie charts, 482
 - PivotChart Report, 495
 - plot area, 443
 - plotting order, specifying, 442
 - population pyramids, 451
 - positioning, 444
 - Power Utility Pak tools, 446
 - printing chart sheets, 10
 - printing embedded charts, 443
 - printing without associated worksheet, 443
 - properties, changing, 444
 - quartile plots, 453–455
 - range, unlinking from, 444
 - range names, referencing, 442
 - range references, 442, 444
 - range references, converting to arrays, 445
 - range references, noncontiguous, 442
 - row, plotting based on active, 460–461
 - row, plotting data in hidden, 455

- scatter, 462, 469
- selecting, 20, 444
- selecting elements in, 443
- SERIES formulas, relation to, 441–442
- shading, 467
- sheets, 8, 10
- SIN function in, 462, 483–484
- single-cell charts, 448
- sizing/resizing, 443, 449
- static, 444
- surface charts, formatting, 467
- surface charts, plotting double-variable functions using, 466
- surface charts, XYZ, 466
- text, linking to cells, 446–447
- text boxes in, 446–447
- “thermometer” type display, 448–449
- tick marks, 452
- titles, 442, 445, 449
- trendlines, 469
- trendlines, coefficient of determination, 471
- trendlines, data series *versus*, 470
- trendlines, decimal places setting, 471
- trendlines, exponential, 474, 477–478
- trendlines, intercept calculations, 470, 471–472
- trendlines, linear, 469, 470–471
- trendlines, logarithmic, 474, 475–476
- trendlines, moving average option, 474
- trendlines, nonlinear, 474–479
- trendlines, polynomial, 474, 478–479
- trendlines, power, 474, 476–477
- trendlines, predicted values calculations, 472–473
- trendlines, R-squared values, 470, 471, 474
- trendlines, slope calculations, 471–472
- type, setting default, 443
- value axis titles, 445
- variables, plotting functions with double, 466–467
- variables, plotting functions with single, 462–463
- Visual Basic for Applications (VBA) macros, using in, 461, 484–485
- windows, sizing to, 443, 479
- windows, viewing embedded charts in separate, 480
- worksheets, 8, 10
- worksheets, changing values by dragging data points, 480–481
- worksheets, referring to other, 442
- worksheets, storing multiple charts on, 479–480
- XY charts, animated, 481
- XY charts, circle plotting in, 485
- XY charts, clock display in, 483
- XY charts, drawing using, 486–487
- XY charts, function plotting using, 463
- XY charts, hypocycloid curve display in, 468
- XY charts, scatter, 462, 469
- checksum digits, 552
- CHIDIST function, 728
- CHIINV function, 728
- CHITEST function, 728
- CHOOSE function, 212, 725
- Chr function, 621
- circle.xls (on the CD-ROM), 771
- Circular Reference toolbar, 51–52, 426–427, 577
- circular references
 - accidental, 51–52, 426–427, 560, 572, 577
 - described, 51, 425–426
 - Do-Loop constructs, 428
 - entering, 51–52
 - equations, solving recursive using, 435–436
 - equations, solving simultaneous using, 436–438
 - error message, 51–52, 426
 - error message, turning off, 52
 - IF function in, 122
 - indirect, 427
 - intentional, 428–430
 - iteration settings, 52, 427, 428–429, 431, 439
 - locating, 426
 - NOW function in, 122
 - random number generation using, 433–434
 - time stamping cell entries using, 432
 - value, determining highest using, 432–433
- circular reference.xls (on the CD-ROM), 770
- CLEAN function, 125–126, 732
- clock chart vba.xls (on the CD-ROM), 771
- clock chart.xls (on the CD-ROM), 771
- clocks, creating in charts, 483–485, 486
- closest match.xls (on the CD-ROM), 767
- CODE function, 118, 732
- color
 - cell backgrounds, 16, 524, 528–529
 - cell references, 567
 - cell shading, 515, 517, 525–526
 - charts, 443, 455, 467, 515
 - conditional formatting, applying using, 515, 516, 517, 525–526, 528–529
 - error values, hiding using, 524
 - fills, chart, 443
 - fills, returning color index number for, 658–659
 - fills, text box, 16
 - font, 443
 - modifying available, 516
 - number formats, custom, 518, 739, 751–752
 - palette, 516
 - workbooks, 516
- Columbus Day, calculating, 164

- COLUMN function, 725
- columns. *See also* worksheets
 - cells, returning last nonempty in, 680–681
 - database columns, mixing data types
 - in, 239
 - database columns as fields, 238
 - deleting, 75, 77–78
 - function arguments, as, 101
 - inserting, cell name behavior during, 77
 - labels, database column, 239
 - labels, using in formulas, 71–72
 - letters, returning for values contained in
 - cells, 132
 - naming, 63
 - number of, maximum, 9
 - numbers, returning, 725
 - pivot table column headers, 492
 - pivot table column orientation, 492
 - pivot table column titles, 501
 - returning last value in, 410–411
 - shading using conditional formatting, 525
 - splitting, 136
 - summing values in meeting
 - specified criteria, 180
 - testing for hidden, 649, 654–655
 - values in, determining largest, 432–433
 - width, changing, 9
 - width, insufficient, 50, 562
- COLUMNS function, 111, 183, 725
- Columns property, 648
- COM automation, retrieving real-time data from
 - programs supporting, 725
- COMBIN function, 726
- COMMISSION function, 665–667
- commission function.xls (on the CD-ROM), 773
- comp.apps.spreadsheets FAQ site, 764
- comparative histogram.xls (on the CD-ROM), 771
- Compare Sheets or Ranges utility (on the CD-ROM), 581
- comparison operations
 - case sensitivity, 120, 129, 131
 - equal to, 34, 36, 119–120, 719
 - greater than, 36
 - greater than or equal to, 36
 - less than, 36, 37
 - less than or equal to, 36, 37
 - logical, 36, 110, 202, 208–209
 - not equal to, 37
 - summing values based on date
 - comparisons, 207
 - summing values based on text
 - comparisons, 207
 - text comparisons, 119–120, 130–131, 137, 207
 - Visual Basic for Applications (VBA), 137, 629
- COMPLEX function, 719
- CONCATENATE function, 121, 732
- conditional formatting. *See* formatting, conditional
- Conditional Formatting dialog box
 - accessing, 514
 - Add button, 518
 - Cell Value Is button, 514, 516
 - Delete button, 520
 - Format button, 514, 515
 - Formula Is button, 514, 516
 - formulas, entering in, 517
- conditional formatting.xls (on the CD-ROM), 772
- Conditional Sum Wizard add-in, 205
- conditional summing.xls (on the CD-ROM), 767
- CONFIDENCE function, 729
- Consolidate_Area internal name, 64
- Const statements, 626
- constants
 - array constants, 378–379
 - array constants, creating from range values, 386–387
 - array constants, naming, 381–383
 - array constants, summing operations
 - involving, 378–379
 - array constants, using in calendars, 416
 - arrays constants, using in lieu of range
 - references, 395–396
 - equations, in, 283
 - naming, 81–83
 - numeric, 81–82
 - scope, 626
 - text constants, 82–83
 - Visual Basic for Applications (VBA), 626–627
- Control Toolbox, 19
- controls, dialog box, 19–20
- conversions, measurement unit. *See* measurement unit conversions
- CONVERT function, 269, 719
- Convert Text to Columns Wizard, 136
- Convert utility, 278
- copy/cut/paste operations
 - cell contents, 15
 - cell ranges, 15, 78
 - cells, 15, 78–79, 566
 - conditional formatting considerations, 519
 - formulas, 45–46, 566
 - keyboard shortcuts, 14, 15
 - name considerations, 32, 64, 68, 78–79
 - pivot tables, 504
 - text, copying from formulas, 544
 - Visual Basic for Applications (VBA) code, 598
 - worksheets, 78–79
- CORREL function, 729

- COS function, 483–484, 726
 COSH function, 726
 COUNT function, 180, 729
 Count property, 647
 count unique.xls (on the CD-ROM), 767
 COUNTA function
 array formulas, in, 411
 cells, counting non-empty using, 180, 184, 411, 646
 charts, in, 457, 458
 described, 729
 formulas, in dynamic named, 92
 COUNTBETWEEN function, 210, 675
 COUNTBLANK function, 180, 183–184, 729
 counters, 615, 639–640
 COUNTIF function, 180, 186–193, 403, 539, 729
 counting
 AND/OR criteria in, 189–191
 array formula involvement in, 183
 AutoCalculate, using, 182
 case sensitivity in, 187, 193
 cell ranges, differences between, 404
 cells, blank, 180, 183–184
 cells, nonempty, 180, 184, 411, 646
 cells, nontext, 185
 cells, numeric, 185
 cells, text, 185, 187, 392–393
 cells, total in range, 183, 644, 647
 cells, visible, 210, 675–676
 cells, words in, 135–136
 cells between two values, 675
 cells containing dates, 187, 210
 cells containing specific formatting, 210
 cells containing specific numeric values, 180
 cells meeting AND criteria, 190–191
 cells meeting multiple criteria, 188–191
 cells meeting OR criteria, 190–191
 cells meeting specified criteria, 180, 186–191, 192–193, 210
 characters in cell ranges, 391–392
 characters in cells, 130
 characters in text strings, 126, 130, 192–193
 database operations involving, 110, 180, 181–182, 242–243
 day of week, occurrences of, 161–162
 error values in ranges, 185–186, 398–399
 frequency distributions, 195–201
 functions related to, 180–181, 210, 674–676
 invalid items, 406
 logical values, 185
 most frequently occurring value, 191
 pivot tables, in, 181, 195, 494
 substring occurrences, 130–131
 text string occurrences, 130–131, 192–193
 text strings, words in, 135–136
 values, non-zero, 403
 values, unique, 193–195
 Visual Basic for Applications (VBA) functions, using, 210, 674–676
 wildcard characters in, 186
 words in cells, 135–136
 counting and summing functions.xls (on the CD-ROM), 773
 counting text in a range.xls (on the CD-ROM), 767
 COUNTVISIBLE function, 210, 675–676
 COUPDAYBS function, 721
 COUPDAYS function, 721
 COUPDAYSNC function, 721
 COUPNCD function, 721
 COUPNUM function, 721
 coupon period calculations, 721
 COUPPCD function, 721
 COVAR function, 729
 Create Names dialog box, 61–62
 creating indices.xls (on the CD-ROM), 769
 credit card calculations.xls (on the CD-ROM), 769
 credit card validation.xls (on the CD-ROM), 772
 credit cards
 American Express, 552
 checksum digits, 552
 Discover, 552
 Mastercard, 552
 minimum payment considerations, 365
 number validity formulas, 552–556
 payment calculations, 324, 365–366
 Visa, 552
 CRITBINOM function, 729
 Criteria internal name, 64
 cross-checking formulas for accuracy, 300
 crosstab tables, dynamic, 413–414
 CUMIPMT function, 322–323, 543, 721
 CUMPRINC function, 322–323, 721
 cumulative sum.xls (on the CD-ROM), 767
 currency
 dollars, fractional, 285, 288, 721
 dollars, rounding, 287
 format, 15, 16, 733, 735, 736
 fraction/decimal conversions, 285
 spelling out, 137, 674
 text, displaying as, 122–123
 cut/paste operations. *See* copy/cut/paste operations
 CVer function, 686
- ## D
- Data Analysis dialog box, 198–199
 data entry. *See also* validation, data array formulas, 30, 376, 377, 383
 AutoFill, using, 15, 48
 databases, in, 248

continued

- data entry *continued*
 - formula results, 35
 - formulas, 30–32, 39–40
 - functions, 103–107
 - introduced, 14–15
 - parentheses, 39
 - Visual Basic for Applications (VBA) code, manually, 595–596
 - Visual Basic for Applications (VBA) code using macro recorder, 596–598
- Data ⇄ Filter ⇄ Advanced Filter, 22
- Data ⇄ Filter ⇄ AutoFilter, 22, 240
- Data ⇄ Filter ⇄ Show All, 242
- Data ⇄ Form, 248
- Data Form feature, 248, 497
- Data ⇄ JWalk Enhanced Data Form, 248
- Data ⇄ PivotTable, 495
- data recovery using link formulas, 45
- Data ⇄ Sort, 22
- Data ⇄ Subtotals, 107
- Data ⇄ Table, 261
- data table summary.xls (on the CD-ROM), 768
- data tables, summary
 - array formulas in, 264, 359
 - calculation mode considerations, 361
 - creating, 263, 358–361
 - crosstab tables, dynamic, 413–414
 - dynamic, 261
 - frequency distributions, 195
 - loan option summaries, 358–361
 - one-way, 358–360
 - pivot tables, using, 261
 - speed considerations, 361
 - two-way, 360–361
- Data ⇄ Text to Columns, 136
- data types
 - arrays, returned by, 687
 - Boolean, 624
 - byte, 624
 - currency, 624
 - database columns, mixing data types in, 239
 - date, 624, 627–628
 - decimal, 624
 - determining, 110, 115–117, 137, 210, 659–660
 - displaying, 615
 - double, 624
 - errors, debugging, 567
 - integer, 624
 - long, 624
 - object, 624
 - single, 624
 - string, 624, 627
 - variant, 624, 687
 - Visual Basic for Applications (VBA), 624
 - Visual Basic for Applications (VBA), automatic assignment, 623
 - Visual Basic for Applications (VBA), declaring in, 625
 - Visual Basic for Applications (VBA), determining using, 659–660
 - Visual Basic for Applications (VBA) functions, specifying type returned by, 603, 620
- data validation. *See* validation, data
- Data ⇄ Validation, 15
- Data Validation dialog box
 - accessing, 534
 - Error Alert tab, 535, 537, 538
 - Input Message tab, 535
 - Settings tab, 534, 537, 538
- data validation.xls (on the CD-ROM), 772
- Database internal name, 64
- databases (worksheet lists)
 - array formula operations, 260, 264
 - arrays, using in filter criteria formulas, 257–258
 - arrays, using in place of criteria range, 260, 395
 - averages, 258, 717
 - cell references, 239, 256
 - cells, blank, 241
 - columns, labeling, 239
 - columns, mixing data types in, 239
 - columns as fields, 238
 - counting operations, 110, 180, 181–182, 242–243
 - data entry/editing, 248
 - date format considerations, 254
 - design, 239
 - field labels, using in filtering, 247, 256
 - fields, columns as, 238
 - fields, maximum, 238
 - filtering, 22, 240
 - filtering, comparison operators in, 250–251
 - filtering, copying data resulting from, 243
 - filtering, count operations on resulting data, 242–243
 - filtering, data hidden by, 240, 242, 243
 - filtering, deleting data resulting from, 243
 - filtering, duplicate record removal using, 247
 - filtering, field labels in, 247, 256
 - filtering, logical comparisons in, 253–255, 256
 - filtering, range name of filtered list, 241
 - filtering, sum operations on resulting data, 242–243, 266
 - filtering, wildcards in, 251–252
 - filtering criteria, AND/OR, 246, 253–254, 257–258
 - filtering criteria, computed, 255–258

