Code samples available on the Companion Web site

Thomas Myer

Apple® Automator with AppleScript®

Bible

Harness Mac OS® X's built-in automation tools

Automate repetitive or time-consuming tasks

Write scripts that save you time and money

The book you need to succeed!

Apple® Automator with AppleScript® Bible

Apple® Automator with AppleScript® Bible

Thomas Myer

WILEY
Wiley Publishing, Inc.

Apple® Automator with AppleScript® Bible

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-52586-9

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, 201-748-6011, fax 201-748-6008, or online at http://www.wiley.com/go/permissions.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (877) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Library of Congress Control Number: 2009938256

Trademarks: Wiley and related trade dress are registered trademarks of Wiley Publishing, Inc., in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

To my divine spouse Hope, for loving me anyway.

About the Author

Thomas Myer is a consultant, author, and developer who lives in Austin, TX. He founded Triple Dog Dare Media in 2001, and since then he's been busy creating applications, training clients, writing books and articles, and helping clients do more with less. His practice focuses on two things: making companies more productive with automation solutions, and helping them adopt social media tools and practices "inside the firewall."

To find out more about Thomas, you can follow him on Twitter. His handle is @myerman. You can also visit his Web site at `http://www.tripledogs.com`.

Credits

Senior Acquisitions Editor
Stephanie McComb

Project Editor
Chris Wolfgang

Technical Editor
Dennis Cohen

Copy Editor
Marylouise Wiack

Editorial Director
Robyn Siesky

Editorial Manager
Cricket Krengel

Business Manager
Amy Knies

Senior Marketing Manager
Sandy Smith

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Barry Pruett

Project Coordinator
Kristie Rees

Graphics and Production Specialists
Ana Carrillo
Andrea Hornberger

Quality Control Technician
Robert Springer

Proofreading and Indexing
C.M. Jones
Sherry Massey

Contents at a Glance

Preface xxix
Acknowledgments xxx
Introduction xxxi
Quick Start: Dive into Automator and AppleScript 1

Part I: The Fundamentals of Automation. 29

Chapter 1: What Is Automation? 31
Chapter 2: Automation from a User's Perspective 47
Chapter 3: Automator Basics 61
Chapter 4: Advanced Automator Topics 83

Part II: A Detailed Look at AppleScript 107

Chapter 5: AppleScript Basics 109
Chapter 6: AppleScript Objects and Dictionaries 125
Chapter 7: Working with Variables and Properties 143
Chapter 8: Operators, Expressions, and Statements 159
Chapter 9: Conditionals and Loops 179
Chapter 10: Handling User Input 197
Chapter 11: AppleScript Subroutines 213
Chapter 12: Applets and Droplets 227
Chapter 13: Folder Actions 241
Chapter 14: AppleScript Studio 255

Part III: Automation Projects 269

Chapter 15: Ten Automation Projects for Files and Folders 271
Chapter 16: Ten Automation Projects for Music and Audio Files 305
Chapter 17: Ten Automation Projects for Photos and Images 335
Chapter 18: Ten Automation Projects for Text Files 371
Chapter 19: Ten Custom Automation Projects 399

Part IV: Appendixes 427

Appendix A: Automator Resources 429
Appendix B: AppleScript Resources 439
Appendix C: AppleScript Reference 451
Appendix D: Automator Actions and Variables 485

Contents

Preface xxix

Acknowledgments xxx

Introduction xxxi

Quick Start: Dive into Automator and AppleScript 1

Why Is Automation Important? ... 2
Who Uses Automation? ... 3
Introducing Automator ... 4
Actions ... 6
Variables ... 8
Creating your first basic workflow ... 9
Customizing your first workflow ... 14
Options for saving your workflow ... 17
Introducing AppleScript ... 17
Summary ... 28

Part I: The Fundamentals of Automation 29

Chapter 1: What Is Automation? 31

Why You Should Use Automation ... 31
A Brief History of Automation ... 33
Differences between mechanization and automation ... 34
Negative connotations of automation ... 35
Positive changes due to automation ... 35
Examples of Automation ... 36
Data entry or collection ... 36
Data analysis ... 37
Data munging ... 37
Data extraction ... 39
Data transformation ... 39

Data integration ... 39
Launching applications and scripts ... 40
Communications ... 41
Setting up reminders ... 42
Systems maintenance and backups ... 43
Why Automate? ... 43
Save time ... 43
Save effort ... 44
Simplify a process ... 45
Reduce errors ... 45
Save on manpower ... 46
Save money ... 46
Summary ... 46

Chapter 2: Automation from a User's Perspective ... 47

The Mac User's Viewpoint ... 47
Typical activities to automate ... 48
Taking inventory ... 49
The Professional's Viewpoint ... 53
The software developer's needs ... 53
The system administrator's needs ... 54
The creative's needs ... 55
The Small Business Owner's Viewpoint ... 56
What processes need automating? ... 57
What files and data need automating? ... 58
Where's the ROI? ... 59
Summary ... 60

Chapter 3: Automator Basics ... 61

Opening Automator for the First Time ... 61
The Automator Interface ... 62
The Toolbar ... 63
The Actions library ... 63
The Variables library ... 65
The Footer ... 67
What Are Workflows? ... 70
What Kinds of Workflows Are Possible? ... 71
What Are Actions? ... 74

What Else Can I Do with Actions? ... 76
How Can Workflows Be Saved? ... 78
Save as workflow ... 78
Save as application ... 79
Save as plugin ... 80
What Are Templates? ... 81
Summary ... 82

Chapter 4: Advanced Automator Topics ... 83

Working with Variables ... 84
What is a variable? ... 84
Types of variables ... 86
Date & Time variables ... 87
Locations variables ... 88
System variables ... 88
Text & Data variables ... 89
User variables ... 91
Utilities variables ... 91
Adding variables to workflows ... 93
Using variables in a workflow ... 93
Working with Loops ... 94
What is a loop? ... 94
Setting up a loop ... 95
A Closer Look at Plugins ... 98
Finder plugins ... 98
Calendar plugins ... 99
Other plugins ... 101
Recording Manual Events ... 102
First things first: Set up accessibility ... 102
Recording a manual event ... 103
Summary ... 106

Part II: A Detailed Look at AppleScript 107

Chapter 5: AppleScript Basics ... 109

What Is AppleScript? ... 109
A Brief History of AppleScript ... 110
Why AppleScript? ... 112

AppleScript Editor 112
Starting AppleScript Editor 113
The AppleScript Editor GUI 113
Basic operations 114
Comments and error messages 114
Saving your scripts 116
The AppleScript Editor Menus 117
AppleScript Editor menu 118
File menu 120
Edit menu 121
View menu 121
Script menu 121
Font menu 122
Format menu 122
Window menu 122
Help menu 123
The Script Menu 124
Summary 124

Chapter 6: AppleScript Objects and Dictionaries 125

What Are Objects? 125
AppleScript is an object-oriented language 126
Everything you work with is an object 127
Objects include properties and actions 129
Even the script you're working with is an object 130
A quick tour of script objects 131
What Are Dictionaries? 133
How to Use Dictionaries 134
A Quick Tour of Dictionaries 135
Viewing by suite 138
Viewing by containment 139
Viewing by inheritance 140
Looking Up an AppleScript 141
Summary 142

Chapter 7: Working with Variables and Properties 143

What Are Variables? 143
Creating variables 144
Getting the value of a variable 146

Working with lists 146
Working with records 151
Global variables 152
What Are Properties? 153
Creating Three Simple Scripts 154
Script 1: Play random iTunes track 154
Script 2: Copy files from one directory to another 155
Script 3: View today's events in iCal 156
Summary 157

Chapter 8: Operators, Expressions, and Statements 159

What Are Operators? 159
Logical operators 160
Logical conjunction (and) 161
Logical disjunction (or) 161
Negation (not) 161
Equality 162
Inequality 162
Greater than 163
Less than 163
Greater than or equal to 163
Less than or equal to 163
Mathematical operators 164
Multiplication (*) 164
Addition (+) 164
Subtraction (-) 164
Division (/) 164
Integral division (div) 165
Modulus (mod) 165
10 mod 3 -- results in 1Exponentiation (^) 165
Other operators 165
Concatenation 165
Containment 166
Object reference 167
Operator precedence 167
What is coercion? 168
Considering and ignoring 169

What Are Expressions? ... 170
What Are Statements? ... 172
Writing Scripts ... 173
Script 1: Checking for e-mail validity ... 173
Script 2: Inspect the Trash ... 175
Summary ... 178