- filtering criteria, multiple,
 - 241, 246, 253–255
 - filtering criteria, specifying,
 - 240–241, 246, 248–258
 - filtering criteria, using arrays in place of
 - ranges, 260, 395
 - filtering using Advanced Filter
 - dialog box, 247
 - filtering using AutoFilter, 240–244
 - functions related to, 110, 258–259, 717–718
 - headings, 239
 - import data cleanup, 245
 - import data, Lotus, 261
 - multiplication, 717
 - pivot table background queries, 501
 - pivot table data sources, as, 496
 - pivot table summaries, 491
 - queries, background in pivot tables, 501
 - records, counting nonempty, 180
 - records, counting using specified criteria,
 - 110, 180, 717
 - records, extracting unique, 247, 717
 - records, maximum number of, 22, 238
 - records, removing duplicates, 247
 - records, rows as, 238
 - rows, empty, 239
 - rows, freezing first, 239
 - rows, hiding, 243, 244
 - rows, labeling, 239
 - sorting, 22
 - standard deviation calculations, 717
 - summing operations, 242–243, 259–260,
 - 264–267, 718
 - table relationships, 237
 - table selection, 14
 - tables, data, 261–264
 - uses of, 238–239
 - values, returning maximum/minimum, 717
 - variance calculations, 718
 - worksheets, storage in, 239
- databases, external, 22, 501
- DATATYPE function, 210
- DATE function
- arguments, invalid, 151
 - arguments, other functions as, 151
 - date calculations, in, 157, 160–166
 - date display using, 151
 - date formats, ensuring compatibility
 - using, 254
 - described, 149, 718
 - displaying dates using, 151
 - string conversions, in, 153
 - summing operations, in, 207
 - TIME function, using with, 169
- date functions.xls (on the CD-ROM), 773
- Date Report utility (on the CD-ROM), 159, 581
- DATEDIF function, 149, 158, 718
- dates
- 1582, prior to, 690
 - 1900, prior to, 147–148, 679–680
 - 1900 system, 140
 - 1904 system, 140, 173
 - add-ins, 112, 148, 149–150, 159
 - age calculations, 156–157
 - AutoFill, entering series using, 152–153
 - calendars, creating in ranges, 166–167,
 - 417–418
 - cells containing, counting, 187, 210
 - cells containing, referencing, 146
 - century, specifying, 148–149
 - chart date format, 451
 - Columbus Day, determining, 164
 - current, displaying, 150–151
 - current, entering, 15
 - current, returning, 109, 122, 523
 - data validation, 536
 - databases, format considerations in, 254
 - day of month, determining last, 165
 - day of week, counting occurrences, 161–162
 - day of week, determining,
 - 158–159, 160–161
 - day of week, determining date of next, 678
 - day of week, returning as integer, 679
 - day of week after specified date,
 - determining first, 160
 - day of year, determining, 157
 - days, adding to dates, 679
 - days, offsetting using work days, 156
 - days between two dates, calculating, 149,
 - 153–154, 158, 679, 718
 - days of dates, returning, 679
 - displaying, 150–152
 - Easter, determining, 165
 - entering, 15, 141–142, 148–149, 151
 - entering with time, 145
 - fence-post analogy, 154
 - format, applying, 140, 145–146, 734, 736
 - format, automatic, 146, 734
 - format, custom, 146, 741, 752
 - format, default, 141
 - format compatibility, 139, 152, 254
 - formats recognized, 141–142
 - formatting, conditional, 522–523, 531
 - functions related to, 109, 112, 148, 149–150,
 - 718–719
 - holidays, determining, 163–165
 - holidays, lists of, 156
 - Julian, 157
 - Labor Day, determining, 164
 - leap year bug, 147
 - Lotus 1-2-3 legacy, 147, 158
 - Macintosh date system, 140
 - Martin Luther King Jr. Day,
 - determining, 163

continued

- dates *continued*
 - Memorial Day, determining, 164
 - Monday, determining date of next, 677–678
 - month, determining last day of, 165
 - month, identifying dates in, 522
 - months, determining number between two dates, 158, 718
 - months of dates, returning, 679
 - ordinals, expressing as, 162
 - Power Utility Pak (on the CD-ROM) tools, 159, 581
 - Presidents' Day, determining, 164
 - quarter, determining, 166
 - range supported, 140
 - regional settings, 141
 - replacement characters for invalid, 50, 562
 - reports, 581
 - searching for, 143
 - serial numbers, converting text strings to, 151, 718
 - serial numbers, converting to days, 149, 150, 718, 719
 - serial numbers, converting to months, 150, 168, 718
 - serial numbers, converting to years, 150, 719
 - serial numbers, entering as, 141
 - serial numbers, returning for current date, 150, 719
 - serial numbers, returning for current date and time, 150, 168, 718
 - serial numbers, returning for specified dates, 149, 150, 718
 - serial numbers, viewing, 140
 - serial numbers, zero value, 145
 - series of, generating, 152–153
 - stamp, 151
 - summing values based on date comparisons, 207
 - Sunday, determining date of the most recent, 160
 - system settings, 141
 - text, converting to, 151, 153
 - text, returning, 157, 162
 - text recognized as, 141, 142, 147–148
 - Thanksgiving Day, determining, 164
 - time, day association with, 145
 - United States format, 139, 152
 - variables, storing in, 627–628
 - version differences, 148–149
 - versions of Excel, differences in, 140, 148–149
 - Visual Basic for Applications (VBA), working with in, 627–628, 677–680
 - week, determining day of, 158–159
 - week number in year, returning, 150, 719
 - week of month, returning integer for, 678–679
 - weekend dates, identifying, 523
 - workdays calculations, 150, 154–155, 718, 719
 - year, determining day of, 157
 - years, converting to Roman numerals, 166
 - years, leap, 147, 165–166
 - years, returning fractional values for date spans, 150, 719
 - years, two-digit, 147, 148–149
 - years between two dates, calculating, 156, 158, 679, 718
 - years of dates, returning, 679
- DATEVALUE function, 149, 151–152, 718
- DAVERAGE function, 258, 717
- DAY function, 149, 718
- day of the week count.xls (on the CD-ROM), 766
- DAYS360 function, 149, 718
- DB function, 348, 349, 721
- DCF. *See* Discounted Cash Flow (DCF)
- DCOUNT function, 110, 180, 181, 258, 717
- @DCOUNT Lotus function 707, 714–715
- DCOUNTA function, 180, 258
- DDB function, 348, 349, 721
- Debug ⇄ Compile, 610
- Debug ⇄ Toggle Breakpoint, 616
- debugging. *See also* auditing; errors in formulas
 - actual *versus* displayed values problems, 570
 - add-ins, missing, 565
 - array formulas, 567
 - breakpoints, using, 614, 615, 616
 - cell dependents, tracing, 576–577
 - cell precedents, tracing, 576
 - cell range intersection errors, 565–566
 - cell references, absolute/relative problems, 567–568
 - cell references invalid errors, 50, 566
 - circular references, 560, 572, 577
 - column width, insufficient, 562
 - data type errors, 567
 - Debug.Print statements, monitoring variables using, 612–613
 - described, 559
 - division by zero errors, 50, 564
 - error checking, background, 578–579
 - error values, identifying, 398, 399, 564
 - error values, tracing, 563, 577
 - floating point number errors, 571
 - Formula Evaluator, using, 580
 - formulas not calculated, 569
 - function arguments, non-numerical when numerical expected, 566
 - function arguments, range when single value expected, 467
 - functions, custom, 677
 - iteration errors, 566
 - link errors, 572

- loop constructs, monitoring values in, 611–613
 - missing data errors, 565
 - MsgBox statements, using, 610–612
 - name/range undefined errors, 565
 - operator precedence errors, 568–569
 - parentheses, mismatched, 561
 - programs *versus* formulas, 559
 - spaces in blank cells, accidental, 563
 - syntax errors, 561, 562, 596, 610
 - values too large/small errors, 566
 - variables, monitoring, 610–613
 - Visual Basic for Applications (VBA)
 - statement execution, viewing in Immediate window, 590
 - Debug.Print statements, monitoring variables using, 612–613
 - DEC2BIN function, 719
 - DEC2HEX function, 719
 - DEC2OCT function, 719
 - decimal numbers, converting to/from, 719, 720
 - Define Name dialog box
 - accessing, 58
 - name creation using, 58–59
 - name definition, changing using, 60, 76
 - name deletion using, 75
 - Names field, 59
 - names listed in, 64, 80
 - Refers to field, 59, 65–66, 80–81, 457, 458
 - Workbook field, 67
 - DEGREES function, 279, 726
 - Delete Conditional Formatting dialog box, 520
 - DELTA function, 719
 - dependents, cell, 575, 576–577
 - depreciation, 347
 - depreciation calculations
 - accounting period, 721
 - cost arguments, 348
 - declining balance, 348, 349, 721
 - double-declining balance, 348, 349, 721
 - factor arguments, 348
 - functions related to, 348
 - life arguments, 348
 - month arguments, 348
 - no-switch arguments, 348
 - rate arguments, 348
 - salvage arguments, 348
 - straight-line, 348, 349
 - sum of year's digits, 348, 349
 - variable-declining balance, 348, 349–350
 - depreciation.xls (on the CD-ROM), 769
 - detailed loan amortization schedule.xls (on the CD-ROM), 769
 - DEVSQ function, 180, 729
 - DGET function, 258, 717
 - Diagram Gallery, 18–19
 - dialog box controls, placing on worksheets, 19–20
 - dialog boxes, built-in, 12–13. *See also specific boxes*
 - dialog sheets, 8, 10
 - Dim keyword, 620, 625, 630–631
 - DISC function, 721
 - Discounted Cash Flow (DCF), 363–364
 - discounted cash flow schedule.xls (on the CD-ROM), 769
 - discounting calculations
 - accumulation compared, 304
 - complex, 304
 - Discounted Cash Flow (DCF), 363
 - Future Value (FV), 308
 - payment, 307, 309–313
 - Present Value (PV), 304–306, 307
 - rate, 306, 307–308
 - rate, discount, 306, 329–330, 339, 341–342, 721
 - simple, 304
 - display, customizing, 14
 - distance measurement unit conversions, 272, 273
 - #DIV/0! division by zero errors, 50, 564
 - DIVIDETWO function, 642–643
 - division operators, 36, 38, 60, 629
 - DMAX function, 259, 717
 - DMIN function, 259
 - Do Until constructs, 640–641
 - Do While constructs, 639–640
 - document shortcuts, 725
 - DOLLAR function, 122–123, 732
 - DOLLARDE function, 285, 288, 721
 - DOLLARFR function, 285, 288, 721
 - dollars
 - fractional, 285, 288, 721
 - rounding, 287
 - DPRODUCT function, 259, 717
 - drag-and-drop editing, 13
 - draw layer, worksheet, 10, 13, 18–20
 - Drawing toolbar, 18
 - DRAWONE function, 663–664
 - DSTDEV function, 259, 717
 - DSTDEVP function, 259, 717
 - DSUM function, 180, 181, 259–260, 718
 - DURATION function, 721
 - DVAR function, 259, 718
 - DVARP function, 259, 718
- ## E
- Easter, calculating, 165
 - EDATE function, 149, 718
 - Edit ⇄ Clear ⇄ Formats, 520
 - Edit ⇄ Copy Picture, 444
 - Edit ⇄ Paste Picture, 444
 - Edit ⇄ Paste Picture Link, 19
 - Edit ⇄ Paste Special, 46, 245

- Edit ⇄ Replace, 128
- editing formulas, 34–35
- EFFECT function, 315, 356, 722
- Effx_AnnEff function
 - described, 315
 - Discounted Cash Flow (DCF) schedule calculations, in, 364
 - formula equivalent, Excel, 315
 - loan calculations, in, 319–320
- Effx_Effy function, 315
- Effx_Nomx function, 315
- Effx_Nomy function, 315
- energy measurement unit conversions, 272, 276
- engineering functions, 110, 719–720
- entering formulas
 - array formulas, 30, 376, 377, 383, 385
 - at sign (@) prefix, 30
 - AutoCorrect, using, 39–40, 562
 - conditional formatting formulas, 517
 - equal sign (=) prefix, 30, 31
 - line breaks, 32
 - manual entry, 30
 - names, entering, 32, 69
 - pointing, by, 31
 - spaces, 32
 - syntax, 30
- EntireColumn property, 649
- EntireRow property, 649
- environment, returning information about, 723
- EOMONTH function, 149, 718
- equations
 - array formulas, using, 284
 - circular references, solving using, 435–438
 - coefficient matrix, inverse of, 284
 - coefficients, 283
 - constants, 283
 - formulas, converting to
 - self-referencing, 435
 - linear, 283
 - recursive, 435–436
 - simultaneous, solving, 283–284, 436–438
 - variables, 283
- ERF function, 720
- ERFC function, 720
- Error Checking dialog box, 579
- error values
 - counting, 185–186, 398–399
 - hiding, 524
 - listed, 50, 563
 - pivot table display, 501
 - printing options, 524, 563
 - referencing, 686
 - replacing with empty strings, 398
 - returning, 685–686, 723
 - summing ranges containing errors, 398
 - tracing, 563, 577
- errors in formulas. *See also* debugging
 - actual *versus* displayed values
 - problems, 570
 - add-ins, missing, 565
 - array formulas, 560, 567
 - background error checking, 578–579
 - bypassing by converting formulas to text, 35
 - cell range intersection, 565–566
 - cell reference errors, 50, 560, 566
 - cell references, absolute/relative problems, 567–568
 - circular references, 51–52, 426–427, 560, 572, 577
 - cross-checking to avoid, 300
 - data type incorrect, 567, 615
 - floating point number errors, 571
 - function arguments, non-numerical when numerical expected, 566
 - function arguments, range when single value expected, 467
 - functions, custom, 567
 - ignoring, 579, 641–642
 - incomplete calculation errors, 560
 - iteration errors, 566
 - link errors, 572
 - logic errors, 560, 610
 - missing data errors, 50, 565
 - name/range undefined, 565
 - operator precedence problems, 568–569
 - parentheses, mismatched, 561
 - prevalence, 560
 - recalculation missing, 569
 - ripple effect, 559
 - runtime, 610, 613–615
 - syntax errors, 560, 561
 - types, 49–50
 - values too large or too small errors, 566
 - Visual Basic for Applications (VBA) error handling, 641–643
 - zero, division by, 50, 564
- ERROR.TYPE function, 723
- Evaluate Formula dialog box, 580
- EVALUATE function, 465
- EVEN function, 285, 289–290, 726
- EXACT function, 732
- Excel 2002 Power Programming with VBA*, 588
- Excel Auditor utility, 582
- Excel Home Page, 760
- Excel Web Source Web site, 764
- EXCELVERSION function, 657
- EXP function, 726
- EXPONDIST function, 729
- exponentiation operators, 36, 38, 568, 629

- expressions
 - function arguments, as, 102
 - order of evaluation, controlling, 30, 38–40
 - Visual Basic for Applications (VBA), 628–629
 - Extended Date Functions add-in (on the CD-ROM), 148, 679
 - Extract internal name, 64
 - EXTRACTELEMENT function, 137, 672–673
 - extrapolation using charts, 474. *See also* interpolation
- ## F
- FACT function, 726
 - FACTDOUBLE function, 720
 - factorials, returning, 720, 726
 - FALSE function, 724
 - FAQ pages, comp.apps.spreadsheets newsgroup, 764
 - FDIST function, 729
 - fields
 - calculated, 506–511
 - columns as, 238
 - filtering, using in, 247, 256
 - labels, 247, 256
 - maximum, 238
 - pivot table fields, calculated, 506–511
 - pivot table fields, column, 492, 501
 - pivot table fields, customizing, 499
 - pivot table fields, drag and drop operations, 499, 502
 - pivot table fields, names in data source, 496
 - pivot table fields, ordering on page, 501
 - pivot table fields, page, 493
 - pivot table fields, row, 493
 - pivot table fields, summarizing non-numeric, 494
 - File ⇄ Page Setup, 563
 - File ⇄ Print, 443
 - File ⇄ Properties, 617
 - File ⇄ Remove, 593
 - File ⇄ Save, 45
 - File ⇄ Save As, 45
 - filenames, extracting from path string, 132–133
 - files, corrupt. *See* data recovery using link
 - formulas
 - FILLCOLOR function, 210, 658
 - filter operations, database
 - Advanced Filter dialog box, using, 247
 - arrays, using in criteria formulas, 257–258
 - arrays, using in place of criteria range, 260, 395
 - AutoFilter, using, 240–244
 - comparison operators in, 250–251
 - copying resulting data, 243
 - count operations on resulting data, 242–243
 - criteria, AND/OR, 246, 253–254, 257–258
 - criteria, computed, 255–258
 - criteria, multiple, 241, 246, 253–255
 - criteria, specifying, 240–241, 246, 248–258
 - data hidden by, 240, 242, 243
 - deleting resulting data, 243
 - described, 22, 240
 - duplicate record removal using, 247
 - field labels in, 247, 256
 - logical comparisons in, 253–255, 256
 - range name of filtered list, 241
 - sum operations on resulting data, 242–243, 266
 - wildcards in, 251–252
 - FilterDatabase internal name, 64
 - financial calculations. *See specific topics*
 - financial concepts, basic
 - amortization, 308
 - depreciation, 347
 - Future Value (FV), 294
 - interest period, 294
 - interest rate, 294
 - interest rate, annual effective, 314
 - interest rate, nominal, 313, 314
 - interest rate, periodic effective, 314
 - interest rate quote methods, 313–314
 - loans, flat rate, 318
 - money flow, 295–296
 - Net Present Value (NPV), 330
 - Payment (PMT), 294
 - Present Value (PV), 294
 - term, 294
 - financial functions, 109, 293–294, 324–326, 721–723
 - Financial Functions add-in (on the CD-ROM), 314–316
 - Find and Replace dialog box, 516
 - FIND function, 128–129, 732
 - FINV function, 729
 - FISHER function, 729
 - Fisher transformation, 729
 - FISHERINV function, 729
 - FIXED function, 732
 - floating point number errors, 571
 - FLOOR function, 285, 287, 726
 - FMT files, 699
 - font
 - character set for, displaying, 117–118
 - color, 443, 515, 518
 - counting cells containing specific formatting, 210
 - formatting, conditional, 515
 - options, data, 16
 - properties, returning, 648, 658
 - Font property, 648
 - For Each-Next constructs, 643–644
 - force measurement unit conversions, 272, 276
 - FORECAST function, 391, 473, 729
 - forecasting using chart trendlines, 469, 473

- Format ⇄ Cells, 16
- Format Cells dialog box
 - conditional formatting version, 515
 - described, 16–17
 - Number tab, 16, 145–146, 736–737
 - Protection tab, 25–26, 49, 755
- Format Chart dialog box, 444
- Format ⇄ Conditional Formatting, 514
- Format Control dialog box, 462
- Format Data Series dialog box, 446
- Format dialog box, 16
- Format Legend Key dialog box, 467
- Format ⇄ Row ⇄ Hide, 243
- Format ⇄ Style, 734
- Format Trendline dialog box, 473
- formatting, conditional
 - borders, 515
 - cell ranges, applying to, 518
 - cell ranges, applying to duplicate values in, 526
 - cell ranges, applying to maximum values in, 524
 - cell ranges, applying to nonsorted values in, 527
 - cell references in conditional formatting
 - formulas, 517, 518, 520–521
 - cells, applying to named, 521
 - cells, applying to specific, 518
 - cells, background patterns using, 515
 - cells, hiding contents using, 524, 528–529
 - cells, shading using, 515, 517, 525–526
 - cells containing above-average values, applying to, 522
 - cells containing formulas, applying to, 530–531
 - cells containing invalid data, applying to, 532–533
 - cells containing link formulas, applying to, 532
 - cells containing, locating, 520
 - cells containing more than one word, applying to, 528
 - cells containing specific characters, applying to, 528
 - cells containing text, applying to, 521
 - color, applying using, 515, 516, 517, 525–526, 528–529
 - column shading, 525
 - conditions, between two values, 516
 - conditions, cell value, 516, 518–519
 - conditions, equal to specified values, 516
 - conditions, formula-based, 517–518
 - conditions, greater than or equal to specified values, 516
 - conditions, greater than specified values, 516
 - conditions, less than or equal to specified values, 516
 - conditions, less than specified values, 516
 - conditions, not between two values, 516
 - conditions, not equal to specified values, 516
 - conditions, specifying multiple, 518–519
 - conditions, specifying simple, 516–518
 - copy/paste considerations, 519
 - dates, working with, 522–523, 531
 - deleting, 520
 - described, 513–514
 - duplicate values, applying to, 526
 - dynamic nature of, 514
 - entering conditional formatting
 - formulas, 517
 - error values, hiding using, 524
 - find and replace, exempted from, 516
 - finding, 520
 - font color, 515, 524
 - font size, 515
 - font strikethrough, 515
 - font style, 515
 - font underline, 515
 - formulas, based on, 517–518
 - formulas, custom functions in, 530–533
 - link formulas, applying to cells
 - containing, 532
 - nonnumeric data, identifying, 521
 - numbers, 518, 750–751
 - results, displaying only when all data entered, 528–529
 - rows, 525–526, 529
 - searching for cells containing, 520
 - shading, 515, 517, 525–526
 - trends, applying to, 527
 - types, applicable, 515–516
 - versions of Excel, differences in, 513
 - Visual Basic for Applications (VBA)
 - functions in conditional formatting
 - formulas, 530–533
 - worksheets, referencing other, 520–521
- Formatting toolbar, 734–735
- Forms toolbar, 461
- formula auditing. *See* auditing
- Formula Auditing toolbar, 537, 574, 576–577
- Formula AutoCorrect, 39–40, 562
- formula bar
 - array formulas in, 383, 384
 - calculator, using as, 35
 - editing formulas in, 34, 384
 - formula appearance in, 30
 - hiding formulas, 26
- Formula Evaluator feature, 580
- Formula Palette, 31–32, 34, 106
- Formula property, 646–647
- Formula Report utility (on the CD-ROM), 581

- Formula view, 573–575
- For-Next constructs, 620–621, 637–639, 646
- fractions
- entering, 15
 - number format, 737, 748–749
- Frankston, Bob, 4
- FreqName function, 352
- frequency distributions
- Analysis ToolPak add-in features, 198–199
 - array formulas, using in calculating, 196
 - bins, 195–196, 198, 200
 - charting, 196–197, 198–200
 - described, 195
 - FREQUENCY function, using, 195–196
 - pivot tables, 195, 489
- frequency distribution.xls (on the CD-ROM), 767
- FREQUENCY function, 180, 195–196, 391, 729
- FTEST function, 729
- Full Screen view, 14
- Function Arguments dialog box, 106
- Function keyword, 602
- functions. *See also* Visual Basic for Applications (VBA), functions; *specific functions*
- advantages of using, 97–99
 - Analysis ToolPak functions, 110, 112, 269, 285
 - arguments, 99–100
 - arguments, arrays as, 103, 388, 694
 - arguments, Boolean, 697
 - arguments, columns as, 101
 - arguments, descriptive placeholders for, 108
 - arguments, determining type, 185, 686, 697
 - arguments, expressions as, 102
 - arguments, literal, 102
 - arguments, maximum number of, 694
 - arguments, mixing type, 694
 - arguments, multiple in same function, 100
 - arguments, names as, 100–101
 - arguments, non-numerical when numerical expected, 566
 - arguments, optional, 693
 - arguments, other functions as, 102–103
 - arguments, range value when single value expected, 467
 - arguments, rows as, 101
 - arguments, value display, 108
 - arrays, returning, 391
 - Bessel functions, returning, 719
 - built-in, 17, 99
 - charts, plotting double-variable functions in, 466–467
 - charts, plotting single-variable functions in, 462–466
 - counting functions, 180–181, 210, 674–676
 - custom, 99, 599–600
 - database functions, 110, 258–259, 717–718
 - date functions, 109, 112, 148, 149–150, 718–719
 - depreciation, related to, 348
 - editing, 108
 - engineering functions, 110, 719–720
 - entering, 103–107
 - financial functions, 109, 293–294, 324–326, 721–723
 - formula elements, as, 30
 - formulas, using in named, 83–84
 - information functions, 110, 723–724
 - logical functions, 110, 400–401, 724
 - lookup functions, 110, 212, 725
 - math functions, 109, 726–728
 - matrix functions, 284
 - multiple-form, 108
 - nesting, 102–103
 - rate conversion, 315
 - reference, 110, 725
 - rounding functions, 285
 - searching for, 105, 108
 - securities, related to, 721, 722, 723
 - SERIES formulas, using in, 442
 - statistical, 109, 728–731
 - summing functions, 180–181, 210
 - text functions, 110, 115, 136–137, 732
 - time-related, 109, 168, 718–719
 - trigonometric functions, 109, 726–728
 - user-defined, 111, 588, 599
 - versions of Excel, function categories
 - inherited from previous, 111
 - Visual Basic for Applications (VBA) code, inserting Excel functions in, 632
 - volatile, 111
 - worksheet functions, 97
- Future Value (FV)
- accumulation calculations, 297–298, 299, 301–302, 303, 346–347
 - amortization calculations, 310, 312, 323–324
 - Analysis ToolPak add-in features, 346
 - defined, 294
 - discounting calculations, 308
 - Net Present Value (NPV), calculating from, 337
 - Present Value, calculating from, 324
- FV function
- accumulation calculations, 297–298, 299, 302
 - amortization calculations, 310, 312, 323–324
 - arguments, required, 293
 - credit card calculations, in, 365
 - described, 722
 - money flow, 295
 - time value of money concept used by, 295
- FVSCHEDULE function, 324, 346–347, 722

G

GAMMADIST function, 729
 GAMMAINV function, 729
 GAMMALN function, 729
 gantt chart.xls (on the CD-ROM), 771
 gauge chart.xls (on the CD-ROM), 771
 GCD function, 726
 GEOMEAN function, 729
 geometry. *See also* trigonometry
 area calculations, 280–281
 circles, calculations involving, 281
 cones, calculations involving, 282
 cubes, calculations involving, 282
 cylinders, calculations involving, 282–283
 degrees, converting radians to, 279
 hypocycloid curves, charting, 468–469
 perimeter calculations, 280–281
 pyramids, calculations involving, 283
 radians, expressing angles in, 103, 279, 720
 rectangles, calculations involving, 281
 spheres, calculations involving, 282
 squares, calculations involving, 280–281
 surface calculations, 282
 trapezoids, calculations involving, 281
 triangles, calculations involving,
 277–280, 281
 volume calculations, 282–283
 GESTEP function, 720
 GETPIVOTDATA function, 725
 GMT. *See* Greenwich Mean Time (GMT),
 expressing in local
 Go To dialog box, 245
 Go To Special dialog box
 accessing, 521, 573
 Dependents option, 576
 Formulas option, 530, 573
 Precedents option, 576
 Goal Seek dialog box, 54, 481
 goal seeking
 described, 53
 Internal Rate of Return (IRR), 345
 payment calculations, 53–54
 precision, 55
 single-cell, 53
 solution not found, 54–55
 Solver, using, 55
 gpa.xls (on the CD-ROM), 767
 grade lookup.xls (on the CD-ROM), 767
 grade point average calculations using lookup
 formulas, 225
 Greenwich Mean Time (GMT), expressing in
 local, 175
 Group and Outline ⇄ Group, 503, 504
 Group and Show Detail ⇄ Group, 503
 Grouping dialog box, 503, 505
 GROWTH function, 391, 729

growth rate calculations
 average, 340–341
 average, annual, 298, 299–300
 average, geometric, 346–347
 indexes based on, 347

H

hard coding, avoiding, 40, 351
 Harker, Norman, 314
 HARMEAN function, 729
 HASDATE function, 531
 HasFormula property, 530, 647
 HASLINK function, 532
 Help ⇄ Lotus 1-2-3 Help, 701
 Herber, Hans, 165
 HEX2BIN function, 720
 HEX2DEC function, 720
 HEX2OCT function, 720
 hexadecimal numbers, converting to/from,
 719, 720
 Hidden property, 649
 hiding formulas, 26, 48–49
 histograms, 123–124, 198–200, 451–452. *See*
 also charts
 history of Excel, 4–7
 HLOOKUP function
 arguments, 215
 case sensitivity, 222
 described, 212, 725
 errors returned by, 565
 looking up closest match, 230–231
 looking up exact values, 220
 syntax, 215
 holidays.xls (on the CD-ROM), 766
 Home Page, Excel, 760
 HOUR function, 168, 718
 HYPERLINK function, 725
 Hypertext Markup Language (HTML), using as
 native file format, 8
 HYPGEOMDIST function, 729
 hypocycloid curves, plotting in charts, 468–469
 hypocycloid.xls (on the CD-ROM), 771

I

If Err statements, 642
 IF function
 cell ranges, identifying differences between
 using, 404
 cell ranges, identifying values in using,
 399, 404–405
 circular references, in intentional, 122
 described, 99, 180, 724
 error values, by displaying empty string
 using, 524
 error values, identifying using,
 398, 399, 564

- nesting, 103, 393
- text operations, in, 115, 132, 133
- time stamp calculations, in, 432
- If-Then constructs, 620–621, 633–635
- If-Then-Else constructs, 634, 635
- IMABS function, 720
- IMAGINARY function, 720
- IMARGUMENT function, 720
- IMCONJUGATE function, 720
- IMCOS function, 720
- IMDIV function, 720
- IMEXP function, 720
- IMLN function, 720
- IMLOG2 function, 720
- IMLOG10 function, 720
- import operations. *See also* Lotus 1-2-3
 - database data, 245
 - numbers as text considerations, 115
 - transition formula evaluation, 74–75, 261, 700–701, 702
 - validation checks following, 406
- IMPOWER function, 720
- IMPRODUCT function, 720
- IMREAL function, 720
- IMSIN function, 720
- IMSQRT function, 720
- IMSUB function, 720
- IMSUM function, 720
- INDEX function
 - array elements, accessing using, 383, 405
 - array specification using, 91
 - cell ranges, return single values from
 - multicell using, 73, 405
 - described, 725
 - form, choosing, 108
 - lookup formulas, in, 212, 217–219, 227
 - MATCH function, combining with, 217–219
 - volatile nature of, 111
- indices, 346–347, 370–372
- INDIRECT function
 - cell range names, working with, 88–90
 - described, 725
 - integers, generating consecutive using, 390
 - lookup formulas, in, 231
 - summing operations, in, 402
 - volatile nature of, 111
 - worksheet-level names, working with, 89
- information functions, 110, 723–724
- INFO function, 723
- Insert Calculated Field dialog box, 508
- Insert Calculated Item dialog box, 510
- Insert ⇄ Diagram, 18
- Insert dialog box, 10
- Insert ⇄ Function, 105
- Insert Function dialog box
 - accessing, 105
 - functions, descriptions displayed, 606–607
 - functions, inserting, 104–106, 108, 607
 - functions, listing available, 105, 605
 - functions, searching, 105
 - functions, selecting, 105–106
 - functions, specifying
 - categories for, 607–609
- Insert ⇄ Map, 19
- Insert ⇄ Module, 600
- Insert ⇄ Name ⇄ Apply, 73
- Insert ⇄ Name ⇄ Create, 61, 228
- Insert ⇄ Name ⇄ Define, 58, 444
- Insert ⇄ Name ⇄ Paste, 32
- Insert-A-Date utility (on the CD-ROM), 159
- INT function, 285, 288–289, 726
- integers
 - days of week, returning as, 679
 - generating consecutive, 389–390, 406
 - generating consecutive arranged
 - randomly, 689–690
 - generating random, 433–434
 - generating random nonduplicated, 689–690
 - summing digits of using arrays, 406–408
 - truncating numbers to, 285, 288–289
- INTERCEPT function, 729
- interest
 - accumulation calculations,
 - 297–304, 362, 721
 - amortization calculations, 308–313
 - amortization schedules, 320, 322, 352–358
 - Analysis ToolPak add-in features, 322
 - annual effective, 314, 722
 - annual effective *versus* nominal
 - compounded monthly, 311
 - annual in arrears basis, 319
 - annual payments / 12 basis, 319–320
 - on balances, negative *versus*
 - positive, 344–345
 - compounding frequency not matching
 - payment frequency, 323–324
 - conversions, 313–316
 - cumulative, returning, 721
 - defined, 294
 - discounting calculations, 304–308,
 - 329–330, 339, 341–342
 - Excel functions limited to one level, 324
 - finance *versus* deposit rate, 343–344
 - flat, 318–319
 - hurdle, 330
 - Internal Rate of Return (IRR) analyses, 343
 - loan analysis, variable rate, 369–370
 - loan payments, returning interest associated
 - with specific, 722
 - loans, interest-free, 319