Chapter 9: Conditionals and Loops ... 179

What Are Conditional Tests and Loops? ... 179
Conditional Tests ... 180
The if test ... 180
The two-value if test ... 181
The multi-value if test ... 182
Repeating Loops ... 185
Repeating forever ... 185
Repeating a set number of times ... 186
Repeating with defined start and stop values ... 187
Repeating until something is true ... 189
Repeating while something is true ... 190
Repeating with a list ... 191
Writing Scripts ... 193
Script 1: Check if folder exists ... 193
Script 2: Export e-mail addresses ... 194
Summary ... 195

Chapter 10: Handling User Input ... 197

What Is User Input? ... 197
Sending a Message to the User ... 198
The beep command ... 198
The say command ... 199
The display dialog command ... 200
Asking for Text Input ... 203
Working with Buttons ... 204
Choosing Folders and Files ... 206
Choosing from a List ... 208
Choosing an Application ... 210
Summary ... 211

Chapter 11: AppleScript Subroutines 213

What Are Subroutines? 213
Defining a Subroutine 215
Running a Subroutine 216
Using Loops, Conditionals, and Variables in Subroutines..... 218
Recursive Subroutines 218
Reusing Subroutines..... 219
Some Useful Subroutines 221
Subroutine 1: Create a folder 221
Subroutine 2: Move an item to the Trash 223
Subroutine 3: Count items in a folder 225
Summary..... 226

Chapter 12: Applets and Droplets 227

What Are Applets? 227
Three Applets..... 228
Open a Web page in Safari..... 228
Run a shell script 231
Get file size 233
What Are Droplets?..... 234
Two Droplets 234
Get file size 234
Set the desktop background..... 237
Summary..... 239

Chapter 13: Folder Actions 241

What Are Folder Actions? 241
Enabling Folder Actions..... 242
Folder Action Scripts..... 244
Creating Your First Folder Action Script 246
Creating Folder Action Plug-ins with Automator 250
Summary..... 254

Chapter 14: AppleScript Studio 255

What Is AppleScript Studio? 255
How Do You Access AppleScript Studio? 256
What Can You Do with AppleScript Studio? 258

Pros and Cons 259
Your First AppleScript Studio Project 259
Summary 267

Part III: Automation Projects 269

Chapter 15: Ten Automation Projects for Files and Folders 271

The Projects 271
Creating a Basic Workflow to Process Specific Files 273
Using Automator 273
Using AppleScript 275
Advanced topics 275
Converting a Basic Workflow to Accept any Files 276
Using Automator 276
Using AppleScript 277
Advanced topics 277
Finding Files and Folders and Renaming Them 280
Using Automator 280
Using AppleScript 282
Finding Files and Folders and Trashing Them 284
Using Automator 284
Using AppleScript 285
Advanced topics 286
Creating Aliases for Files and Folders 286
Using Automator 287
Using AppleScript 288
Advanced topics 288
Filtering Finder Items 289
Using Automator 289
Using AppleScript 290
Advanced topics 291
Connecting to a Server 293
Using Automator 293
Using AppleScript 294
Getting Folder Contents 294
Using Automator 294
Using AppleScript 295
Advanced topics 296

Opening Files with the Proper Application 297
Using Automator 297
Using AppleScript 297
Advanced topics 298
Setting Spotlight Comments for Files and Folders 299
Using Automator 299
Using AppleScript 301
Advanced topics 302
Summary 303

Chapter 16: Ten Automation Projects for Music and Audio Files . . .305

The Projects 305
Playing a Specific iTunes Song 306
Using Automator 306
Using AppleScript 309
Advanced topics 310
Adding Songs to a Playlist 312
Using Automator 313
Using AppleScript 314
Advanced topics 315
Filtering iTunes Songs 316
Using Automator 316
Advanced topics 317
Setting iTunes Volume 318
Using Automator 319
Using AppleScript 319
Advanced topics 320
Pausing and Playing iTunes 321
Using Automator 322
Using AppleScript 322
Advanced topics 323
Setting Information on iTunes Songs 323
Using Automator 323
Using AppleScript 324
Removing Empty Playlists 327
Changing Case of Song Names 328
Using Automator 328
Advanced topics 329

Converting Text to Audio Files 330
Using Automator 330
Using AppleScript 331
Advanced topics 332
Adding Audio Files to an iPod 332
Using Automator 333
Using AppleScript 333
Advanced topics 333
Summary 334

Chapter 17: Ten Automation Projects for Photos and Images 335

The Projects 335
Applying Color Changes to Groups of Images 336
Using Automator 336
Using AppleScript 339
Advanced topics 342
Cropping and Resizing Images 342
Using Automator 342
Using AppleScript 344
Advanced topics 347
Creating Thumbnails 349
Using Automator 349
Advanced topics 350
Converting Images 352
Using Automator 352
Using AppleScript 352
Advanced topics 353
Flipping and Rotating Images 354
Using Automator 354
Using AppleScript 355
Advanced topics 357
Finding Specific Images 359
Using Automator 359
Advanced topics 361
Importing Images to iPhoto 361
Using Automator 361
Advanced topics 362

Exporting Images from iPhoto 364
Reviewing Photos in a PDF Contact Sheet 365
Using Automator 366
Advanced topics 367
Automating Taking Pictures with a Digital Camera 368
Using Automator 368
Using AppleScript 369
Summary 369

Chapter 18: Ten Automation Projects for Text Files 371

The Projects 371
Opening Text Files 373
Using Automator 373
Using AppleScript 375
Advanced topics 375
Asking for Text from the User 376
Using Automator 376
Using AppleScript 377
Advanced topics 378
Getting a Specific Word 379
Using AppleScript 379
Advanced topics 381
Getting a Specific Character 382
Using AppleScript 382
Advanced topics 383
Getting a Specific Paragraph 383
Using Automator 383
Using AppleScript 384
Combining Text Files 387
Using Automator 387
Using AppleScript 388
Advanced topics 389
Getting the Definition of a Word 390
Using Automator 390
Using AppleScript 391
Advanced topics 392
Using BBEdit: Working with Quotes 393
Using Automator 394
Using AppleScript 395

Using BBEdit: Convert Spaces to Tabs 395
Using Automator 395
Using AppleScript 396
Using BBEdit: Zapping Gremlins 396
Using Automator 396
Using AppleScript 397
Summary 398

Chapter 19: Ten Custom Automation Projects 399

The Projects 399
Finding Specific Contacts in Address Book 400
Using Automator 400
Using AppleScript 401
Advanced topics 402
Finding People with Birthdays 403
Using Automator 403
Advanced topics 404
Creating a Group Mailer 406
Using Automator 406
Advanced topics 406
Finding Specific Calendar Items 409
Using Automator 409
Using AppleScript 410
Advanced topics 410
Getting New Mail Messages 410
Using Automator 410
Using AppleScript 411
Advanced topics 412
Combining Mail Messages 413
Using Automator 413
Advanced topics 415
Adding Attachments to Messages 416
Extracting Text from PDFs 419
Using Automator 419
Advanced topics 421
Extracting PDF Pages 422
Downloading Specific URLs 423
Summary 425

Part IV: Appendixes 427

Appendix A: Automator Resources 429

Web-Based Tutorials ... 429
VTC Mac OS X Automator tutorials ... 429
Automator.us ... 430
Online Community ... 432
Automator World ... 432
Automator-dev mailing list ... 434
Tools and Downloads ... 434
Photoshop Action Pack ... 434
OttoMate ... 435
Automated Workflows ... 436
Training ... 437
Automated Workflows ... 437
TECSoft ... 438
Summary ... 438

Appendix B: AppleScript Resources 439

Web-Based Tutorials ... 439
MacResearch AppleScript tutorials ... 439
MacScripter ... 441
MacTech ... 442
Video tutorials on the Web ... 444
Online Community ... 445
MacScripter ... 445
Tools and Downloads ... 446
Doug's AppleScripts for iTunes ... 446
AppleScript downloads on CNET ... 447
Training ... 448
Summary ... 449

Appendix C: AppleScript Reference 451

Class Reference ... 452
Alias ... 452
Properties ... 452
Coercions ... 453
Example ... 453

Application 453
Properties 453
Coercions 454
Example 454
Boolean 454
Properties 454
Operators 454
Coercions 454
Example 455
Class 455
Properties 455
Operators 456
Coercions 456
Example 456
Constant 456
Properties 456
Operators 456
Coercions 456
Example 456
Date 457
Properties 457
Operators 457
Coercions 457
Examples 458
File 458
Coercions 458
Example 458
Integer 459
Properties 459
Operators 459
Coercions 459
Examples 459
List 459
Properties 459
Coercions 460
Examples 460

Number 460
Properties 461
Operators 461
Coercions 461
Examples 461
POSIX file 461
Properties 461
Coercions 461
Example 461
Real 462
Properties 462
Operators 463
Coercions 463
Examples 463
Record 463
Properties 464
Operators 464
Coercions 465
Example 465
Script 465
Properties 465
Coercions 465
Example 465
Text 465
Properties 465
Elements 466
Coercions 466
Examples 466
Commands Reference 466
Operators Reference 469
Logical operators 470
Logical conjunction (and) 470
Logical disjunction (or) 470
Negation (not) 470
Equality 470
Inequality 470
Greater than 471
Less than 471