continued

- interest *continued*
 - megaformulas, using for calculating, 542–543
 - monthly effective, 330
 - multiple, calculation from regular cash flows, 324, 342–345
 - Net Present Value (NPV) analyses, 343
 - nominal, 313, 314, 722
 - payment for given period, returning, 722
 - period, 294
 - periodic effective, 314
 - principal *versus* interest calculations, 320–323
 - quote methods, 313–314
 - rental calculations, 305–308
 - return, calculating average, 302, 340–341
 - securities, calculating accrued interest, 721
 - tax effects on interest payments
 - calculations, 320
 - term, 294
 - total, 353, 354, 542–543
 - Interest Conversion Functions Demo.xls (on the CD-ROM), 769
 - Internal Rate of Return (IRR)
 - Analysis ToolPak add-in features, 366
 - annual effective, 364
 - averages, calculating, 340–341
 - cash flow requirements, 339–340, 341
 - cash flows, calculation against irregular, 366–367
 - cash flows, discounted, 363, 364
 - credit card calculations, in, 365
 - cross-checking, 342
 - defined, 341
 - goal seeking, 345
 - growth rates, calculating, 340–341
 - interest rate assumptions, 343
 - loan analysis using, 339
 - multiple, calculations involving, 342–345, 369
 - Net Present Value (NPV), as special case of, 339
 - Risk Rate Equivalent IRR, 345
 - Internet resources
 - comp.apps.spreadsheets FAQ site, 764
 - Excel Home Page, 760
 - Excel Web Source, 764
 - Microsoft Knowledge Base, 759–760
 - newsgroups, 760–763
 - Office 2002 Web site, 760
 - search engines, 762
 - Spreadsheet Page, The, 763–764
 - Stephen Bullen's Excel Page, 764
 - interpolated lookup.xls (on the CD-ROM), 767
 - interpolation. *See also* extrapolation using charts
 - charts, in, 454, 474
 - lookup formulas, in, 231–234
 - Intersect function, 650–651
 - Intersect method, 697
 - INTRATE function, 722
 - INVALIDPART function, 533
 - IPMT function, 320–322, 722
 - IRR. *See* Internal Rate of Return (IRR)
 - IRR function
 - credit card calculations, 365
 - described, 339, 722
 - Discounted Cash Flow (DCF), working with, 363, 364
 - income flow requirements, 339
 - iteration, 339
 - loan cost before tax relief calculations, 356
 - range argument, 339
 - syntax, 339
 - ISBLANK function, 110, 724
 - ISBOLD function, 210, 658
 - IsDate function, 531, 659
 - ISEMPTY function, 659
 - ISERR function, 185–186, 724
 - ISERROR function, 185–186, 398–399, 659, 724
 - ISEVEN function, 724
 - ISITALIC function, 210, 658
 - ISLIKE function, 137, 670–671
 - ISLOGICAL function, 185, 659, 724
 - IsMissing function, 693
 - ISNA function, 185–186, 398, 724
 - ISNONTEXT function, 185, 686, 724
 - IsNull function, 658
 - ISNUMBER function, 724
 - ISNUMERIC function, 659, 697
 - ISODD function, 724
 - ISPMT function, 722
 - ISREF function, 724
 - ISTEXT function, 110, 116, 185, 521, 724
 - iteration. *See also* loop constructs
 - allowing, 429
 - calculation mode considerations, 430–431
 - circular references, using, 428–429, 439
 - errors, 566
 - IRR function, 339
 - maximum permitted, setting, 430
 - RATE function, 302
 - settings, 52, 427, 428–429, 431, 439
 - speed considerations, 430
- ## J
- Jansen, Thomas, 165
 - jogging log.xls (on the CD-ROM), 766
 - Julian dates, 157
 - JWalk Enhanced Data Form add-in, 248
- ## K
- Kapor, Mitch, 4
 - keyboard shortcuts, 11, 14
 - KURT function, 730

L

- Labor Day, calculating, 164
- LARGE function, 100, 204, 402, 524, 730
- last nonempty cell.xls (on the CD-ROM), 773
- LASTINCOLUMN function, 680–681
- LASTINROW function, 681
- LCM function, 726
- leap year bug, 147
- LEFT function, 102, 127, 133, 693, 732
- LEN function (Excel), 126, 391, 406, 732
- Len function (Visual Basic for Applications (VBA)), 620
- Like operator, 533
- line breaks
 - cells, in, 15, 119–120
 - formulas, in, 32
- linear interpolation in lookup formulas, 231–234
- linear trendline.xls (on the CD-ROM), 771
- LINEST function, 391, 730
- link formulas, 44–45, 532
- link reports, 581
- linked picture.xls (on the CD-ROM), 771
- liquid measurement unit conversions, 272, 274
- List separator setting, 100
- lists (worksheet databases), 237. *See also* databases (worksheet lists)
- LN function, 726
- loan data tables.xls (on the CD-ROM), 769
- loans. *See also* Payment (PMT)
 - amortization calculations, 308–313
 - amortization schedules, 320, 322, 352–358
 - annual in arrears basis, 319
 - cash price equivalent, comparing, 319
 - cost, calculating effective, 316–320
 - cost before tax relief calculations, 356
 - Effx_AnnEff function in loan calculations, 319–320
 - fees, bank, 317–318
 - flat rate, 318–319
 - interest associated with specific payments, returning, 722
 - interest-free, 319
 - Internal Rate of Return (IRR), analysis using, 339
 - mortgage calculations, 309–310, 311–313, 317–318, 319–320, 322–323
 - overdrafts, 303
 - principal calculations, 320–323, 721
 - RATE function in cost calculations, 318–319
 - summary data tables, 358–361
 - terminology, 294
 - variable rate analysis, 369–370
- LOG function, 726
- LOG10 function, 726
- logarithms, returning, 726
- LOGEST function, 391, 730
- logical comparison operators, 36, 38, 568
- logical functions, 110, 400–401, 724
- logical functions.xls (on the CD-ROM), 770
- LOGINV function, 730
- LOGNORMDIST function, 730
- lookup address.xls (on the CD-ROM), 767
- lookup formulas
 - array formulas, using, 222, 224, 225–226, 229, 395–396
 - averages, calculating using, 221, 225
 - case-sensitive, 222
 - cells, empty, 219
 - described, 211–212
 - functions relevant to, 110, 212, 725
 - grade point average calculations, 225
 - horizontal lookups, 212, 215, 220
 - linear interpolation in, 231–234
 - missing data errors, 565
 - ranges, returning item position in, 212, 217, 229–230
 - ranges, returning references to, 212
 - ranges, returning values from, 212, 216–219
 - sort order considerations, 213, 216
 - tax calculations using, 110, 214, 216
 - test score calculations, 224
 - two-column lookups, 228–229
 - two-way lookups, 226–228
 - values, estimating missing, 231–234
 - values, looking up closest match, 230–231
 - values, looking up exact, 220
 - values, looking up in columns other than first, 221
 - values, returning specific from listed, 212
 - vertical lookups, 212, 213–214, 220–221
 - wildcard characters in, 214, 215, 217
- LOOKUP function, 212, 216–217, 234, 725
- lookup tables, 213, 215, 223
- lookup to the left.xls (on the CD-ROM), 767
- Lookup Wizard add-in, 227
- loop constructs. *See also* iteration
 - cell ranges, looping through, 643–644
 - circular references, using, 428
 - counters, using, 639–640
 - Do Until loops, 640–641
 - Do While loops, 639–640
 - For Each-Next loops, 643–644
 - For-Next loops, 620–621, 637–639, 646
 - variables in, monitoring values using Debug.Print statements, 612–613
 - variables in, monitoring values using MsgBox statements, 611
 - Visual Basic for Applications (VBA), 611–613, 620–621, 636–641

- Lotus 1-2-3
- array formulas, using to convert database functions, 715
 - Boolean expressions, differences in values returned, 702-703
 - calculation order differences, 701
 - commands, listing, 701
 - database criteria evaluation differences, 703
 - database functions, converting, 714-715
 - date features legacy, 147, 158
 - dates containing hyphens, importing, 703
 - file types, 699
 - files, importing from, 261, 700
 - files supported in Excel, 700
 - formulas, differences from Excel, 700
 - formulas, importing from, 700-701
 - functions, Excel equivalents, 704-713
 - help feature, 701
 - history, 4
 - logical expressions, differences in values returned, 702-703
 - macros, executing in Excel, 704
 - number values as text, 114
 - operator precedence differences, 701
 - operators, Excel accommodation of, 14, 30, 101
 - text data in numerical operations, differences in values returned, 702
 - transition formula evaluation, 74-75, 261, 700-701, 702
 - Version Manager, 23
- LOWER function, 126, 732
- ## M
- Macauley modified duration, returning, 722
- Macintosh computers, 3, 4, 140
- Macro dialog box, 606
- Macro Options dialog box, 607
- macro recorder, 596-598. *See also* Visual Basic for Applications (VBA)
- macros. *See also* Visual Basic for Applications (VBA)
- cell range names, using in, 58
 - names created by, hidden, 64
 - objects, attached to, 14, 16
 - sheets, 8, 10
- Madison, Josh, 278
- maps, 19
- Martin Luther King Jr. Day, calculating, 163
- MATCH function
- arguments, 217
 - cell offset, using to determine, 405
 - described, 725
 - INDEX function, combining with, 217-219
 - missing data errors, 565
 - space characters in arrays, using to locate, 549-550
 - syntax, 217
- math. *See also* algebra; geometry; summing; trigonometry
- absolute values, returning, 173, 407
 - addition, 36, 38, 568, 629
 - binary numbers, converting to/from, 719, 720
 - decimal numbers, converting to/from, 719, 720
 - division, 36, 38, 60, 629, 727
 - division by zero error, 50, 564
 - divisor, returning greatest common, 726
 - equal to comparisons, 34, 36, 119-120, 719
 - even numbers, testing for, 724
 - exponentiation, 36, 38, 568, 629, 726
 - factorials, returning, 720, 726
 - floating point number errors, 571
 - functions related to, 109, 726-728
 - greater than comparisons, 36
 - greater than or equal to comparisons, 36
 - hexadecimal numbers, converting to/from, 719, 720
 - less than comparisons, 36, 37
 - less than or equal to comparisons, 36, 37
 - modulo arithmetic, 629
 - multiple, returning least common, 726
 - multiplication, 33
 - multiplication, database entries, 717
 - multiplication operators, 36, 38, 568, 629
 - negation operator, 38, 568
 - octal numbers, converting to/from, 719, 720
 - odd numbers, testing for, 724
 - percent operator, 36, 38, 568
 - pi, returning value of, 727
 - rounding numbers, 176-177, 285-290, 306, 726, 727
 - square root, 35, 102, 338, 631, 727
 - subtraction operator, 36, 38, 569, 629
 - threshold values, testing if greater than, 720
- matrices, 284, 726, 727
- MAX function, 405, 433, 453, 524, 730
- MAXA function, 730
- MAXALLSHEETS function, 682-683
- maximums
- cells, characters in, 113-114
 - cells, number of, 9
 - columns, number of, 9
 - data type ranges, 624
 - database records, number of, 22
 - formula size, 32, 541
 - iterations, 430
 - name length, 60
- MDETERM function, 726
- MDURATION function, 722
- measurement unit conversions
- Analysis ToolPak add-in features, 269
 - distance, 272, 273
 - energy, 272, 276
 - force, 272, 276

- liquid measurements, 272, 274
 - metric, to/from, 270–271
 - surface, 272, 275
 - tables (on the CD-ROM), 272
 - temperature, 277
 - time, 272, 277
 - volume calculations, 272, 275
 - weight, 272, 273
 - MEDIAN function, 730
 - megaformulas
 - advantages/disadvantages, 557
 - arrays in, 548–552, 554–555
 - cell references, substituting formula text for, 544
 - checksum digits, working with, 552
 - creating, 542–543, 550
 - credit card validity checks using, 552–556
 - described, 541
 - interest calculations using, 542–543
 - intermediate formulas, avoiding using, 541
 - length, maximum, 541
 - names, removing middle using, 544–547
 - parentheses in, 551
 - spaces, returning positions of using, 548–552
 - speed considerations, 547–548
 - text, copying from formulas, 544
 - text manipulations using, 544–547
 - Memorial Day, calculating, 164
 - menus, 11. *See also* toolbars
 - messages, custom data validation, 534
 - metric units, conversion to/from, 270–271
 - Microsoft resources
 - Excel Home Page, 760
 - Knowledge Base, 759–760
 - newsgroups, 761–762
 - Office 2002 Web site, 760
 - support options, listing, 759
 - MID function (Excel), 127, 153, 407, 549, 732
 - Mid function (Visual Basic for Applications (VBA)), 621
 - military time, converting from, 174
 - MIN function, 405, 453, 730
 - MINA function, 730
 - MINUTE function, 168, 718
 - MINVERSE function, 391, 726
 - MIRR function, 324, 342–345, 722
 - MMULT function, 391, 727
 - MOD function, 173, 409, 456, 525, 727
 - MODE function, 191, 730
 - modules. *See also* Visual Basic for Applications (VBA)
 - adding, 592, 600
 - class, 593
 - exporting, 593
 - importing, 593
 - introduced, 588
 - naming/renaming, 592
 - removing, 593
 - sheets, 8
 - modulo arithmetic, 629
 - MONTH function, 150, 161, 168, 522, 718
 - MONTHNAMES function, 687–688
 - monthnames.xls (on the CD-ROM), 773
 - MONTHWEEK function, 678–679
 - MROUND function, 285, 287, 727
 - MsgBox statements, 610–612
 - multi-cell array formulas.xls (on the CD-ROM), 770
 - MULTINOMIAL function, 727
 - MultiPlan, 4
 - multiple charts.xls (on the CD-ROM), 771
 - multiple criteria counting.xls (on the CD-ROM), 767
 - multiple lookup tables.xls (on the CD-ROM), 767
 - multiplication
 - database entries, 717
 - introduced, 33
 - operators, 36, 38, 568, 629
 - multisheet functions.xls (on the CD-ROM), 773
 - Myfuncs.xls, 616
 - MYSUM function, 695–698
 - mysum function.xls (on the CD-ROM), 773
- ## N
- N function, 724
 - NA function, 724
 - #N/A missing data errors, 50, 185–186, 398, 565
 - Name box
 - accessing, 58, 61
 - cell range selection using, 60–61
 - cell selection using, 60–61
 - name changing using, 80
 - name creation using, 59–61, 67
 - name display in, 59, 64, 66, 67, 83
 - Name Lister utility (on the CD-ROM), 64, 581
 - Name property, 648
 - #NAME? undefined name/range errors, 50, 565
 - names. *See also* cell names; cell ranges, names
 - add-ins, created by, 64
 - application names, returning, 657
 - array constants, naming, 381–383
 - changing, 8, 76, 80
 - characters, prohibited, 60
 - charts, naming, 80
 - columns, naming, 63
 - constants, naming, 81–83, 381–383
 - copying/pasting considerations, 32, 64, 78–79
 - deleting, 75
 - described, 57–58
 - formulas, dynamic named, 91–93
 - formulas, named, 80
 - formulas, using arrays in named, 90–91, 381–383
 - formulas, using cell range references in named, 84–85

continued

- names *continued*
 - formulas, using cell references in named, 84–87
 - formulas, using functions in named, 83–84
 - hidden, 64
 - internal to Excel, 64
 - internal to Excel, modifying, 64
 - internal to Excel, overriding, 60
 - length, maximum, 60
 - listing, 64, 80, 581
 - macros, created by, 64
 - multisheet, 65–66
 - objects, naming, 80
 - personal, changing case, 126–127
 - personal, extracting portions of, 134–135
 - personal, removing middle, 544–547
 - pivot tables, naming, 500
 - precedence, 67
 - redefining, 76
 - rows, naming, 63
 - rules, 60
 - scope, 66–67
 - spaces in, 60, 67
 - unapplying, 74–75
 - workbook-level, 67–68, 69
 - worksheet names, changing, 8
 - worksheet-level, 67–68, 69
- natural language formulas, 71–72
- negation operator, 38, 568
- NEGBINOMDIST function, 730
- nested subtotals.xls (on the CD-ROM), 768
- Net Present Value (NPV). *See also* Present Value (PV)
 - accumulation calculations, 324, 337–338, 346–347
 - Analysis ToolPak add-in features, 366
 - cash flows, calculating accumulation at different rates, 324, 346–347
 - cash flows, calculating for irregular, 366, 367–368
 - cash flows, calculating sums of irregular, 325, 366, 367–368
 - cash flows, calculating sums of regular, 324, 329–338
 - cash flows, discounted, 363, 364
 - cash flow's effect on, 330
 - cash flows beginning at end of first period, calculating for, 332
 - cash flows in advance, calculating for, 333–334
 - cash flows initially positive, calculating for, 332–333
 - cash flows series, calculating for, 331
 - cash flows with rate time period mismatch, calculating for, 335–336
 - cash flows with terminal values, calculating for, 333–335
 - credit card calculations, in, 365
 - cross-checking calculations, 331, 339, 342
 - defined, 330
 - discount rate, calculations involving, 329–330, 339, 341–342, 363–364
 - Future Value (FV), calculating from, 337
 - interest rate assumptions, 343
 - Internal Rate of Return (IRR) as special case of, 339
 - negative, 330
 - positive, 330
 - rate, affect on, 330
 - rental situations, 333, 336, 338
 - zero, returning, 339–342
- net profit (circular).xls (on the CD-ROM), 770
- net profit (not circular).xls (on the CD-ROM), 770
- NETWORKDAYS function, 150, 154–155, 718
- newsgroups, Excel, 760–763
- NEXTDAY function, 678
- NEXTMONDAY function, 677–678
- no middle name.xls (on the CD-ROM), 772
- NOMINAL function, 315–316, 722
- Nomx_AnnEff function, 315
- Nomx_Effx function, 315, 318, 322–323
- Nomx_Effy function, 315, 353
- Nomx_Nomy function, 315
- nonlinear trendline.xls (on the CD-ROM), 771
- NORMDIST function, 730
- NORMINV function, 730
- NORMSDIST function, 730
- NORMSINV function, 730
- NOT function, 724
- #NOT# Lotus logical operator, 703
- NOW function
 - chart updates using, 466
 - circular references, in, 122
 - clocks, displaying using, 484
 - date, returning using, 122, 150, 168
 - described, 718
 - time, returning using, 122, 150, 168, 432
 - volatile nature of, 111
- NPER function
 - accumulation calculations, 298, 303
 - amortization calculations, 310–311
 - arguments, required, 294
 - credit card calculations, 324, 365
 - described, 722
 - money flow, 295
 - PMT function, relationship with, 295
 - RATE function, relationship with, 295
 - time value of money concept used by, 295
- NPV. *See* Net Present Value (NPV)
- NPV function
 - cash flows beginning at end of first period calculations, 332
 - cash flows in advance calculations, 333–334
 - cash flows initially positive calculations, 332–333
 - cash flows series calculations, 331

- cash flows with rate time period mismatch calculations, 335–336
- cash flows with terminal value calculations, 333–335
- credit card calculations, in, 365
- described, 722
- Discounted Cash Flow (DCF), working with, 363, 364
- PMT function, nesting in, 338
- rental situations, calculations in, 333, 336
- #NULL! range intersection errors, 50, 565–566
- Null values, testing for, 658
- #NUM! errors, 50, 566
- number formats
 - Accounting format, 736
 - aligning numbers using custom formats, 748, 750
 - automatic, 16, 733–734
 - automatic, avoiding, 734
 - cell format, returning, 648, 659
 - cell references, correspondence in, 31
 - cells, displaying number format of, 756
 - cells, filling with repeating characters using custom formats, 756–757
 - cells, padding with dashes using custom formats, 756–757
 - cells containing specific, counting, 210
 - charts, formatting numbers in, 452, 737
 - codes, custom format, 738–739, 740–741
 - colors in custom formats, 518, 739, 751–752
 - commas, 733, 735
 - conditional formatting, 518, 750–751
 - creating custom, 738–742
 - currency, 15, 16, 733, 735, 736
 - date format, applying, 140, 145–146, 734, 736
 - date format, automatic, 146, 734
 - date format, custom, 146, 741, 752
 - date format, default, 141
 - date format compatibility, 139, 152, 254
 - date formats recognized, 141–142
 - decimal places, increasing/decreasing, 735, 742
 - default, 733
 - dots, displaying leading using custom formats, 757
 - exponential format, 734, 737
 - Format Cells dialog box, using, 736–737
 - Formatting toolbar, using, 734–735
 - fractions, 737, 748–749
 - General format, 733, 736
 - hiding values using, 755
 - locale-dependent, 737
 - negative sign display in custom formats, 750
 - percentages, 15, 734, 735, 736, 747
 - predefined, 16
 - preformatting cells to avoid
 - automatic, 734, 740
 - scaling values in custom formats, 742–746
 - scientific notation, 734, 737
 - shortcut keys for applying, 735
 - Social Security Numbers, 737
 - strings, custom format, 738–739
 - symbols in custom formats, 754–755
 - testing custom formats, 749
 - text, displaying in quotes in custom formats, 749
 - text, displaying N/A for in custom formats, 749
 - text, displaying with numbers in custom formats, 753
 - text, repeating in custom formats, 749
 - text, treating numbers as, 737
 - TEXT function, using for, 754
 - time, 145–146, 734, 736
 - time, custom formatting, 146, 741, 752
 - value display, effect on, 733
 - worksheets, storage with, 738
 - zeros, adding to custom formats, 746
 - zeros, displaying leading in custom formats, 747
 - zeros, displaying with dashes in custom formats, 753–754
 - zeros, hiding in custom formats, 746–747
 - Zip Codes, 737, 739, 751
- NUMBERFORMAT function, 210, 659
- NumberFormat property, 648
- numbers. *See also* math
 - binary, converting to/from, 719, 720
 - complex, functions related to, 719, 720
 - conditional formatting, avoiding using custom number formats, 518
 - data entry restriction to number ranges, 536
 - data entry restriction to whole, 536
 - decimal numbers, converting to/from, 719, 720
 - even, testing for, 724
 - fraction/decimal conversions in financial formulas, 285
 - hexadecimal, converting to/from, 719, 720
 - integers, generating consecutive, 389–390, 406
 - integers, generating consecutive arranged randomly, 689–690
 - integers, generating nonduplicated, 689–690
 - octal, converting to/from, 719, 720
 - odd, testing for, 724
 - ordinals, expressing as, 131–132, 162
 - padding, 124–125
 - random, generating, 100, 433–434, 662–664, 689–690, 727
 - Roman numerals, 166, 633
 - rounding, 176–177, 285–290, 306, 726, 727

continued

- numbers *continued*
 - sign of, returning, 727
 - spelling out, 137, 674
 - text, as, 114–115
 - text, displaying as, 121–122
 - text string conversion to numeric values, 407
 - thousands separator, 298, 733, 735
 - truncating, 285, 288–289
- O**
- object model, 7
- objects
 - container objects, 647
 - hierarchy, 656
 - macros, attached, 14, 16
 - naming, 80
 - parent objects, returning, 647, 655–656
 - selecting, 14, 16
 - Visual Basic for Applications (VBA), 593
- OCT2BIN function, 720
- OCT2DEC function, 720
- OCT2HEX function, 720
- octal numbers, converting to/from, 719, 720
- ODD function, 285, 289–290, 727
- ODDFPRICE function, 722
- ODDFYIELD function, 722
- ODDLPRICE function, 722
- ODDLYIELD function, 722
- Office 2002 Web site, 760
- Office XP, Excel 2002 relation to, 6
- OFFSET function
 - array formulas, in, 411
 - charts, using in, 457, 458–459, 464–465
 - described, 725
 - formulas, in dynamic named, 92
 - lookup formulas, in, 212
 - values in specified cells, returning using, 411
 - volatile nature of, 111
- Offset property, 646
- OLAP pivot table data sources. *See* Online Analytical Processing (OLAP) pivot table data sources
- On Error statements, 641–643, 651
- Online Analytical Processing (OLAP) pivot table data sources, 506
- operating environment, returning information about, 723
- operators
 - addition, 36, 38, 568, 629
 - #AND# Lotus logical operator, 703
 - And Visual Basic for Applications (VBA) operator, 629
 - arithmetic, 36
 - assignment, 622
 - concatenation, 36, 38, 119–120, 568, 629
 - division, 36, 38, 60, 629
 - Eqv Visual Basic for Applications (VBA) operator, 629
 - exponentiation, 36, 38, 568, 629
 - formula elements, as, 29
 - Imp Visual Basic for Applications (VBA) operator, 629
 - integer division Visual Basic for Applications (VBA) operator, 60
 - intersection operators, 36, 70, 72
 - Like, 533
 - logical comparison, 36, 38, 568
 - Mod Visual Basic for Applications (VBA) operator, 629
 - multiplication, 36, 38, 568, 629
 - negation, 38, 568
 - #NOT# Lotus logical operator, 703
 - Not Visual Basic for Applications (VBA) operator, 629
 - #OR# Lotus logical operator, 703
 - Or Visual Basic for Applications (VBA) operator, 629
 - percent, 36, 38, 568
 - precedence, 38–40
 - precedence, debugging, 568–569
 - precedence, Lotus differences from Excel, 701
 - Quattro Pro, 101
 - range operators, 36, 72
 - range reference, 36, 101
 - reference operators, 36
 - search, 129
 - subtraction, 36, 38, 569, 629
 - union, 36
 - Xor Visual Basic for Applications (VBA) operator, 629
- Option Base keyword, 630
- Option Explicit statements, 625, 626
- optional keyword, 693
- Options dialog box
 - Calculation tab Accept labels in formulas setting, 71, 255
 - Calculation tab calculation mode setting, 361
 - Calculation tab date system setting, 140, 173
 - Calculation tab Iteration setting, 52, 427, 429, 430
 - Calculation tab Maximum change setting, 430
 - Calculation tab Precision as displayed setting, 570
 - Chart tab, 454, 457
 - Color tab, 516
 - Edit tab, 13, 34
 - Error Checking tab, 578
 - General tab, 597

Transition tab, 74, 261, 700
 View tab, 14, 573
 Zero values option, 184
 OR/AND criteria, 189–191, 208–210
 OR function, 37, 103, 400–401, 404, 724
 #OR# Lotus logical operator, 703
 ordinal dates.xls (on the CD-ROM), 766
 outlines, 23, 266–267
 overdrafts, 303

P

page breaks between subtotal groups, 265
 Page Setup dialog box, 524
 Panko, Ray, 560
 ParamArray keyword, 694
 Parent property, 647
 parentheses
 entering, 39, 561
 function argument delimiters, 99, 100
 megaformulas, in, 551
 mismatched, debugging, 561
 nesting, 39–40
 operator precedence control using, 30, 38–40
 SERIES formula argument delimiters, 442
 passwords
 databases, external, 501
 project passwords, 590
 security of, 26, 618
 worksheets, protecting using, 25
 Paste Function dialog box, 105, 606
 Paste Name dialog box, 32, 83
 paste operations. *See* copy/cut/paste operations
 Paste Special feature, 46, 48, 245
 Payment (PMT)
 accumulation calculations, 299, 301–302
 aggregate calculations, 322–323
 amortization calculations, 309, 311–313
 annual / 12 basis, 319–320
 credit card calculations, 324, 365
 deferred, 325
 defined, 294
 discounting calculations, 307, 309–313
 Excel functions limited to one level of, 324
 frequency mismatch with rate, 323–324
 goal seeking calculations, 53–54
 mortgage calculations, 296, 310, 311–313
 negative values, 296
 past-due, 204–205
 positive values, 296
 principal *versus* interest calculations, 320–323
 rent, 305, 306, 338
 schedules calculations, 353
 signing, 296
 tax effects on interest calculations, 320
 varying, 324–326
 varying, replacing with time-weighted average, 338
 Pearson, Chip, 764
 PEARSON function, 730
 peppercorn rent, 306
 percentages
 charts, displaying as data series in, 448–449
 frequency distributions consisting of, 196
 number format, 15, 734, 735, 736, 747
 operator, 36, 38, 568
 PERCENTILE function, 730
 PERCENTRANK function, 730
 perimeter calculations, geometric, 280–281
 PERMUT function, 730
 Perpetual Calendar utility (on the CD-ROM), 159
 personal names, text manipulations in, 126–127, 134–135, 544–547
 pi, returning value of, 727
 PI function, 727
 pictures, linked, 19, 447–448
 Pivot Table toolbar, 508
 pivot tables
 AutoFormatting, 500
 cache, 501
 cells, empty, 501
 column headers, 492
 column orientation, 492
 column titles, printing on each page, 501
 copying, 504
 counting operations in, 181, 195, 494
 creating, 495–503
 cross-tabulations, 489, 494
 data appropriate for, 493–494
 data area, 492
 Data Form feature, using, 497
 data sources, databases as, 496
 data sources, external, 496, 501
 data sources, multiple consolidation ranges as, 496
 data sources, Online Analytical Processing (OLAP), 506
 data sources, other pivot tables as, 496
 data sources, specifying, 496–497
 data sources, specifying location, 495
 database queries, background, 501
 database summaries using, 491
 databases, external, 501
 dates, summarizing by, 503–505
 described, 24, 489
 drag and drop, 499, 502, 510
 drilling to details, 501
 error value display, 501
 fields, calculated, 506–511
 fields, column, 492, 501
 fields, customizing, 499
 fields, dragging, 499, 502

continued

- pivot tables *continued*
 - fields, names in data source, 496
 - fields, ordering on page, 501
 - fields, page, 493
 - fields, row, 493
 - fields, summarizing non-numeric, 494
 - formatting, preserving, 500
 - formulas, using for calculations in, 506, 508–509
 - formulas, using in place of, 489
 - frequency distributions, 195, 489
 - grand totals, 492, 500
 - groups, 492, 503
 - groups, creating automatically, 503
 - groups, dates, 503–505
 - groups, post hoc, 489
 - interactivity, 489
 - item labels, repeating on each page, 501
 - items, 492
 - items, calculated, 506–511
 - items, dragging, 510
 - layout, 498–499, 501, 502
 - layout, saving data with, 501
 - location, 497
 - memory, optimizing, 501
 - merge labels, 500
 - naming, 500
 - numeric values in, 499
 - page setup, 493, 500
 - passwords for external databases, saving, 501
 - printing, 500, 501
 - range, specifying, 496–497
 - refreshing, 489, 493, 501
 - row headers, 492
 - row orientation, 493
 - saving data with layout, 501
 - source data, 493
 - subtotals, 493
 - subtotals of hidden page items, 500
 - summarizing data using, 414
 - summing operations in, 181, 492, 500, 501
 - totals, marking with asterisks, 501
 - updates, automatic, 489
 - uses of, 489
 - version differences, 495
 - versions of Excel, differences between, 495, 498, 499, 504
 - worksheets, saving in, 497
- PivotChart Report, 495
- PivotChart Wizard, 495–498
- PivotTable Field List toolbar, 499, 502
- PivotTable ⇄ Formulas ⇄ Calculated Field, 508
- PivotTable ⇄ Formulas ⇄ Calculated Item, 510
- PivotTable Options dialog box, 500
- plot every nth data point.xls (on the CD-ROM), 772
- plot last n data points.xls (on the CD-ROM), 772
- PMT. *See* Payment (PMT)
- PMT function
 - amortization calculations, 309, 312–313
 - amortization schedules, payment calculation in, 353, 356–357
 - arguments, required, 293
 - credit card calculations, in, 365
 - described, 722
 - discounting calculations, 307
 - money flow, 295
 - NPER function, relationship with, 295
 - NPV function, nesting in, 338
 - payments, calculating aggregate of, 322–323
 - principal *versus* interest payment calculations, cross-checking using, 322
 - RATE function, relationship with, 295
 - time value of money concept used by, 295
- POISSON function, 730
- pop-up boxes, monitoring variables using, 610–612
- position of last space.xls (on the CD-ROM), 772
- POWER function, 727
- Power Utility Pak (on the CD-ROM)
 - Acrobat Reader, 765, 776
 - auditing features, 575, 581–582
 - charts tools, 446
 - Compare Sheets or Ranges feature, 581
 - Date Report, 159, 581
 - described, 21, 765
 - discount coupon, 582
 - formula listing feature, 575
 - Formula Report feature, 581
 - Insert-A-Date, 159
 - installing/uninstalling, 774
 - link report feature, 581
 - Name Lister, 64, 581
 - Perpetual Calendar, 159
 - printing tool, 575
 - registering, 774
 - Reminder Alarm, 159
 - Sound-Proof 2000, 765, 775–776
 - summary reports tool, 581
 - Time Tracker, 159
 - VBA Project Summary Report, 581
 - Workbook Link Report, 581
 - Workbook Summary Report, 581
 - Worksheet Map, 581
- PPMT function, 320–322, 722
- precedents, cell, 575–576
- precision as displayed setting, 570
- Present Value (PV). *See also* Net Present Value (NPV)
 - accumulation at different rates, calculating, 324, 346–347
 - amortization calculations, 309–310

calculating, 304–305, 324–325
 defined, 294
 discounting calculations, 304–306, 307
 Future Value (FV), calculating from, 324
 Internal Rate of Return (IRR) checks using
 sum of, 341–342
 Net Present Value (NPV) checks using sum
 of, 341–342
 Presidents' Day, calculating, 164
 PRICE function, 722
 PRICEDISC function, 722
 PRICEMAT function, 722
 Print keyword, 615
 Print_Area internal name, 64
 printing
 characters, removing nonprinting, 125–126
 charts, 10, 443
 error values options, 524, 563
 pivot tables, 500, 501
 Power Utility Pak (on the CD-ROM) tool,
 575
 Print_Titles internal name, 64
 PROB function, 730
 PRODUCT function, 727
 projects. *See also* Visual Basic for Applications
 (VBA)
 code window, 593
 described, 590
 expanding/contracting, 590
 modules, adding to, 600
 naming/renaming, 591, 605
 passwords, 590
 saving, 598
 workbooks as, 590
 PROPER function, 98, 126, 732
 Protect Sheet dialog box, 25, 26
 Protect Workbook dialog box, 26
 protection features, 25–26, 49
 Public keyword, 630
 PV. *See* Present Value (PV)
 PV function
 accumulation calculations, 298–299,
 303–304
 amortization calculations, 309–310, 324
 arguments, required, 293
 credit card calculations, 365
 described, 723
 discounting calculations, 304–306, 307–308
 money flow, 295
 payment levels, chaining to deal with
 differing, 325–326
 time value of money concept used, 295