Greater than or equal to 471
Less than or equal to 471
Mathematical operators 472
Multiplication (*) 472
Addition (+) 472
Subtraction (-) 472
Division (/) 472
Integral division (div) 472
Remainder (mod) 472
Exponentiation (^) 473
Other operators 473
Concatenation 473
Containment 473
Control Statements Reference 474
Considering and ignoring 474
If 474
Repeat 475
Tell 477
Try 478
Handler Reference 478
AppleScript Reserved Keywords 480
AppleScript Error Numbers 483
Summary 484

Appendix D: Automator Actions and Variables 485

Actions 485
Calendar 486
Contacts 487
Files & Folders 488
Fonts 490
Internet 492
Mail 494
Movies 495
Music 497
PDFs 499
Photos 501
Text 503
Utilities 505
Most Used 506

Variables 507
Date & Time 507
Locations 508
System 509
Text & Data 510
User 511
Utilities 511
Summary 512

Preface

This book was a lot of fun to write. I've been playing around with AppleScript and Automator for years, and with other scripting languages (notably UNIX shell and Perl) for about a decade before that. I'm positively in love with the subject of Getting Things Done, and not just in a I-need-to-knock-out-this-checklist kind of way.

The world we live in is a wonderful place. Never before have we been so connected to human events, to the brands we buy, to the people we love, to our colleagues. The advent of the World Wide Web and social media tools has added a huge amount of data influx to each of our lives—that on top of all the mainstream media, books, music, film, and other infostuff we consume.

At the same time, computing power has skyrocketed to unthinkable levels. When my father worked for UNIVAC in the 1960s, the story goes, they carted 4KB of memory around in surplus torpedo casings. When you hear stories like that, it makes you wonder if he was just pulling my leg. However, think about this—the iPhone in your pocket likely has more computing power in it than all the computers used by the Apollo 11 astronauts to land on the moon 40 years ago this summer.

So there you have it—a huge influx of tasks on one side of the ledger, and raw computing power on the other. What's needed is an easy, intuitive way to process all the stuff in our lives faster and more efficiently.

Enter Automator and AppleScript, two fantastic automation technologies built right in to Mac OS X. My sincere hope is that you will come to love these two powerful tools as much as I do, and that you'll use them to not only create innovative solutions to the ever growing tasks on your list, but also to free up more time to do other more productive tasks—like being with family, taking trips, exercising more, or enjoying life.

If any of you have any questions, don't hesitate to reach out to me. The easiest way to find me is on Twitter. My handle there is @myerman. It may take me a little bit of time to answer you, but I do my best to stay in touch.

Acknowledgments

A book is never solely the work of a lone author. In order to get everything done, the author needs a team of dedicated professionals. This book is no different, as without the following people, there's no way I could have finished:

Neil Salkind, my agent, introduced me to the good folks at Wiley. He's a mentor, guide, and coach. Thanks for everything you do.

Stephanie McComb at Wiley helped me flush out the original idea and guided the book through the acquisitions process. She's scary smart, savvy, and disarms you with a wonderful sense of humor.

Chris Wolfgang was my patient, long-suffering editor on this project. Everything good here had her involvement. Anything bad is pretty much my fault. Also, she kept my whining and moaning to a strict minimum with an even hand (and a few slaps to the back of my head).

Dennis Cohen, the technical editor, knows enough about all this to write 5 books on Automator and AppleScript. His advice and enthusiasm throughout kept me going. Without him, I wouldn't have gotten through the upgrade from Leopard to Snow Leopard, which happened just as I was finishing the original manuscript.

My wife Hope gave me room and time to write, keeping my chores to a minimum. She exacted payment at the end of the writing process though, all of which involved tons of yard work, and all of which I richly deserved.

My two lovely pups, Marlowe and Kafka, could frequently be found sleeping on one of the many dog beds in my home office. If you've never had the pleasure of coding to the cadence of snoring dogs, then you're really missing out on the finer things in life.

Introduction

Welcome to the first Automator book in the history of the Bible series. Currently, as I sit here and write this introduction, there isn't another book out on the market that covers both Automator and AppleScript for the Snow Leopard (Mac OS X 10.6) release of Apple's vaunted operating system.

One of the things that makes writing a book on automation so interesting is the time of its release. Currently, we're deep in the doldrums of what some historians and economists call the worst recession in living memory. Every night on the news, the headlines are all about housing and mortgage crises, mass layoffs, soaring federal deficits, car manufacturers in Chapter 11, and all kinds of other bad news.

It's easy to fall into despair. Millions of us are becoming entrepreneurs, small business owners, and freelancers. Most of them didn't have much of a choice — one day folks are gainfully employed, and suddenly there's a layoff. Established small businesses are also taking their lumps, having to find any way they can to compete. If they can find any way to cut corners, become more efficient, last a little bit longer, do more with less, and all those other clichés, then that's a good thing.

All of that was on my mind when I wrote this book. Yes, it's certainly important to get your hands dirty when it comes to topics like Automator and AppleScript, but all the how-to in the world is worthless without the why-to and should-you that provides the context.

It's never been easier to leverage the power of automation to solve all kinds of problems. The world isn't going to slow down any time soon; tomorrow you'll have more data to sift through then you did yesterday; in the next year of your life, you'll deal with more information than your ancestors did in their whole lives. In many ways, those who are able to cut through all the dross and get to the good stuff, the important stuff, will be those who get ahead.

Who the Book Is For

If you're drowning in boring, manual, repetitive, error-prone tasks, then this book is for you. This covers a whole bunch of territory, but it's easy to imagine you out there, cracking this book open:

- You're the freelance photographer, having to perform many discreet operations (crop, resize, color synch) on thousands of images every week.
- You're the newly hired, entry-level designer or illustrator at a big advertising agency who needs a way to stay ahead of the thousands of stock photographs you have access to for upcoming projects.

- You're the freelance programmer, having to convert data from one format to another, over and over again.
- You're the system administrator who needs to set up network-based processes such as mounting remote drives and transferring files to them at certain times.
- You're the time-starved event planner who needs a better way to send out e-mails with attachments.
- You're the small business owner who needs a way to manage all the emails, RSS feeds, and other bits of data that constantly flow in your direction.

Regardless of who you really are, it's likely that you find most aspects of your job boring and unfulfilling. It's no fun just sitting at your desk and mashing buttons in an unending cycle of boredom. That kind of environment breeds mistakes, frustration, and a certain dislike for the person who gave you the unhappy task to begin with.

This book, then, is about transforming your relationship with drudgery. It's about giving you some knowledge and some tools, followed by some practical exercises. The rest is up to you, but even if you only do a few things described in this Bible, automation will change the way you work and live.

How? Well, think about the next big task that awaits you. Instead of spending three or four days (or "just" hours) plowing through piles of undifferentiated photos, data files, or what have you, what if you could spend an hour creating an automated solution, followed by a five-minute run of that script or workflow?

Think about it. Even if your first run only does half of what you need to be done and you have to do the rest by hand, you're still way ahead of the game. Consider now what you could do with all the time you've freed up (not just now, but every time you have to perform this task)? Instead of being a drone, you could turn your attention to tasks that actually require intelligence, intuition, and judgment — all those things that humans are so good at.

How the Book Is Organized

This book is organized into four main parts.

- **Part I:** The Fundamentals of Automation
- **Part II:** A Detailed Look at AppleScript
- **Part III:** Automation Projects
- **Part IV:** Appendixes

In Part I, you'll find chapters that start you off right by letting you know in detail exactly what automation is, as well as what automation options are available with Snow Leopard. I'll walk you through the basics of Automator before I plunge into some more advanced topics. You'll learn how to create workflows, combine actions, and save your work.

Part II picks up with the basics of AppleScript. This Part has a few more chapters than Part I in order to give you more opportunity to learn about objects, dictionaries, variables, expressions, conditionals, loops, and subroutines. Also included are discussions of applets, droplets, handling user inputs, folder actions, AppleScript Studio, and of course, working directly with the AppleScript Editor.

Part III contains five chapters, each of which contains ten automation projects. All of the projects involve some work with Automator, and in many cases, those examples are padded with advanced discussions: how to do the same workflow in AppleScript, for example, or how to add more user-friendly enhancements to make the workflow more general.

The Appendixes contain Automator and AppleScript resources (so you can continue your learning), an AppleScript command reference, and a list of standard Automator actions in Snow Leopard.

QUICK START

QS

Dive into Automator and AppleScript

This book is about making your life easier, at least the part that intersects with your MacBook, MacBook Pro, iMac, or any other kind of computer running Mac OS X 10.6 Snow Leopard. In these pages you can learn how to automate various computing tasks, making yourself more productive and, I hope, happier.

Along the way, you'll get answers to questions like these:

- What is automation?
- Why care about automation?
- What is Automator, and how do you use it?
- What is AppleScript, and how do you use it?
- How do you use these two technologies together?

No assumptions are made in these pages that you know any of this, so I start from scratch and move ahead. You should find the book's organization intuitive enough that you can skip the parts you're already familiar with.

This chapter, however, is a bit different. It's a quick start, a microcosm if you will, of the entire book. You're probably reading it in the comfort of a book store, or through the Amazon Look Inside feature, trying to decide whether to shell out your hard-earned money for the whole book. A lot of the material in this chapter is covered later in much greater detail, but for now, a brief tour is what you're after, and that's what you'll get.

Without further ado, follow me into the world of automation in Mac OS X.

IN THIS CHAPTER

Why is automation important?

Who uses automation?

Introducing Automator

Introducing AppleScript

Why Is Automation Important?

It's ironic to consider that computers were invented to make our lives easier. They are able to reliably calculate impossible sums, allow us to take on tasks (like page layout, Web design, and video editing) that only the professionals could do before, and store all kinds of data, ranging from mundane receipts all the way to personal collections of photos, songs, and even that novel you've been working on for ten years.

For most of us, computers have brought a new way of working and interacting with others, which means that the whole nature of our society has changed dramatically. A hard day at work for my grandfather meant busting his back in the fields; for my father, it was working in a pre-computer age office; for me, it means manipulating documents and data. A lot of documents and data.

With a lot of anything come a lot of repetitive tasks. Consider the task of renaming one file. Simple, right? All you have to do is open the Finder, browse to the appropriate folder, click the name of the file, watch it become editable (see Figure QS.1), and then type in a new name for the file.

FIGURE QS.1

Manually changing a filename in Mac OS X

At some point, though, you may encounter a more onerous task: renaming 100 files. It's not really that difficult — it doesn't tax your mental facilities — but it is tedious. If you're not careful, you might not get all of them right, or you might miss a few, or you might get a call from someone and then come back to the task later, not really sure where you left off.

Given this trivial example, I come to the heart of the question: why is automation important? Automation is important because it:

- speeds up otherwise tedious, time-consuming tasks;
- removes the possibility of errors while doing those tasks; and
- gives you more time and headspace for tasks that invite your full concentration and skills.

In this book, you can learn how to use two tools for automation in Mac OS X: Automator and AppleScript. The first is an intuitive, wizard-like tool that allows you to put together visual workflows. You'll find that it's useful in about 50 percent of situations where you need to automate a task. The second tool is a scripting language used for many different types of tasks. It does a rather good job of allowing you to customize any automation task and also gives you much finer control over your Mac.

Who Uses Automation?

You might think that the primary users of automation technologies are big businesses. In a sense, you're right. The world's largest organizations spend hundreds of millions (if not billions) of dollars each year trying to become more productive. They hire business analysts to figure out how different workers really do their jobs; then they make decisions based on that analysis to purchase tools, software, and training to make those workers more productive.

So it's not surprising that they spend a great deal of money automating all areas of their business. They have warehouse management and fleet management software to keep distribution flowing. They deploy storage devices and the backup software to run nightly backups. They install customer relationship management (CRM) systems and sales force automation (SFA) tools to facilitate the process of turning strangers into paying customers and then keeping those customers happy.

Automation isn't limited to the big guys, though. Small businesses use automation whenever they can — think about the investment the small retailer makes in a point-of-sale system. These systems help store owners keep track of sales, customer accounts, and much, much more. They can be integrated into other systems, like inventory management, to make reordering a lot more streamlined.

Think of the small design agency or software company. They need automated backups of files, as well as tools that help them put the finishing touches on production-ready products. On the design side, this may be a series of filters that convert masses of images into different formats. On the software development side, there are tools that automate builds and aid in the process of finding bugs.

So what about you? You may be a home-based business owner, a consultant, a lone-gun freelancer, or just a Mac owner with 10,000 photos to process. The same rules and processes that make automation a good deal for the big guys also apply to you.

Now take a look at Automator and AppleScript for a working example of what I've been talking about.

Introducing Automator

Automator is a workflow creation tool that features an easy-to-use interface. It's perfect for the non-programmer, because it allows users to piece together workflows using actions and other automation concepts. You'll be introduced to workflows, actions, and other terms here in this Quick Start chapter; you'll see those terms again later in the book, when the chapters have more space for a longer look.

Before I get started, let me introduce you to the user interface components you'll encounter the first time you open Automator (see Figure QS.2).

FIGURE QS.2

The Automator user interface

Along the top is Automator's Toolbar. The Toolbar gives you quick access to common functions. For example, on the left side you can click the Media button to access any photos, audio files, and other media that you might have stored on your Mac (see Figure QS.3).

On the right side of the Toolbar, you can record manual tasks with the Record button (this is in case the existing actions aren't what you need), and play and stop any workflows you may have built.

Cross-Ref

You can learn more about the handy Record feature in Chapter 4.

The rest of the interface is broken up into two halves. On the left side is the Library. The Library contains a list of available actions and variables that you can use in your workflows. This pane is split into two columns — on the left is a list of content types (each with its own associated application, like Mail or the Finder, and so on) followed by a list of categories (such as Recently Used and Most Relevant).

FIGURE QS.3

Clicking Media on Automator's Toolbar reveals media files.

The right pane is where the workflows that you build reside. A workflow is just a series of steps to automate a process. For example, if you wanted to rename a bunch of files, you'd probably follow this process manually:

1. **Select the files that you want to work with.**
2. **Copy those files (in case you make any mistakes) into a new folder, probably on your desktop.**
3. **Rename the files.**

In Figures QS.2, you can see that the workflow on the right side follows this thinking. To automate, all you have to do is transcribe your manual process into a visual Automator workflow.

Before you go any farther along the actual Automator trail, I'll talk a little more about actions and variables, as they are key to understanding how workflows work.

Actions

Most Automator *actions* are designed to do a specific task: copy a file, open a URL, create an archive, and so on. In some cases, an action might do everything you need, but most of the time, you'll be adding actions together into a linked, multipart, automated process known as a *workflow*.

In a workflow, an individual action can receive input (or information) from a previous action and then pass along some output (results) to the next action in line. For example, if you're going to build a backup workflow for your documents, your actions might look like this:

1. **Process a list of documents.**
2. **Pass the list on to the next action, the archive action.**
3. **Create a ZIP file using the archive action.**
4. **Pass the ZIP file on to the final action, backing up your documents.**
5. **Open an FTP connection to a remote server, or mount an attached drive (either of these could be the backup action).**

In Figure QS.4, that's precisely what's happening. First, specify which documents to select in the Home directory, then archive them, and then use a Fetch FTP Upload action to upload the archive to a server. I'm using the Fetch FTP Upload action to make a point: Certain applications (like Fetch) make certain extra actions available to Automator. If you don't have Fetch installed, then you won't have this action available to you.

Note

If you install applications with Automator support, or download actions from www.apple.com/downloads/macosx/automator, they will be added to your available list of actions in Automator.

FIGURE QS.4

Backing up documents with Automator

This little example brings up a few points about actions:

- **Your basic workflows almost always specify certain files to start with.** As you get more advanced, you'll learn how to ask the user for their input on which files or items to process.
- **Actions are usually created by developers and made available to you.** The makers of the FTP client Fetch have made a variety of actions available to folks who use their software. I've taken advantage of that generosity here in this sample workflow.
- **Notice on the workflow how the actions are connected.** Normally, you see one action flowing into another by way of a downward-pointing triangle superimposed on a bubble. Later on, as you troubleshoot, the triangle is a good thing to look for. If you don't see the triangle (representing output) and bubble (representing an input channel), then your actions probably aren't connected.
- **Actions may take in one kind of input and provide a different kind of output.** You might start by giving a workflow action a list of URLs, for example, but then end up with a list of images found on those URLs. You have to make sure that an action provides an output that another action can accept as input.

Furthermore, actions have options associated with them, but I'll get into all that when I build a sample workflow later in this Quick Start chapter. For now, it's important to understand that an action is the building block of a workflow and that each action accepts input and generates output. By chaining your actions in different combinations, you can build extremely complex workflows.

Variables

Starting with Mac OS X 10.5 (Leopard), Automator added a handy feature known as a *variable*. If you're a programmer, you know that a variable can hold or store some information, usually a number, name, date, or other string. In Automator, a variable is just that: a placeholder for some kind of information that you might use later on in a workflow. Suppose you want to create a workflow that downloads the text of a Web page to a text file on your computer. Normally, you would start your workflow with a New Folder action, which prompts you for a folder name and location. Then you'd add the other actions that would open the current Web page in Safari, grab the text, and save it to the folder you specified.