Q

QUARTILE function, 453, 730
 quartile plots, 453–455
 Quattro Pro, 5, 101
 Query utility, 496

QUOTIENT function, 727

R

R1C1 notation, 43–44
 RADIANS function, 103, 279, 727
 RAND function, 100, 111, 663, 727
 RANDBETWEEN function, 727
 random integers function.xls (on the CD-ROM),
 773
 random number generation
 0 and 1, between, 100
 cell ranges, in, 433–434, 663, 689–690
 cells, selecting random using, 663–664
 circular reference formulas, using, 433–434
 integers, 433–434
 integers, random nonduplicated, 689–690
 range, in specified, 727
 recalculation, nonchanging during, 663
 RANDOMINTEGERS function, 689–690
 Range objects
 cell ranges, returning for, 646, 651–652
 parent, returning, 655–656
 variables, Range object, 649, 654
 Range property, 644–645
 range randomize function.xls (on the CD-ROM),
 773
 range reference operator, 36, 101
 RANGERANDOMIZE function, 691–692
 RANK function, 412–413, 730
 ranking data, 412–413
 rate, interest. *See* interest
 rate, return. *See* return rate
 RATE function
 accumulation calculations, 298, 299–300,
 302
 amortization calculations, 311
 arguments, required, 294
 credit card calculations, 365
 cross-checks using, 305
 described, 723
 discounting calculations, 306, 307–308
 iteration, 302
 loan effective cost calculations, 318–319
 money flow, 295
 NPER function, relationship with, 295
 PMT function, relationship with, 295
 time value of money concept used, 295
 real estate database.xls (on the CD-ROM), 768
 RECEIVED function, 723
 Record Macro dialog box, 596
 recursive equations.xls (on the CD-ROM), 770
 #REF! invalid cell reference errors, 50, 565
 reference functions, 110, 725
 references, circular. *See* circular references
 References dialog box, 604–605
 References node, 591
 Reminder Alarm utility (on the CD-ROM), 159

- REMOVESPACES function, 619–621, 631
- rental situations
- annualized, 308, 336
 - cash flow calculations, 333, 336
 - discounting calculations, 305–308
 - Net Present Value (NPV) calculations, 333, 336, 338
 - payments, replacing varying with time-weighted average, 338
 - peppercorn rent, 306
- REPLACE function (Excel), 128, 732
- Replace function (Visual Basic for Applications (VBA)), 631–632
- reports
- date reports, 159, 581
 - formulas, listing all in worksheet, 581
 - PivotChart Report, 495
 - Visual Basic for Applications (VBA) procedures, 581
 - workbook links, 581
 - workbook summary, 581
- REPT function, 123–124, 732
- return rate. *See also* Internal Rate of Return (IRR)
- annual effective, 364, 367–368
 - average, calculating, 302, 340–341
 - average, geometric, 340
 - Risk Free Rate of Return, 345
 - Risk Rate Equivalent Internal Rate of Return, 345
- REVERSETEXT function, 137, 611, 614, 668, 685–686
- REVERSETEXT2 function, 668
- RIGHT function, 127, 153, 732
- ROMAN function, 166, 633, 727
- Roman numerals, 166, 633
- ROUND function, 176–177, 285, 286–287, 727
- round numbers
- creating, 176–177, 285–290, 306, 726, 727
 - summing, 408
- ROUNDDOWN function, 285, 286, 727
- ROUNDUP function, 166, 285, 286, 727
- ROW function, 389–390, 406, 525, 725
- RowOfLargest function, 639
- rows. *See also* worksheets
- cell ranges, returning from, 649
 - cell ranges, returning number in, 648
 - cell ranges, testing for hidden in, 649, 654–655
 - cells, returning last nonempty in, 680–681
 - charts, plotting based on active, 460–461
 - charts, plotting data in hidden, 455
 - database rows, as records, 238
 - database rows, freezing first, 239
 - database rows, labels, 239
 - deleting, 75, 77–78
 - empty, 239
 - finding row of value's nth occurrence, 405
 - formatting, conditional, 525–526, 529
 - function arguments, as, 101
 - height, changing, 9
 - hiding, 243, 244, 355
 - inserting, cell name behavior during, 77
 - labels, using in formulas, 71–72
 - naming, 63
 - number, returning, 456
 - number of, maximum, 9, 238
 - pivot table row headers, 492
 - returning last value in, 412
 - shading using conditional formatting, 525–526
 - testing for hidden, 649, 654–655
- ROWS function, 111, 183, 725
- Rows property, 648
- RSQ function, 474, 731
- RTD function, 725
- Run ⇄ Run Sub/UserForm, 614
- ## S
- Sachs, Jonathon, 4
- sales by date.xls (on the CD-ROM), 772
- sales commission calculations, 664–667
- scenario management, 23
- schedules
- accumulation, 361–363
 - amortization, 320, 322, 352–358
 - balance calculations, 354
 - cash flow, 324–325, 351
 - cash flow, discounted, 363–364
 - dynamic, 351–352
 - hard coding in, 351, 354
 - holding periods calculations, 353
 - indices, creating from schedules of changing values, 370–372
 - interest rates, calculating periodic, 353
 - interest rates, calculating total, 353, 354
 - interest rates, variable, 356–358
 - intermediate calculations section, 352, 353
 - labels, descriptive, 354
 - payment calculations, 353, 356–357
 - principal calculations, 354
 - rows, hiding unused, 355
 - self-checking, 355
 - summary output section, 352, 353–354
 - time periods, cross-comparison between, 370
 - time periods, incrementing, 354
 - time periods, variable, 352
 - user input section, 352
 - worksheets, storage in, 352
- SCRAMBLE function, 137, 669
- search and replace operations, 129–130, 516
- SEARCH function, 128–129, 732

- searching. *See also* lookup formulas
 - cell ranges, finding nth occurrence of values in, 405
 - cell ranges, finding values in, 403–404
 - conditional formatting, for cells containing, 520
 - dates, 143
 - formulas, identifying cells containing, 530–531, 573
 - functions, for, 105
 - text, for, 128–130
 - wildcard characters, using, 129, 137
- SECOND function, 168, 718
- securities, functions related to, 721, 722, 723
- security. *See also* passwords
 - formulas, hiding, 26, 48–49
 - number padding, 124–125
 - passwords, 26, 618
 - Visual Basic for Applications (VBA) code, 618
- Select Case constructs, 635–636, 666
- SERIES formulas. *See also* charts
 - activating, 445
 - arguments, 442
 - category_labels argument, 442
 - charts, relation to, 441–442
 - described, 441–442
 - formulas, using named in, 444, 460–461
 - name argument, 442
 - names, referencing, 77, 200, 442, 444, 460–461
 - order argument, 442
 - plotting order, 442
 - range, unlinking from, 444
 - range references, 442
 - syntax, 442
 - values argument, 442
 - workbooks, referencing, 442
 - worksheet functions, using in, 442
 - worksheets, referencing, 442
- SERIES function, 464
- SERIESSUM function, 727
- Set keyword, 649–650
- shapes, 18
- SHEETNAME function, 655–656
- SHEETOFFSET function, 683–685
- Sheet_Title internal name, 64
- shortcut keys, 11, 14
- shortcut menus, 11
- shortcuts, document, 725
- SIGN function, 727
- simple functions.xls (on the CD-ROM), 773
- simple loan amortization schedule.xls (on the CD-ROM), 769
- SIMPLESUM function, 694–695
- simultaneous equations.xls (on the CD-ROM), 768, 770
- SIN function
 - angles, returning sine of, 103
 - chart operations, in, 462, 483–484
 - described, 727
- single-cell array formulas.xls (on the CD-ROM), 770
- SINH function, 727
- SKEW function, 731
- SLN function, 348, 349, 723
- SLOPE function, 731
- SMALL function, 204, 392, 405, 731
- Smart Icon feature, 50
- Smart Tags, 12, 47, 578–579
- Social Security Numbers number format, 737
- solve right triangle.xls (on the CD-ROM), 768
- Solver add-in
 - goal seeking, using in, 55
 - introduced, 24
 - names created by, hidden, 64
- SORTED function, 419–420
- sorted function.xls (on the CD-ROM), 770
- sorting
 - array formulas, using, 416
 - databases, in, 22
 - lookup formulas, considerations in, 213, 216
 - ranges dynamically, 416
- Sound-Proof 2000 add-in (on the CD-ROM), 765, 775–776
- Southern Cross Software, 582
- spaces
 - cells containing, empty, 563
 - characters, determining if spaces, 621
 - formulas, in, 32
 - names, in, 60, 67
 - positions of, returning, 548–552
 - removing, 125–126, 619–621, 631
- special characters, 119
- speech feature, 775
- speed considerations
 - arrays, 385
 - formulas, intermediate, 548
 - functions, custom, 599
 - iteration, 430
 - summary data tables, 361
 - Visual Basic for Applications (VBA), 548, 599, 623, 625
- SPELLDOLLARS function, 137, 674
- spelldollars function.xls (on the CD-ROM), 773
- Spreadsheet Detective, 582
- Spreadsheet Page, The, 763–764
- Spreadsheet Research (SSR) Web site, 560
- Sqr function, 631
- SQRT function, 35, 727
- SQRTPI function, 727
- Square function, 645
- square root, 35, 102, 338, 631, 727

- squares
 - returning, 645
 - summing, 180–181, 643–644
 - summing differences of, 181
- SSR Web site. *See* Spreadsheet Research (SSR) Web site
- Standard toolbar
 - AutoSum button, 107
 - Copy button, 46
 - Insert Function button, 105
 - Paste button, 46
- STANDARDIZE function, 731
- STATFUNCTION function, 660–662
- statfunction function.xls (on the CD-ROM), 773
- STATICRAND function, 663, 664
- statistics. *See also* counting
 - Analysis ToolPak add-in features, 109
 - covariance, 109
 - frequency distributions, 195–201
 - functions related to, 109, 728–731
 - sums of differences of squares, 181
 - sums of squares, 180–181
 - variance, 718
- STDEV function, 731
- STDEVA function, 731
- STDEVPA function, 731
- STDEVPA function, 731
- Stephen Bullen's Excel Page, 764
- STEYX function, 731
- Stop Recording toolbar, 597
- strings, 113. *See also* text operations
- stylistic formatting, 16–17
- SUBSTITUTE function, 128, 130, 136, 621, 732
- Subtotal dialog box, 264–265
- SUBTOTAL function, 180, 242–244, 264–267, 727
- subtotals, 107, 180, 242–243, 244, 264–267
- subtraction, 33–34, 36, 38, 569, 629
- sum every nth.xls (on the CD-ROM), 770
- SUM function. *See also* summing
 - arguments, 201–212, 694, 695
 - described, 727
- SUMIF function, 180, 204, 206–208, 399, 727. *See also* summing
- summary information, schedule, 352, 353–354
- summary reports using data tables. *See* data tables, summary
- summary reports using pivot tables. *See* pivot tables
- summary reports using Power Utility Pak, 581
- summing
 - AND/OR criteria, using, 208–210
 - array constants, summing operations involving, 378–379
 - arrays, summing operations returning, 180
 - AutoCalculate, using, 182
 - AutoSum, 107
 - cell ranges, all values in, 201–202
 - cell ranges, every nth value in, 409–410
 - cell ranges, largest values in, 204, 402
 - cell ranges, smallest values in, 204, 392
 - cell ranges, squares of values in, 643–644
 - cell ranges, visible cells in, 210, 676–677
 - cell ranges based on different ranges, 206–207
 - cell ranges containing error values, 398
 - checksum digits, 552
 - columns meeting specified criteria, 180
 - conditional sums, 180, 204–210, 399–400, 401
 - cumulative sums, 202–203
 - databases, in, 242–243, 259–260, 264–267, 718
 - date comparisons, based on, 207
 - error values, cell ranges containing, 398
 - formatting, based on cell, 210
 - functions related to, 180–181, 210
 - INDIRECT function, involving, 402
 - integers, digits of, 406–408
 - introduced, 33, 34
 - negative values only, 206
 - Net Present Value (NPV) of irregular flows, 325, 366, 367–368
 - Net Present Value (NPV) of regular flows, 324, 329–338
 - operator, 36
 - pivot tables, in, 181, 492, 500, 501
 - positive values only, 399
 - power series, 727
 - Present Value (PV) sums in cross-checking, 341–342
 - products, 180, 209
 - round values, 408
 - running totals, 202–203
 - squares, differences of, 181
 - squares, of, 180–181
 - subtotals, 107, 180, 242–243, 244, 264–267
 - text comparisons, based on, 207
 - time periods, 170–172, 175
 - Visual Basic for Applications (VBA) functions, using, 210, 643–644, 676–677, 694–698
- SumOfSquares function, 644
- SUMPRODUCT function, 180, 209
- SUMSQ function, 180, 728
- SUMVISIBLE function, 210, 676–677
- SUMX2MY2 function, 181, 728
- SUMX2PY2 function, 181, 728
- SUMXMY2 function, 181, 728
- surface calculations
 - geometry, 282
 - measurement unit conversions, 272, 275
- surface chart.xls (on the CD-ROM), 772
- SYD function, 348, 349, 723

Symbol dialog box, 119
symbols, working with, 119
system settings
 date format, 141
 list separator, 100

T

T function, 732
Table dialog box, 263
TABLE function, 359, 361
tables, dynamic crosstab, 413–414
TAN function, 728
TANH function, 728
taxes
 interest payments calculations, effect on, 320
 lookup formulas, using, 110, 214, 216
TBILLEQ function, 723
TBILLPRICE function, 723
TBILLYIELD function, 723
TDIST function, 731
temperature measurement unit conversions, 277
term (interest period)
 accumulation calculations, 298, 303
 amortization calculations, 310
 defined, 294
test score calculations using lookup formulas, 224
text boxes
 charts, in, 446–447
 fill color, 16
TEXT function, 121–122, 123, 174, 732
text manipulation functions.xls (on the CD-ROM), 773
text operations
 acronyms, returning, 137, 669–670
 case conversion, 98, 119, 126–127, 631, 693
 cell ranges, returning longest string in, 405
 cells, determining if text-containing, 115–117, 137, 671–672
 character codes, working with, 117–118
 characters, removing nonprinting, 125–126
 characters, repeating, 123
 characters, returning from beginning of strings, 102, 127, 133
 characters, returning from end of strings, 127, 133
 characters, returning from middle of strings, 110, 127, 134–135
 characters, returning from nth position in strings, 137, 672–673
 characters, returning from strings based on separator characters, 137
 characters, returning position in string, 128–129
 characters, reversing order of, 137, 610–611, 668–669
 characters, scrambling, 137, 669
 column letters, returning for values contained in cells, 132
 comparison, 119–120, 130–131, 137, 207
 concatenation, 36, 38, 119–120, 568, 629
 counting characters in text strings, 126, 130, 192–193
 counting most frequently occurring text, 191
 counting occurrences of substrings, 130–131, 192–193
 counting text cells, 185, 187, 392–393
 currency values, displaying as text, 122–123
 data entry restriction by text length, 536
 data entry restriction to text only, 538
 dates, converting strings to, 151, 153
 dates, returning strings from, 157, 162
 filenames, extracting from path, 132–133
 formula conversion to text, 35
 formulas, copying text from, 544
 functions related to, 110, 115, 136–137, 732
 histograms, 123–124
 IF function, using in, 115, 132, 133
 length of strings, returning, 130, 219, 620
 megaformulas, using, 544–547
 non-numeric characters, removing from strings, 410
 number formats, displaying N/A for using custom, 749
 number formats, displaying text in quotes using custom, 749
 number formats, displaying text with numbers using custom, 753
 number formats, repeating text using custom, 749
 numbers, displaying as text, 114–115, 121–122
 numbers, spelling out, 137, 674
 numeric values, converting text strings to, 407
 pattern matching, 137, 670–671
 personal names, changing case, 126–127
 personal names, extracting portions of, 134–135
 personal names, removing middle, 544–547
 replacement, 127–128, 129–130
 spaces, removing excess, 125–126
 spaces, returning positions of, 548–552
 splitting strings, 136
 substrings, returning position of, 128–129
 summing based on text comparisons, 207
 symbols, working with, 119
 time serial numbers, converting text strings to, 168, 170
 time values, returning text from, 174
 words, counting, 135–136
 words, returning from strings, 133–135