What you're going to add is a Set Value of Variable action in between the first two actions. This stores the name of the folder into a variable. You can then use that variable in the final New Text File action. All you have to do to use your new variable is to drag it from the Variable list to the pop-up menu.

Tip

If you can't see the Variable list, simply click the icon with two blue bars at the bottom of the Automator window. In the following sequence, the path to the Web Content folder (which lives on the desktop) is saved in the variable *myfolder* (see Figure QS.5). You can reuse this variable later in the workflow.

Admittedly, this is a very simple example, but simple or not, variables can be very handy in making your workflows more flexible.

One more note about variables: Automator has a number of built-in variables that you can use. To see them, simply click the Variables button on the left pane. You'll see variables relating to the following:

- Date & Time (such as today's date and current time, both of which are handy if you're doing backups).
- Locations (such as your Downloads folder, your Home folder, and so on).
- System (such as your computer's name, uptime, host name, IP address, and so on).
- Text & Data (for specifying new variables and data).
- User (such as a user's first name, e-mail address, and other personalized information).
- Utilities (such as a random number or new shell script).

FIGURE QS.5

Using a variable to customize workflows

Creating your first basic workflow

Now that you have some introductory concepts out of the way, you can learn to create a very basic workflow from start to finish. You've already seen some examples here of different workflows, but this time you can focus not only on building a workflow, but the entire process behind a successful workflow.

This first workflow is pretty basic, and is based on some client tasks that I recently automated. The client had a folder full of eBooks in PDF format. What they wanted was some way to look for certain keywords within each PDF and then to label or tag those PDFs as having those keywords if found.

This is a perfect workflow to start with; everyone knows just how tedious it is to go through a bunch of PDFs and try to tag them in some way. The way this workflow is constructed, you can also reuse it in a variety of other contexts.

The first step is to figure out how to do the search. If you've been using Mac OS X for any length of time, you probably know about Spotlight, the built-in search feature. Lucky for you, there's also a Spotlight action that you can use in your workflows.

So this is what you're going to do:

1. **Add the Spotlight action.**
2. **Limit the search to a certain folder (in this case, the client had all his eBooks in a folder named *marketingtips*).**
3. **Run a fixed search for a meaningful phrase like** social media**, as that's a good place to start.**

In Figure QS.6, you can see that I've clicked Results under the Spotlight action. When the workflow runs, I can see exactly what comes back from my query.

FIGURE QS.6

The Spotlight action allows you to search most types of files.

The Results button is a handy way to debug your workflows, by the way. You can see your results in Icon, Table, or List view. In this example, the Icon view (shown in Figure QS.6) obviously displays a list of thumbnails. The Table view shows a list of paths formatted as a table (see Figure QS.7), and the List view shows the list of files as an AppleScript array (see Figure QS.8).

The point is that you can view these results in different formats. Not all result sets can display as icons, and in some cases, the List view won't be that useful to you unless you know AppleScript (which you will by the time you finish this book!).

FIGURE QS.7

The same result set of PDFs in Table view

FIGURE QS.8

The same result set of PDFs in List view

Now that you've found some documents that actually contain the keyword or phrase you're looking for, it's time to do something useful with those documents. What you want to do is label those

documents using the Label feature in the Finder. If you're not familiar with labels, they're just colors you can attach to a file in the Finder view (see Figure QS.9).

FIGURE QS.9

Adding a label to the Finder items

When this workflow runs, you end up with a blue label on all the PDFs that have the phrase **social media** in them, as illustrated in Figure QS.10.

FIGURE QS.10

The result of adding labels

This is immediately useful, of course, but isn't very interesting. I also decided to add some metadata to the Spotlight Comments for each of these files; that way, the client could keep better track of them in the future. Why use Spotlight Comments? Well, it's a good way to tag your files so that you can do something useful with them later — like create Smart Folders.

So I added a Set Spotlight Comments for Finder Items action in between the two actions already there, and made sure that I typed in **social media**. I also checked the box next to Append to existing comments (see Figure QS.11).

FIGURE QS.11

Appending Spotlight Comments with Automator

Once this workflow runs, you can Ctrl+click (or right-click if you have a two-button mouse) on any of these PDFs and select Get Info. You can see that you now have **social media** inserted into the Spotlight Comments in Figure QS.12.

FIGURE QS.12

Comments added to Spotlight Comments

Customizing your first workflow

So far, you've built a workflow that performs like this:

1. **It takes a hard-coded search phrase** (social media).
2. **It finds documents that match that search phrase in a single folder.**
3. **It tags those files with that phrase in the Spotlight Comments.**

Immediately useful, yes, but it's hardly extensible. What happens if you need to search other files for other phrases, and then apply other labels to them?

To begin customizing this workflow into a more flexible tool, follow these steps:

1. **Delete the phrase** social media **from the Spotlight action.**
2. **Click Options along the bottom of the action.**
3. **Check the Show this action when the workflow runs checkbox.** This causes the workflow to pause while you enter your own search string. Now when you run the workflow, Automator pauses and displays a dialog. The user can now enter a search phrase and even change the location of their search (see Figure QS.13).

FIGURE QS.13

Causing the workflow to prompt for user input

4. **Don't click Continue yet, because the search phrase is still hard-coded in the workflow.**
5. **Click Cancel (see Figure QS.14) to go back into Edit mode.** What you want is to allow the user to enter whatever comments they feel like entering during the workflow.

FIGURE QS.14

Now Spotlight prompts for a search string.

6. **Delete the hard-coded string** social media **from the action entry box.**
7. **Click Options.**
8. **Check the Show this action when the workflow runs checkbox (see Figure QS.15).** One more bit of customization, and you'll be ready to run your new workflow. Instead of using the same label color, you can allow the user to choose their own label color.

FIGURE QS.15

Allowing the user to enter their own Spotlight Comments

9. **Click Options (located under Label Finder Items).**
10. **Check the Show this action when the workflow runs checkbox (see Figure QS.16).**

FIGURE QS.16

Allowing the user to pick a label color

The result is a workflow with more interruptions (three, in fact) but with the results you want. As you can see from Figure QS.17, I've run a second search on **email marketing** and tagged a number of PDFs with the **email marketing** comment and a yellow label. Because files may exist that contain both **social media** and **email marketing** as phrases, the labels become less useful, but the comments are still appended.

Now that some files have Spotlight Comments, you could open a Finder window, select File➪ New Smart Folder from the menu, limit the search to your user's home area, and click the + sign to

add another criterion for searching and filtering. Once there, click Other and select Spotlight Comments from the list; then add **social media** to your match criteria. The result is a list of files in a new Smart Folder (see Figure QS.18).

You can then save this Smart Folder and add it to your Saved Searches, which puts it in the sidebar of your Finder window. If you add more **social media** Spotlight Comments to other files, they automatically show up in the list. You've just made the job of finding certain files a lot easier, and you didn't have to pick your way through hundreds of files in order to find what you were looking for.

FIGURE QS.17

Labels and Spotlight Comments at work

FIGURE QS.18

Using Spotlight Comments to create a Smart Folder

Options for saving your workflow

For now, you're going to save your workflow as a workflow. A good suggestion is to create a Workflows folder in your home directory; that way, they stay organized. You can then load the workflow into Automator and run it again.

There are other options for saving your work, like a stand-alone application or plugin, but I'll talk about those later. For now, just know that an application can be invoked independently, while plugins are added to other contexts — for example, to the right-click menu or to the Print options menu.

Being able to save your workflows in different formats gives you a great deal of flexibility, putting your automation solutions closer to where you need them.

Introducing AppleScript

AppleScript is a scripting language that comes with your Mac. In fact, it's been available since the days of Mac OS 7, making it a venerable automation tool. So what is AppleScript? Put simply, it is a language you can use to control your operating system, files, and applications using a set of common instructions.

If you're new to programming, don't worry; you'll find AppleScript a pretty easy first language to learn. It uses some pretty intuitive structures and English-like names for operations. For example, you'll find yourself using the `tell` command a lot – `tell application "TextEdit "` is simply the way you open a text editor from AppleScript.

If you're an experienced Objective-C or Cocoa coder, you won't have that much trouble with AppleScript, as you'll already have your head wrapped around some of the same concepts. If you're a UNIX shell scripter, a Perl programmer, or a PHP/Ruby Web developer (or even a serious Java coder), then you might have a little transition time. Of all those languages, AppleScript is probably closest (at least in function) to shell scripting, but the way it's put together is very different from what you may be used to.

At its simplest, an AppleScript opens an application, tells it to do certain tasks, and then releases that application. In order for you to understand what that fully means, though, you have to realize the following:

- **Mac OS X may have a different idea than you of what an application is.** For example, many developers who are new to the Mac don't realize that the Finder is an application. If you're in doubt, take a look at the list of applications in your Applications folder.
- **Not all applications are equally scriptable.** Some applications may come with broad support for AppleScript (by way of robust dictionaries; more on this later), while others may have no support or offer only a limited amount of support.

Because you can find a lot more detail about AppleScript in Part III of this book, let's just jump in so that you can start to become familiar with it.

When you're creating an AppleScript, you'll be using the Script Editor, which is available under Applications➪Utilities➪AppleScript Editor. The Script Editor is a lightweight environment in which you can create, edit, compile, debug, and run your AppleScripts (see Figure QS.19).

You enter your script code in the top pane, run it using the Run button on the toolbar, and see results and other data appear in the bottom pane. If the code opens Finder windows or applications like Mail or Address Book, then those events transpire as well.

FIGURE QS.19

The Script Editor

You can create a very basic script and then add to it as you go along. For this first foray, write a script that opens your user's Documents folder in the Finder. To do that in AppleScript, you would use the following commands:

```
tell application "Finder"
    open "myerman:Users:myerman:Documents"
end tell
```

Figure QS.20 shows what that would look like in the Script Editor.

Note

You will need to fine-tune this script on your computer. You may have Macintosh HD as your root, or you may be using some other name. You'll most certainly have a different username.

By the way, another way of running this script in a more universally accepted way is:

```
tell application "Finder"
    open (path to documents folder)
end tell
```

All you have to do now is click Run (the green button with the arrow pointing right) in the Toolbar or press ⌘+R on the keyboard, and you'll see your Mac open the directory specified in the script, as shown in Figure QS.21.

FIGURE QS.20

The Script Editor displaying basic script

FIGURE QS.21

A Finder window opens using AppleScript.

Add another line to the script. What if you wanted to open a certain Web page as well as a Finder window? You can do that with an `open location` command.

```
tell application "Finder"
    open "myerman:Users:myerman:Documents"
open location "http://www.tripledogs.com"
end tell
```

When you run this command, you not only get a Finder window, but a new Safari window (if Safari is your default browser!) with the URL also loads, as shown in Figure QS.22.

FIGURE QS.22

Loading a Finder window and Safari using AppleScript

As you can see, you can open a lot of things with the Finder application, including applications and documents. Just keep adding them to the list in the Script Editor.

```
tell application "Finder"
    open "myerman:Users:myerman:Documents"
    open location "http://www.tripledogs.com"
    open "myerman:Applications:Dictionary.app"
    open "myerman:Users:myerman:Documents:TDDM:tddm_logo.pdf"
end tell
```

In Figure QS.23, I've asked for not only a Finder window and a Web document, but also the Dictionary application and a PDF of my company's logo.

FIGURE QS.23

Loading applications, Finder windows, URLs, and documents using AppleScript

This is all very handy, of course, as you can imagine setting this script up as something your Mac runs on start up every morning. However, at the moment all you're doing is filling up the screen with documents, Finder windows, and applications. It's automated, and it's a bit fun, but it isn't very useful.

Instead, get ready to learn more commands. A useful construct in AppleScript is `get items of` *x*, where *x* can be a folder or some other container. An *item* is just a generic object, something you might expect to find, like a file, folder, or shortcut.

If you go back to the idea of our *marketingtips* folder, the one that contained all those PDFs, you could run a simple AppleScript to get a list of all the items in that folder.

```
tell application "Finder"
    get items of folder "myerman:Users:myerman:Documents:marketingt
   ips"
end tell
```

The result is a list of files found in the directory, as illustrated in Figure QS.24. As you can see, each item (in this case, a PDF) is listed in this format: document file X of folder A of folder B of folder C of folder D of startup disk of application Finder. Each file is thus marked in reverse path order, with each one culminating at the Finder application (if you're coming from Windows or Linux, you might find this a bit odd; just run with it).

FIGURE QS.24

Listing items in a folder with AppleScript

```
{document file "2000Ideavirus.pdf" of folder "marketingtips" of folder "Documents" of folder "myerman"
of folder "Users" of startup disk of application "Finder", document file "75-secrets-of-the-masters.pdf" of
folder "marketingtips" of folder "Documents" of folder "myerman" of folder "Users" of startup disk of
application "Finder", document file "7Day_Ebook.pdf" of folder "marketingtips" of folder "Documents" of
folder "myerman" of folder "Users" of startup disk of application "Finder", document file "affiliate-
masters.pdf" of folder "marketingtips" of folder "Documents" of folder "myerman" of folder "Users" of
startup disk of application "Finder", document file "art-of-the-white-paper.pdf" of folder "marketingtips"
of folder "Documents" of folder "myerman" of folder "Users" of startup disk of application "Finder",
document file "blog-profits-blueprint.pdf" of folder "marketingtips" of folder "Documents" of folder
"myerman" of folder "Users" of startup disk of application "Finder", document file "building-traffic-with-
article-marketing.pdf" of folder "marketingtips" of folder "Documents" of folder "myerman" of folder
"Users" of startup disk of application "Finder", document file "classified-magic-how-to-make-small-ads-
pay-off-big.pdf" of folder "marketingtips" of folder "Documents" of folder "myerman" of folder "Users" of
startup disk of application "Finder", document file "copywriting-tricks-of-the-trade-50-secrets.pdf" of
folder "marketingtips" of folder "Documents" of folder "myerman" of folder "Users" of startup disk of
application "Finder", document file "core-strategies-of-successful-copywriting-awai.pdf" of folder
"marketingtips" of folder "Documents" of folder "myerman" of folder "Users" of startup disk of
application "Finder", document file "creating-white-papers-that-drive-sales.pdf" of folder "marketingtips"
of folder "Documents" of folder "myerman" of folder "Users" of startup disk of application "Finder",
document file "design-to-sell-white-paper-sidebars.pdf" of folder "marketingtips" of folder "Documents"
of folder "myerman" of folder "Users" of startup disk of application "Finder", document file "desperate-
buyers-only.pdf" of folder "marketingtips" of folder "Documents" of folder "myerman" of folder "Users"
of startup disk of application "Finder", document file "direct-response-pr.pdf" of folder "marketingtips" of
folder "Documents" of folder "myerman" of folder "Users" of startup disk of application "Finder",
document file "effective-followup-marketing.pdf" of folder "marketingtips" of folder "Documents" of
folder "myerman" of folder "Users" of startup disk of application "Finder", document file "how-to-choose-
your-carrot-effective-lead-generation.pdf" of folder "marketingtips" of folder "Documents" of folder
```

Of course, you could be a little smarter about this and set the name of the folder in a variable. You can do this with the `set` command.

```
tell application "Finder"
    set foldername to "myerman:Users:myerman:Documents:marketingtips"
    get items of folder foldername
end tell
```

The result is the same, but now you can reuse the variable `foldername` as needed, without having to type up that entire path each time.

Change the command a bit and ask for item 2 only:

```
tell application "Finder"
    set foldername to "myerman:Users:myerman:Documents:marketingtips"
    get item 2 of folder foldername
end tell
```

The returned list in Figure QS.25 is now pretty short — in fact, it returns only the second file in the folder.

To get a sense of what you can do with AppleScript, select Window ➪ Library from the Script Editor menubar. What you see is a library of dictionaries for each application on your Mac. A dictionary in this context is simply a list of available hooks you can script. To find dictionary items available to AppleScript that relate to the Finder, double-click the Finder icon in the Library window.

When you do that, a long list of categorized items appears, as shown in Figure QS.26. Perusing the list should give you a pretty good idea of what's available to AppleScript.

FIGURE QS.25

Getting one item with AppleScript

```
document file "75-secrets-of-the-masters.pdf" of folder "marketingtips" of folder "Documents" of
folder "myerman" of folder "Users" of startup disk of application "Finder"
```

FIGURE QS.26

The Finder dictionary

For example, say you wanted to loop through the *marketingtips* folder and pull out the name and label index of each item you find, as shown in Figure QS.27. In this case, you would use the following script:

```
tell application "Finder"
    set foldername to "myerman:Users:myerman:Documents:marketingtips"
    get {name, label index} of items of folder foldername
end tell
```

The result is pretty straightforward: two lists, one containing the item name, the other containing an index label (where 0 is no label, 1 is red, and so on).

What's next? Well, if you know how to `get` a value, you can test that value with an `if` and then use `set` to change the value. For example, what if you wanted to write a script that would remove any label assigned to a list of files?

The first step in such a script would be to build on what you have. You already have a `foldername` variable that holds the path to the folder you're working with. The next step is to create a second variable, one called `myfiles`, that will hold all the items of that folder.