- text wrap, 9, 15, 120–121
- text-to-speech feature, 775
- Thanksgiving Day, calculating, 164
- thermometer chart.xls (on the CD-ROM), 772
- thousands separator, 298, 733, 735
- time
 - 24-hour values, exceeding, 146, 169, 170–172
 - AM/PM, determining, 634, 636
 - Analysis ToolPak add-in features, 112
 - cells containing, referencing, 146
 - clocks, creating in charts, 483–485, 486
 - current, displaying, 168–169
 - current, entering, 15, 168
 - current, returning, 122, 634
 - data validation criteria, 536
 - dates, entering with, 145
 - days, as fractional, 142–144
 - days, associating with, 145
 - decimal units, converting, 174
 - differences, calculating, 172–173
 - displaying, 168–170
 - entering, 15, 144–145, 168
 - format, applying, 145–146, 734, 736
 - format, custom, 146, 741, 752
 - formats recognized, 144–145
 - functions related to, 109, 168, 718–719
 - Greenwich Mean Time (GMT), expressing in local, 175
 - IF function in time stamp calculations, 432
 - measurement unit conversions, 272, 277
 - midnight, spanning, 173
 - military time, converting from, 174
 - negative values, 140, 173
 - NOW function in time stamp calculations, 432
 - periods, non–time–of–day, 177–178
 - replacement characters for invalid values, 50, 562
 - rounding time values, 176–177
 - serial numbers, 142–144
 - serial numbers, converting text strings to, 168, 170, 718
 - serial numbers, converting to hours, 168, 718
 - serial numbers, converting to minutes, 168, 718
 - serial numbers, converting to seconds, 168, 718
 - serial numbers, returning for current date and time, 150, 168, 718
 - serial numbers, returning for current time, 150
 - serial numbers, returning for particular time, 168, 169, 718
 - smallest unit possible, 143
 - stamps, 168
 - stamps using intentional circular references, 432
 - summing operations, 170–172, 175
 - text, returning, 174
 - text recognized as, 145
 - Visual Basic for Applications (VBA), working with in, 634, 636
 - zones, converting between, 175–176
- TIME function (Excel), 168, 169, 175, 718
- Time function (Visual Basic for Applications (VBA)), 634
- time sheet.xls (on the CD-ROM), 766
- time stamp.xls (on the CD-ROM), 771
- Time Tracker utility (on the CD-ROM), 159
- TIMEVALUE function, 168, 170, 174, 718
- TINV function, 731
- TODAY function
 - conditional formatting, in, 523
 - described, 109, 719
 - displaying current date using, 150
 - update, automatic, 150
 - volatile nature of, 111
- toolbars. *See also* menus
 - customizing, 21
 - docking, 13
 - floating, 13
 - predefined, 13
- Tools ⇄ Add-Ins, 21
- Tools ⇄ Auditing, 24
- Tools ⇄ Conditional Sum, 205
- Tools ⇄ Data Analysis, 198
- Tools ⇄ Formula Auditing, 24
- Tools ⇄ Formula Auditing ⇄ Evaluate Formula, 580
- Tools ⇄ Formula Auditing ⇄ Formula Auditing Mode, 574
- Tools ⇄ Formula Auditing ⇄ Show Formula Auditing Toolbar, 576
- Tools ⇄ Goal Seek, 54
- Tools ⇄ Macro ⇄ Macros, 606
- Tools ⇄ Macro ⇄ Record New Macro, 596
- Tools ⇄ Macro ⇄ Visual Basic Editor, 588
- Tools ⇄ Options, 15
- Tools ⇄ Protection ⇄ Protect Sheet, 25
- Tools ⇄ Protection ⇄ Protect Workbook, 26
- Tools ⇄ References, 604
- total interest.xls (on the CD-ROM), 772
- transition formula evaluation, 74–75, 261, 700–701, 702
- TRANSPOSE function, 382–383, 388–389, 688, 725
- Treasury bills, 345, 723
- TREND function, 234, 391, 731
- trends
 - chart trendlines, 469
 - chart trendlines, coefficient of determination, 471
 - chart trendlines, data series *versus*, 470

chart trendlines, decimal places setting, 471
 chart trendlines, exponential, 474, 477–478
 chart trendlines, intercept calculations, 470, 471–472
 chart trendlines, linear, 469, 470–471
 chart trendlines, logarithmic, 474, 475–476
 chart trendlines, moving average option, 474
 chart trendlines, nonlinear, 474–479
 chart trendlines, polynomial, 474, 478–479
 chart trendlines, power, 474, 476–477
 chart trendlines, predicted values
 calculations, 472–473
 chart trendlines, R-squared values, 470, 471, 474
 chart trendlines, slope calculations, 471–472
 downward, identifying, 527
 formatting, conditional, 527
 upward, identifying, 527
 trigonometry. *See also* geometry
 Analysis ToolPak add-in features, 112
 arccosines, 726
 arcsines, 726
 arctangents, 726
 cosines, hyperbolic, 726
 cosines, inverse hyperbolic, 726
 cosines of complex numbers, 720
 functions related to, 109, 726–728
 logarithms, returning, 726
 radians, assumed in trigonometry functions, 109
 sines, hyperbolic, 727
 sines, inverse hyperbolic, 726
 sines of angles, 103, 727
 tangents, 728
 tangents, hyperbolic, 728
 tangents, inverse hyperbolic, 726
 TRIM function, 125–126, 136, 528, 732
 TRIMMEAN function, 731
 TRUE function, 724
 TRUE/FALSE logical values, working with. *See*
 comparison operations, logical
 TRUNC function, 285, 288–289, 728
 TTEST function, 731
 two-column lookup.xls (on the CD-ROM), 768
 two-way lookup.xls (on the CD-ROM), 768
 TYPE function, 116, 724
 TypeName function, 697

U

UCase function, 631, 693
 UDFs. *See* user-defined functions (UDFs)
 UI. *See* user interface (UI)
 Unicode character set, 118
 Union function, 651
 union operator, 36
 unique random integers.xls (on the CD-ROM), 771

unit conversion tables.xls (on the CD-ROM), 768
 unit conversions, measurement. *See*
 measurement unit conversions
 UPPER function, 126, 732
 Usenet spreadsheet newsgroup, 761
 User function, 600–602
 user interface (UI), 11–16
 user name, displaying using custom function, 600–602
 user-defined functions (UDFs), 111, 588, 599.
 See also Visual Basic for Applications (VBA), functions
 UserForms versus dialog sheets, 10
 UserName property, 597

V

Val function, 657
 validation, data
 arrays, determining range values validity
 using, 406
 blank entries, allowing/disallowing, 537
 conditional formatting, applying to invalid
 data, 532–533
 criteria, specifying, 534–535
 criteria, types, 536–537
 data entry restriction by text length, 536
 data entry restriction to dates, 536
 data entry restriction to larger values than
 previous cell, 538–539
 data entry restriction to list entries, 89, 352
 data entry restriction to nonduplicate
 values, 539
 data entry restriction to number ranges, 536
 data entry restriction to specific characters,
 539
 data entry restriction to text only, 538
 data entry restriction to times, 536
 data entry restriction to whole numbers, 536
 functions, using custom, 540
 invalid data, allowing entry of, 537
 invalid data, circling, 537
 messages, custom, 534
 removing, 536
 versions of Excel available in, 533
 #VALUE! errors, 50, 567, 610
 VALUE function, 407, 732
 VAR function, 731
 VARA function, 731
 variable rate analysis.xls (on the CD-ROM), 770
 variable rate loan amortization schedule.xls (on
 the CD-ROM), 770
 variables
 assigning values to, 622–623, 628–629
 case sensitivity, 596
 cell ranges, assigning to object variables,
 649–650

continued

- variables *continued*
 - charting functions, double-variable, 466–467
 - charting functions, single-variable, 462–466
 - constants, 626–627
 - counters, 615, 639–640
 - data type, displaying, 615
 - dates in, 627–628
 - Debug.Print statements, monitoring value using, 612–613
 - declaring, 594, 620, 624–626
 - equations, in, 283
 - Immediate window, monitoring values in, 612–613
 - monitoring values in loops, 611–613
 - object variables, 649–650
 - option explicit setting, 625, 626
 - pop-up boxes, monitoring value using, 610–612
 - Range object, 649, 654
 - spelling errors, 625
 - string, 621, 627
- VARP function, 731
- VARPA function, 731
- VB Editor. *See* Visual Basic Editor (VB Editor)
- VBA. *See* Visual Basic for Applications (VBA)
- VBA Project Summary Report utility (on the CD-ROM), 581
- VDB function, 348, 349–350, 723
- versions of Excel
 - add-in differences, 618
 - character support differences, 119
 - conditional formatting differences, 513
 - data validation differences, 533
 - date support differences, 140, 148–149
 - dialog sheets, inserting from previous, 10
 - error checking, background, 578–579
 - error value printing differences, 524, 563
 - Formula Palette removed from Excel 2002, 32, 106
 - function categories inherited from previous, 111
 - history, 5–6
 - leap year bug, 147
 - number, returning, 657
 - pivot table differences, 495, 498, 499, 504
 - rows, maximum number supported, 9, 238
 - sub-versions, 7
 - Visual Basic Editor differences, 588
 - Visual Basic for Applications (VBA) compatibility, 632
- View ⇄ Full Screen, 14
- view options, 14, 573–574
- View ⇄ Sized with Window, 443
- View ⇄ Toolbars ⇄ Customize, 11, 21
- View ⇄ Toolbars ⇄ Forms, 461
- VisiCalc, 4
- Visual Basic Editor (VB Editor). *See also* Visual Basic for Applications (VBA)
 - activating, 8, 588–589
 - Auto List Members option, 631
 - case conversion, automatic, 596
 - code, entering, 595–596
 - code window, 590, 593
 - Debug button, 614
 - Editor tab, 626, 631
 - functions, listing, 631–632
 - Immediate window, 590, 608, 612–613, 615
 - menu bar, 589
 - Project window, 590–591
 - Properties window, 590, 592
 - Redo, 596
 - Require Variable Declaration option, 626
 - shortcut menus, 589
 - Sub procedures, executing from, 614
 - syntax error checking, 596, 610
 - toolbars, 589
 - Undo, 596
 - versions of Excel, differences, 588
 - windows, customizing, 589
 - windows, minimizing/maximizing, 593–594
- Visual Basic for Applications (VBA). *See also* Visual Basic Editor (VB Editor); *specific functions, keywords, methods and properties*
 - acronyms, returning, 137, 669–670
 - add-ins, creating, 616–618
 - addition operator, 629
 - And operator, 629
 - application names, returning, 657
 - Application object, 664
 - arrays, declaring, 630–631
 - arrays, returning from functions, 418–421, 687–688
 - arrays of consecutive integers arranged randomly, generating, 689–690
 - arrays of range contents arranged randomly, generating, 691–692
 - breakpoints, 614, 615, 616
 - case conversion, 631, 693
 - case sensitivity, 596, 603
 - cell range address, returning, 647
 - cell ranges, assigning to object variables, 649–650
 - cell ranges, combining, 651
 - cell ranges, looping through using For Each-Next, 643–644
 - cell ranges, randomizing, 691–692
 - cell ranges, referencing, 644–646
 - cell ranges, returning cells-used subset, 651–652
 - cell ranges, returning columns of, 649
 - cell ranges, returning font properties, 648, 658

- cell ranges, returning intersections, 650–651
- cell ranges, returning names of, 648
- cell ranges, returning number format, 648, 659
- cell ranges, returning number of columns in, 648
- cell ranges, returning number of rows in, 648
- cell ranges, returning parent of Range object, 655–656
- cell ranges, returning Range objects for, 646, 651–652
- cell ranges, returning rows from, 649
- cell ranges, returning worksheets containing, 647
- cell ranges, summing squares of values in, 643–644
- cell ranges, summing visible, 210, 676–677
- cell ranges, testing for hidden, 654–655
- cell ranges, testing for hidden rows/columns in, 649, 654–655
- cells, counting between two values, 675
- cells, counting total in range, 644, 647
- cells, counting total in worksheet, 645
- cells, counting visible, 210, 675–676
- cells, determining data type, 659–660
- cells, determining if formula contained in, 647, 654
- cells, determining if text contained in, 137, 671–672
- cells, referencing, 645
- cells, returning font properties of, 648, 658
- cells, returning formatting information of, 648, 657–659
- cells, returning formulas contained in, 646–647
- cells, returning last nonempty, 680–681
- cells, returning names of, 648
- cells, selecting random, 663–664
- cells containing formulas, identifying, 647, 654
- charts, using in, 461, 484–485
- code, case sensitivity, 596
- code, copying, 598
- code, entering manually, 595–596
- code, entering using macro recorder, 596–598
- code, indenting, 595
- code, inserting Excel worksheet functions in, 632
- code, types of, 594
- color fills, returning color index number for, 658–659
- comments, 620, 622
- commission calculations, 664–667
- comparison operations, 137, 629
- concatenation operator, 629
- constants, 626–627
- counters, 615, 639–640
- counting functions, 210, 674–676
- data types, 624
- data types, automatic assignment, 623
- data types, declaring, 625
- data types, determining, 659–660
- data types, displaying, 615
- data types returned by functions, specifying, 603, 620
- data validation, using in, 540, 661
- dates, working with, 627–628, 677–680
- day of week, calculating date of next, 678
- day of week, returning as integer, 679
- days between two dates, calculating, 679
- days of dates, returning, 679
- Debug.Print statements, monitoring
 - variables using, 612–613
- declarations, 594
- division operator, 629
- Do Until constructs, 640–641
- Do While constructs, 639–640
- dollars, spelling out, 137, 674
- Eqv operator, 629
- error handling, 641–643
- error values, returning, 685–686
- errors, ignoring, 641–642
- errors, logical, 610
- errors, runtime, 610, 613–615
- errors, syntax, 596, 610, 623
- Excel version number, returning, 657
- Excel worksheet functions, using, 632
- execution control, 633–643
- exponentiation operator, 629
- expressions, 628
- expressions, assignment, 628–629
- flow control, 633–643
- font properties, returning, 648, 658
- For Each-Next constructs, 643–644
- formatting information, returning, 648, 657–659
- For-Next constructs, 620–621, 637–639, 646
- function arguments, 605
- function arguments, Boolean, 697
- function arguments, determining if missing, 693, 697
- function arguments, determining type, 697
- function arguments, maximum number of, 694
- function arguments, mixing type, 694
- function arguments, representing by variable lists, 603
- function categories, assigning to, 111, 607–609
- function categories listed, 609