FIGURE QS.27

Using AppleScript to get names and label indexes in the Finder

{{"2000Ideavirus.pdf", "75-secrets-of-the-masters.pdf", "7Day_Ebook.pdf", "affiliate-masters.pdf", "art-of-the-white-paper.pdf", "blog-profits-blueprint.pdf", "building-traffic-with-article-marketing.pdf", "classified-magic-how-to-make-small-ads-pay-off-big.pdf", "copywriting-tricks-of-the-trade-50-secrets.pdf", "core-strategies-of-successful-copywriting-awai.pdf", "creating-white-papers-that-drive-sales.pdf", "design-to-sell-white-paper-sidebars.pdf", "desperate-buyers-only.pdf", "direct-response-pr.pdf", "effective-followup-marketing.pdf", "how-to-choose-your-carrot-effective-lead-generation.pdf", "how-to-do-your-own-pr.pdf", "how-to-double-response-rates-at-half-the-cost.pdf", "how-to-write-a-white-paper-a-closer-look-at-the-white-paper-definition.pdf", "how-to-write-a-white-paper-the-five-laws.pdf", "how-to-write-a-white-paper.pdf", "how-to-write-and-sell-your-own-information-products.pdf", "magic-words-that-bring-you-riches.pdf", "magnetic-sales-letters.pdf", "million-dollar-emails.pdf", "million-dollar-sales-letters.pdf", "new-rules-of-pr.pdf", "niche-hunting.pdf", "nmkh-bg.pdf", "power-of-selling-art-of-copywriting-beyond-direct-mail.pdf", "profit-pulling-ebooks.pdf", "SMARTS-social-marketing-strategy-guide.pdf", "steak-behind-the-sizzle-effective-marketing-using-white-papers.pdf", "surefire-sales-letter-secrets.pdf", "test-and-track.pdf", "the-white-paper-white-paper.pdf", "top-10-biggest-online-email-marketing-mistakes-sherpa.pdf", "ultimate-sales-letter-toolbox.pdf", "viral-copy-trading-words-for-traffic.pdf", "worlds-best-kept-copywriting-secrets.pdf"}, {1, 0, 1, 4, 1, 1, 1, 4, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 4, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1}}

Use the `repeat` command to go through every file in `myfiles`, testing to see if the `label index` of the file is greater than zero. If it is, then you know that the file has a label associated with it, so reset the `label index` on the file to zero, thereby stripping every single file of its label.

```
tell application "Finder"
  set foldername to "myerman:Users:myerman:Documents:marketingtips"
  set myfiles to items of folder foldername
  repeat with myFile in myfiles
    if label index of myFile > 0 then
      set label index of myFile to 0
    end if
  end repeat
end tell
```

Once this code runs, each file in the folder is stripped of its label, as shown in Figure QS.28.

There is one last step before you move on. Right now, this AppleScript has limited usefulness, as it has a hard-coded variable. What if you could ask the user for a folder to work with? The way to do that is with a `choose folder` command, which you will use to prompt the user to select a folder (see Figure QS.29) and then capture all the items within it as an alias list.

```
tell application "Finder"
  set myfiles to (entire contents of (choose folder) as alias list)
  repeat with myFile in myfiles
    if label index of myFile > 0 then
      set label index of myFile to 0
    end if
  end repeat
end tell
```

FIGURE QS.28

Files stripped of their labels using AppleScript

Notice also that you are setting the value of the `myfiles` variable right on the same line. This makes for a very long line of code, given all those prepositions, but it's classic AppleScript. It is a lot wordier than Unix shell, Perl, Python, and other languages you may be used to, but it gets the job done.

FIGURE QS.29

Using AppleScript to prompt the user to choose a folder

The only thing that's left is to save your work. You can save your AppleScripts wherever you like, but a good place to put them is in `~/Library/Scripts`, where you'll find a lot of other AppleScripts. For example, the label-stripping script you just wrote here would be right at home in `~/Library/Scripts/Finder Scripts`. You can save it as Strip Labels and run it by double-clicking it (or adding it to an Automator workflow, but that would be skipping ahead).

At this point, the rest of AppleScript is in the details. Granted, there are a lot of details, but you won't need to know all of them in order to become productive. The aim of this book is to make you better at automation, so any AppleScript topics tackled in this book all relate to that objective.

Summary

So far, you've taken a very quick dive into the world of Automator and AppleScript. You've learned how to put together a simple workflow, and learned how to use actions and variables. On the AppleScript side, you've learned the bare-bones basics of working with the Finder, as well as getting and setting variables, built a simple repeat loop, added a conditional test, and asked for user input.

Part I

The Fundamentals of Automation

IN THIS PART

Chapter 1
What Is Automation?

Chapter 2
Automation from a User's Perspective

Chapter 3
Automator Basics

Chapter 4
Advanced Automator Topics

CHAPTER 1

What Is Automation?

Before getting into any of the details concerning Mac OS X and automation, it's time to take a breather and get some perspective. There's plenty of time to go into all the intricacies of AppleScript, Automator, and all the ways to combine the two to automate workflows in later chapters. If this were any other book, then I could certainly get away with just a how-to manual.

But this is a Bible, and that means letting you know not just how-to but also why-should-we, as well as why-do-we. By no means is this book meant to imply that you must do this instead of that because people who do *this* are good, and people who do *that* are bad, but it is meant to be a one-stop, comprehensive look at the topic.

There is going to be just a little bit of preaching, but rest assured, it will all be confined to this section of the book. Many of you are already converted, but some still need some prodding.

After this section, I'll get back to the details, but for now, it is important to understand the whys and wherefores of automation, and that means taking a short trip through history.

IN THIS CHAPTER

- **Why you should use automation**
- **A brief history of automation**
- **Examples of automation**
- **Why automate?**

Why You Should Use Automation

I'm going to return to an important theme expressed in the Quick Start chapter: Simply put, people are good at some things, and computers are good at others. Humans have reason, intuition, and can apply their judgment, experience, and insight to the data they sense with their own eyes and ears.

Computers, on the other hand, are good at rapid calculation and data crunching; unsurpassed in the areas of storage and retrieval; and very, very good at transforming one set of data into another. What computers can't do very well (yet) are all those things that humans are good at: giving all of that data some context. Your iMac may be very good at resizing photographs, but it can't tell you which photos are the best ones you took on your last vacation.

Fortunately (and unfortunately) for us in the early twenty-first century, computers are everywhere. They're on our desktops at home and work (something that was deemed unthinkable 25 years ago). Portable laptops are finally small and light enough to carry in messenger bags. Processors are jammed into iPods, car engines, traffic control systems, industrial turbines — you name the place, and there's a little electronic brain analyzing input and spewing out data.

What all of this means is that we're faced with a geometric increase in the kinds of data we have to process. Graphic designers may have to process 1,000 images at one go, doing the same tasks over and over again: Resize to no bigger than 300x300 pixels, grayscale, rename the file, and then create a thumbnail. System administrators have to pick through 100,000 lines of log files to figure out what happened if a server is hacked. Marketing professionals are confronted by thousands of data points from all across the business — Web analytics, open rates from e-mail marketing campaigns, number of Twitter followers and retweets, network analyses of their LinkedIn and Facebook groups, and all of it compared to their sales pipeline and forecasts.

It's easy to forget that there's a purpose to all of this data: What we're meant to do with the data is to make decisions and gather insight, not just wallow in it. But there's so much data, so many files, so much to do that those of us without the proper skills in automation, well, we just sift data through a sieve all day long.

I once sat and watched as a young graphic designer, working on an iMac, went through the painful manual processing of thousands of images. It took her most of a day to perform all of her tasks, with an average processing time of two to three minutes per image. I've never seen anyone so beat from a full day of sitting at a desk in an air-conditioned office. I won't even talk about how sore her mousing hand was!

My job at the time didn't include intervention, just assessment, so all I did was report back to the agency she worked for. And she was by no means the exception to the rule. Just about every single graphic designer in that department had little or no exposure to automation, even though every single Mac came with Automator and AppleScript installed.

Simply adding a very short Automator workflow to their process would save countless person hours of image processing. When I suggested this change to management, I got the immediate response: "This will save us a lot of time and money, but what do we do with these people now that they have all this free time?"

I was astonished by the question. My response was curt and immediate: "Put them to work doing something valuable! They're all graduates of design programs. Put them to work designing great

things for your clients." Any low-level intern with minimal training can spend their entire day crunching through a series of repetitive tasks. It isn't fulfilling, it doesn't add that much value, and therefore, it needs to be automated as quickly as possible.

It's important to remember, though, that process automation is by no means an end unto itself. Some tasks cannot be fully automated, nor should they be. The point is to provide repeatable, automated workflows where they make sense. It's about leveraging the strengths of the computer, particularly in the arenas of quick calculation and task completion, such that the human has more time to do what they do best: Apply expertise, intuition, and skill.