continued

- Visual Basic for Applications (VBA) *continued*
- Function procedures, 594–595, 602, 619–621
 - Function procedures, creating using macro recorder, 596
 - Function procedures in add-ins, 605
 - functions, built-in, 631–633
 - functions, calling from Sub procedures, 613–615
 - functions, creating, 600, 620
 - functions, creating descriptions for, 606–607
 - functions, custom, 99, 599–600
 - functions, data type returned by, 603, 620
 - functions, debugging, 609–616
 - functions, declaring, 602–603, 620
 - functions, ending, 603, 621
 - functions, exiting, 603
 - functions, in conditional formatting
 - formulas, 530–533
 - functions, in data validation, 540
 - functions, inserting in formulas, 104–106, 108, 605–609
 - functions, interest rate conversion, 315
 - functions, limitations of, 601
 - functions, listing, 631–632
 - functions, multifunctional, 660–662
 - functions, multisheet, 681–685
 - functions, naming, 603, 607
 - functions, naming with variable string
 - contents, 621
 - functions, optional arguments, 693
 - functions, passive nature of, 601
 - functions, private, 602
 - functions, public, 602
 - functions, recalculation control, 664
 - functions, referencing in other workbooks, 604–605
 - functions, returning arrays from, 418–421, 687–688
 - functions, static, 602
 - functions, storage location, 592, 600
 - functions, testing, 609–610
 - functions, worksheet changes using, 601
 - history, 6
 - If Err statements, 642
 - If-Then constructs, 620–621, 633–635
 - If-Then-Else constructs, 634, 635
 - Immediate window, monitoring variable
 - values in, 612–613
 - Imp operator, 629
 - indentation, 595
 - integer division operator, 60
 - integers, generating consecutive arranged
 - randomly, 689–690
 - integers, generating nonduplicated, 689–690
 - interest rate conversions, 315
 - Intersect method, 697
 - logical operators, 629
 - loop constructs, 611–613, 620–621, 636–641
 - math operations, 629
 - Mod operator, 629
 - module sheets, 8
 - modules, 588
 - modules, adding, 592, 600
 - modules, class, 593
 - modules, exporting, 593
 - modules, importing, 593
 - modules, naming/renaming, 592
 - modules, removing, 593
 - Monday, calculating date of next, 677–678
 - months of dates, returning, 679
 - MsgBox statements, 610–612
 - multiplication operator, 629
 - nodes, 590–591
 - Not operator, 629
 - Null values, testing for, 658
 - number format, returning, 648, 659
 - numbers, generating random, 662–664
 - object variables, 649–650
 - objects, 593
 - On Error statements, 641–643, 651
 - operators, 595, 629
 - Option Explicit statements, 625, 626
 - options, choosing among, 635–636
 - Or operator, 629
 - parent objects, returning, 647, 655–656
 - pop-up boxes, monitoring variables using, 610–612
 - procedures, 594
 - procedures, marking as volatile, 664
 - procedures, reports on, 581
 - procedures, types of, 20–21
 - projects, 590
 - projects, adding modules to, 600
 - projects, expanding/contracting, 590
 - projects, naming/renaming, 591, 605
 - projects, passwords, 590
 - projects, saving, 598
 - projects, workbooks as, 590
 - projects code window, 593
 - random number generation, 662–664
 - Range object, returning parent of, 655–656
 - sales commission calculations, 664–667
 - security, 618
 - Select Case constructs, 635–636, 666
 - speed considerations, 548, 599, 623, 625
 - squares, returning, 645
 - statements, 603
 - statements, commenting out, 622
 - statements, executing individually as Sub
 - procedures, 608
 - statements, executing individually in
 - Immediate window, 608, 615
 - statements, executing line-by-line, 614

- statements, lengthy, 595
 - statements, multi-line, 595, 634
 - statements, viewing execution in Immediate window, 590
 - string variables, 621, 627
 - Sub procedures, 594
 - Sub procedures, calling functions from, 613–615
 - Sub procedures, executing from VB Editor, 614
 - Sub procedures, executing statements as, 608
 - Sub procedures, listing available, 606
 - subtraction operator, 629
 - summing operations, 210, 643–644, 676–677, 694–698
 - text comparison operations, 137
 - text strings, determining if cells contain, 137, 671–672
 - text strings, extracting nth element from, 137, 672–673
 - text strings, pattern matching, 137, 670–671
 - text strings, returning acronyms from, 137, 669–670
 - text strings, reversing character order, 137, 610–611, 668–669
 - text strings, scrambling characters in, 137, 669
 - time, working with, 634, 636
 - user name display, 600–602
 - uses, 587
 - variables, 622
 - variables, assigning values to, 622–623, 628–629
 - variables, declaring, 594, 620, 624–626
 - variables, displaying data type, 615
 - variables, misspelled, 625
 - variables, monitoring values in loops, 611–613
 - variables, object, 649–650
 - variables, option explicit setting, 625, 626
 - variables, Range object, 649, 654
 - variables, storing dates in, 627–628
 - variables, string, 621, 627
 - versions of Excel, compatibility with earlier, 632
 - Volatile method, 657–658, 664
 - week of month, returning integer for, 678–679
 - wildcard characters, 670
 - words, reserved, 623
 - workbooks, returning names of, 656
 - workbooks, storage in, 8
 - worksheet index, returning, 684–685
 - worksheets, as, 8
 - worksheets, relative references to, 683–685
 - worksheets, returning maximum value across multiple, 682–683
 - worksheets, returning names of, 655–656
 - worksheets, working with multiple, 681–685
 - Xor operator, 629
 - years between two dates, calculating, 679
 - years of dates, returning, 679
 - Visual Basic toolbar, 588
 - VLOOKUP function
 - arguments, 213–214, 224
 - case-sensitivity, 222
 - closest match lookups, 230
 - described, 212, 213–214, 725
 - lookup tables, working with multiple, 223
 - missing data errors, 565
 - sales commissions calculations, in, 665
 - tax rate calculations, in, 214
 - values, looking up exact, 220
 - values, looking up in columns other than first, 221
 - wildcards, using, 214
 - VMONTHNAMES function, 688
 - Volatile method, 657–658, 664
 - volume calculations
 - geometry, 282–283
 - measurement unit conversions, 272, 275
- ## W
- Web resources. See Internet resources
 - WEEKDAY function, 150, 158–161, 523, 719
 - WEEKNUM function, 150, 719
 - WEIBULL function, 731
 - wildcard characters
 - counting operations, in, 186
 - examples, 252
 - lookup formulas, in, 214, 215, 217
 - searches using, 129, 137
 - Visual Basic for Applications (VBA), 670
 - Window ⇄ Freeze Panes, 239
 - Window ⇄ Hide, 8
 - windows
 - charts, sizing to, 443, 479
 - charts, viewing embedded charts in separate, 480
 - formula/result display, showing in separate, 573–574
 - moving, disallowing, 26
 - resizing, disallowing, 26
 - Visual Basic Editor, 589, 593–594
 - workbooks, displaying in multiple, 8
 - WK1 files, 699
 - WK3 files, 699
 - WK4 files, 699
 - WKS files, 699
 - work days.xls (on the CD-ROM), 766
 - Workbook Link Report utility (on the CD-ROM), 581

Workbook Summary Report utility (on the CD-ROM), 581

WORKBOOKNAME function, 656

workbooks

- active, 8
- add-ins, creating from, 616–618
- calculation mode setting, 431
- code window for, viewing, 590
- color formatting, 516
- display options, 8
- hiding, 8
- introduced, 7–8
- names, returning, 656
- names, spaces in, 44
- open, maximum, 8
- projects, as, 590
- protecting, 26
- SERIES formulas, referencing in, 442
- summary reports, 581
- Visual Basic for Applications (VBA) module storage in, 8
- windows, displaying in multiple, 8
- worksheets contained in, default, 10
- worksheets contained in, maximum, 8

WORKDAY function, 150, 156, 719

worksheet databases. *See* databases (worksheet lists)

worksheet functions, 97. *See also* functions

worksheet lists. *See* databases (worksheet lists)

Worksheet Map utility (on the CD-ROM), 581

worksheet outlines. *See* outlines

WorksheetIndex function, 684–685

worksheets. *See also* cells; columns; rows

- active, 8
- cell ranges, referencing in other worksheets, 44–45
- cell ranges, returning worksheet containing, 647
- cell ranges spanning multiple, 65–66
- chart sheets, 8, 10
- charts, referring to other worksheets in, 442
- charts, storing multiple on, 479–480
- clocks, displaying in, 484
- conditional formatting, referencing in, 520–521
- copying within same workbook, 78–79
- database storage in, 239
- deleting, named object considerations, 66, 79
- dialog box controls, placing on, 19–20
- dialog sheets, 8, 10
- draw layer, 10, 13, 18–20
- formulas in, listing all, 581
- index, returning, 684–685
- inserting, 66

- introduced, 9
- macro sheets, 8, 10
- maps, inserting in, 19
- menu bar, 11
- multiple, working with, 9, 681–685
- names, changing, 8
- names, referencing, 67, 89
- names, returning, 655–656
- navigating using cell names, 58
- number format storage with, 738
- number of, default, 10
- number of, maximum, 8
- password protecting, 25
- pivot tables, saving in, 497
- references, relative, 683–685
- references in conditional formatting, 520–521
- schedule storage in, 352
- SERIES formulas, referencing in, 442
- types, 8
- Visual Basic for Applications (VBA) modules, as, 8

X

XDATE functions, 679

XIRR and XNPV functions.xls (on the CD-ROM), 770

XIRR function, 325, 366–368, 723

.xla files, 617

xlErr constants, 686

XLM macro functions, 465

XLM macro language, 587

XLM macro sheets, 8, 10

.xls files, 7

XNPV function, 325, 366, 367–368, 723

xy sketch.xls (on the CD-ROM), 772

Y

YEAR function, 150, 151, 161, 719

YEARFRAC function, 150, 157, 719

YIELD function, 723

YIELDDISC function, 723

YIELDMAT function, 723

Z

zero

- division by, 50, 564
- number format manipulations involving, 746–747, 753–754

Zero values option, 184

Zip Code number format, 737, 739, 751

ZTEST function, 731

Save \$30.00!

Power Utility Pak 2000

"The Excel tools Microsoft forgot"
A \$39.95 value. Yours for only \$9.95.

PRO-QUALITY TOOLS

The PUP 2000 add-in is a dynamite collection of 50 general purpose Excel utilities, plus 40 new worksheet functions. Download the trial version from the URL listed below. If you like it, use this coupon to receive \$30.00 off the normal price.

VBA SOURCE CODE IS AVAILABLE

You can also get the complete VBA source files for only \$20.00. Learn how the utilities and functions were written, and pick up useful tips and programming techniques in the process. This is a must for all VBA programmers!

YES! Please send Power Utility Pak 2000 to...

Name: _____

Company: _____

Address: _____

City: _____ State: _____ Zip: _____

Daytime Phone: _____ E-mail: _____

Check one:

- PUP 2000 Licensed Version (only \$9.95 + \$4.00 s/h) . \$13.95
- Developer's Pak: Licensed Version (\$9.95 + \$4.00 s/h) + VBA Source (\$20.00) \$33.95

Delivery method:

- Send me the disk
- Send download instructions to my e-mail address (shipping/handling fee still applies)

Credit Card No: _____ Expires: _____

Make check or money order (U.S. funds only) payable to:

JWalk & Associates Inc.
P.O. Box 12861
La Jolla, CA 92039-2861

Download the latest version of PUP from: <http://j-walk.com/ss/>

- PUP 2000 is compatible with Excel 97, Excel 2000, and Excel 2002.

Hungry Minds, Inc.

End-User License Agreement

READ THIS. You should carefully read these terms and conditions before opening the software packet(s) included with this book (“Book”). This is a license agreement (“Agreement”) between you and Hungry Minds, Inc. (“HMI”). By opening the accompanying software packet(s), you acknowledge that you have read and accept the following terms and conditions. If you do not agree and do not want to be bound by such terms and conditions, promptly return the Book and the unopened software packet(s) to the place you obtained them for a full refund.

1. **License Grant.** HMI grants to you (either an individual or entity) a non-exclusive license to use one copy of the enclosed software program(s) (collectively, the “Software”) solely for your own personal or business purposes on a single computer (whether a standard computer or a workstation component of a multi-user network). The Software is in use on a computer when it is loaded into temporary memory (RAM) or installed into permanent memory (hard disk, CD-ROM, or other storage device). HMI reserves all rights not expressly granted herein.
2. **Ownership.** HMI is the owner of all right, title, and interest, including copyright, in and to the compilation of the Software recorded on the disk(s) or CD-ROM (“Software Media”). Copyright to the individual programs recorded on the Software Media is owned by the author or other authorized copyright owner of each program. Ownership of the Software and all proprietary rights relating thereto remain with HMI and its licensors.
3. **Restrictions On Use and Transfer.**
 - (a) You may only (i) make one copy of the Software for backup or archival purposes, or (ii) transfer the Software to a single hard disk, provided that you keep the original for backup or archival purposes. You may not (i) rent or lease the Software, (ii) copy or reproduce the Software through a LAN or other network system or through any computer subscriber system or bulletin-board system, or (iii) modify, adapt, or create derivative works based on the Software.
 - (b) You may not reverse engineer, decompile, or disassemble the Software. You may transfer the Software and user documentation on a permanent basis, provided that the transferee agrees to accept the terms and conditions of this Agreement and you retain no copies. If the Software is an update or has been updated, any transfer must include the most recent update and all prior versions.

4. **Restrictions on Use of Individual Programs.** You must follow the individual requirements and restrictions detailed for each individual program in Appendix E of this Book. These limitations are also contained in the individual license agreements recorded on the Software Media. These limitations may include a requirement that after using the program for a specified period of time, the user must pay a registration fee or discontinue use. By opening the Software packet(s), you will be agreeing to abide by the licenses and restrictions for these individual programs that are detailed in Appendix E and on the Software Media. None of the material on this Software Media or listed in this Book may ever be redistributed, in original or modified form, for commercial purposes.

5. **Limited Warranty.**

- (a) HMI warrants that the Software and Software Media are free from defects in materials and workmanship under normal use for a period of sixty (60) days from the date of purchase of this Book. If HMI receives notification within the warranty period of defects in materials or workmanship, HMI will replace the defective Software Media.
- (b) **HMI AND THE AUTHOR OF THE BOOK DISCLAIM ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE PROGRAMS, THE SOURCE CODE CONTAINED THEREIN, AND/OR THE TECHNIQUES DESCRIBED IN THIS BOOK. HMI DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE.**
- (c) This limited warranty gives you specific legal rights, and you may have other rights that vary from jurisdiction to jurisdiction.

6. **Remedies.**

- (a) HMI's entire liability and your exclusive remedy for defects in materials and workmanship shall be limited to replacement of the Software Media, which may be returned to HMI with a copy of your receipt at the following address: Software Media Fulfillment Department, Attn.: *Excel 2002 Formulas*, Hungry Minds, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, or call 1-800-762-2974. Please allow four to six weeks for delivery. This Limited Warranty is void if failure of the Software Media has resulted from accident, abuse, or misapplication. Any replacement Software Media will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

- (b) In no event shall HMI or the author be liable for any damages whatsoever (including without limitation damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising from the use of or inability to use the Book or the Software, even if HMI has been advised of the possibility of such damages.
 - (c) Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation or exclusion may not apply to you.
7. **U.S. Government Restricted Rights.** Use, duplication, or disclosure of the Software for or on behalf of the United States of America, its agencies and/or instrumentalities (the "U.S. Government") is subject to restrictions as stated in paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFARS 252.227-7013, or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and in similar clauses in the NASA FAR supplement, as applicable.
8. **General.** This Agreement constitutes the entire understanding of the parties and revokes and supersedes all prior agreements, oral or written, between them and may not be modified or amended except in a writing signed by both parties hereto that specifically refers to this Agreement. This Agreement shall take precedence over any other documents that may be in conflict herewith. If any one or more provisions contained in this Agreement are held by any court or tribunal to be invalid, illegal, or otherwise unenforceable, each and every other provision shall remain in full force and effect.

CD ROM Installation Instructions

The CD-ROM that you find in the back of this book contains software for both Macintosh and Windows 95/98/NT/ME/2000/XP users. You will need a copy of Excel 2002 to use the examples and software on the CD-ROM. Read the “What’s on the CD-ROM” appendix for complete information about the stuff on the CD-ROM.

Note: You do not need to install all the items on the CD-ROM. Just install the programs that appeal to you.

To start the CD-ROM using Windows, follow these steps:

1. Insert the CD-ROM into your computer’s CD-ROM drive.
2. Launch Windows Explorer.
3. Click Browse to browse the CD. This enables you to access the author-created files. (Note that you must save files to your hard drive if you make changes.)

For detailed information about installing the demonstration programs from the CD, please see the “What’s on the CD-ROM” appendix.