In the case of the designer working at the agency, yes, of course, automation will mean fewer billable hours for resizing digital images, but it will also mean a lot more time left over each day for client meetings. It's the face-to-face time with clients, the back and forth, the design process that makes a designer a more effective professional, not the time spent head down with repetitive tasks.

And it's not just about making creative people more productive, either: Think about all the accountants, programmers, and other left-brainers out there. Come tax time, accountants can benefit greatly from adding automation to their work processes. And there's hardly a single programmer left on the planet who hasn't benefited from some kind of automation, whether it is in build management or some other area. The same goes for those who have to manage a lot of people and the data they produce: sales managers, casino floor bosses, storeowners, and more.

A Brief History of Automation

If you look up the word *automation* in Merriam-Webster's Collegiate Dictionary, you get something close to the following definition:

1. The technique of making an apparatus, a process, or a system operate automatically. 2. The state of being operated automatically. 3. Automatically controlled operation of an apparatus, process, or system by mechanical or electronic devices that take the place of human labor.

The term automation itself was coined in 1912, but it wasn't really popularized until after World War II. Of course, other related terms, such as automatic (as in *semi-automatic pistol* or *automatic reflex*), have been around long before the advent of computers (the former since 1898, and the latter since the late 1600s). The ancient Greeks had a word, *automaton*, that described a self-moving, self-operating apparatus (what we would call a robot); later on, this word evolved into the French *automatique* in the seventeenth century to denote a mechanism designed to follow a predetermined sequence of operations.

So, well before the computer age, we had this notion of machines or systems doing repetitive tasks. That's because anyone who sat down to analyze the kind of work people did day in and day out would recognize that properly automating a process greatly increases the speed at which the work is done and reduces the error rate.

Differences between mechanization and automation

In areas of agriculture or industrial production, mechanization is often conflated with automation, but the two aren't the same. *Mechanization* is all about giving human operators the tools and machinery to assist them with physical labor — for example, in the early days of industrialization, a steam-powered tool like a lathe dramatically decreased the time it took to turn out table legs, but the lathe still required a human operator. The same goes for water-powered mills, windmills, or foot-powered threshers. These mechanized industries make you more productive, but they don't necessarily involve automation.

Automation, on the other hand, is a step beyond mechanization and is all about reducing the need for human intervention. It's about requiring less human sensory and cognitive input into some kind of process, system, or task. An automated teller machine is a good example: Here you have a networked appliance that allows users to carry out basic transactions without the need for a human teller. The use of sophisticated lab equipment to automate the analysis of genes, cells, and tissue samples has led to many incredible breakthroughs in medical science (not to mention the launch of various blockbuster CSI franchises on network television). Autoresponder e-mail systems are yet another example of how a small business can use technology to provide better service and response times to customers.

Regardless, automation and mechanization are most linked to the Industrial Revolution. That this wasn't your typical revolution with armies and uprisings is beyond the point — it was a paradigm shift, one that started innocently enough. First you had the use of the steam engine in the 1760s to drive machinery, mostly in the textile industry, and then that turned into a whole bunch of applications across a wide swath of human work.

Of course, some historians have pointed out that you had all kinds of mechanization long before then, ranging from the aforementioned windmills to foot-powered threshers, but from a historical perspective, the Industrial Revolution wasn't just about mechanization; it was about how we organized ourselves to work. Over the past three centuries, there's been an inexorable shift from small villages and farms to urban centers, from handcrafted goods to mass-produced widgets.

Since 1946, automation has increasingly been applied to the world of information technology, data processing, and computers. Each year, the amount of data grows and grows, and so does the need for improving how we store, handle, secure, and transform (and any other verb you can think of) all of this data. Without automation, without some tools for sifting through or processing data, we would never get to the point where we could actually add value to the system: giving that data some context, turning it into knowledge, and transforming that knowledge into action.

In industrial contexts, the focus of automation has shifted over the past 20 years from mere quantity to more focus on quality output and flexibility on the line. Whereas hand-installed pistons for truck engines might involve a 1-percent to 1.5-percent error rate, automating that process has reduced errors to 0.00001 percent, according to Wikipedia.

Negative connotations of automation

Of course, you can't have a discussion of automation without mentioning the naysayers, those who, like the ad agency referenced earlier, argue that "increased automation puts people out of work." It's an old argument, one that we can take back to the Luddites of the early nineteenth century.

The Luddites were textile artisans who protested the use of mechanized looms. The looms, they argued, put too many skilled artisans out of work, because they could be operated by relatively unskilled (and cheap) labor. The Luddite response to the paradigm shift of mechanization was to wreck the machinery, which led to a harsh retaliation by the British government (the ringleaders were executed publicly). Some historians argue that the Luddites were anti-technology and anti-progress; others argue that they were just a pragmatic group of people trying to preserve their way of living (after all, churning out more product meant lower prices and lower wages).

Regardless of your position on the Luddites, there's no stopping a paradigm shift. The factories were coming, and industrialization would not stop short at textiles. The steam engine, the railroads, the steel mills, all the changes that would come would lead to more men leaving their villages and seeking work on the factory floors. That in turn would lead to more products out there in the world, and to a higher demand for services related to those products, as well as to more people being involved in the paperwork of commerce.

More recently, it is true that manufacturing jobs fled North America during the 1980s and 1990s, but that's only partially due to the rise of automation. Yes, we have more robots and automatic assembly lines putting products together, but we've also seen a shift in consumer values: We want cheap products. Cheap products don't get made by expensive workers in expensive plants. They get made in dollar-a-day factories in China.

Running parallel to the manufacturing decline, we've experienced a giant boom in the IT sector. Those jobs pay just as well, with great benefits and safer work environments than their counterparts in manufacturing. In fact, we've added entire sectors of professional specialties unimaginable in the 1940s and 1950s. For example, in a typical corporation you have business system analysts, enterprise architects, solution architects, customer relationship managers, Web designers, event coordinators, online community managers, intellectual property lawyers, and a whole host of other specialties. The only drawback is that all of these specialties require special skills and education; it isn't as easy as graduating from high school, walking across the parking lot to the tractor plant, and applying and getting a job as an apprentice welder or line worker.

Positive changes due to automation

You could argue that the widespread use and availability of technology today, and a professional technical class who know how to use that technology, has led to an unprecedented number of freelancers and consultants who have economically viable (and personally fulfilling) careers without much need to work for the Man.

In turn, this shift has created a hyper cottage industry of professionals that are turning out great work. It doesn't matter if the end product is a Web site, a business management assessment, or a carbon footprint calculation; what we're seeing as a result is a return to a kind of work we haven't seen since the giant factories started pulling millions of people away from their villages and farms and into the urban areas.

What we're seeing, in fact, is technology driving yet another paradigm shift. Increased automation is allowing more and more of us to make a living how we want to, using tools like Automator and AppleScript to do the work of 5, 10, or 15 people. It's kind of wonderful how the cycle has turned, that we're at this point where a highly trained artisan (for example, a consultant with 15 years' experience) can be as productive as 10 to 15 low-skilled workers.

Examples of Automation

It's time to leave the dry discussion of history to the past (no pun intended). Let's talk about something more relevant to what you're trying to do right now. You might be wondering what you can automate, and a good way to go about that exercise is to look at the list of tasks, processes, and systems that others have been able to automate successfully.

Of course, this entire section will be circumscribed by a simple filter: I'm not here to talk about automation of car factories, as I assume you don't have one of those handy. Instead, I'll focus your attention on how computer-based tasks have been automated by people and small businesses. The discussion will be largely limited to AppleScript and Automator, but keep in mind that any tool or system (for example, shell and Perl scripts, proprietary vendor tools, and others) that has simplified a process or system can also apply.

Data entry or collection

It's not something you think about every day, but one of the original boons of business computing was moving processes from paper-based to computer-based forms. In many cases, all that was needed was physically laying out the paper form onto the screen. Error rates fell because data entry clerks didn't have to parse out horrible handwriting. In many cases, a simple data entry screen removed various steps from an impossibly complex workflow, providing more cost savings.

Later on came further advances in usability, giving us more powerful and intuitive electronic data gathering systems that took full advantage of the medium. Why ask for a user's name and other personal data if they've already logged in and therefore identified themselves? Why ask for a date when the system already knew the date?

Data collection grew in other areas as well, as more and more applications joined the landscape, each with their own logging tools. Soon we were awash in error logs, messages, and all manner of electronic notes about all kinds of system, application, and network events. Soon we had more data from systems and applications than we did from humans. A single day on the stock market could generate more data than a month of operations in a pre-computer age corporation.

Data analysis

All that data meant an increased need for automated analysis. We needed some way to sift through all of that data in a quick fashion, looking for rough patterns, and trying to mine insight from the slew of messages. If we knew, for example, that an increase in traffic to a Web site at certain times of the day meant something different (say, more traffic from Europe than from Japan), we could make smarter decisions — like offering more Europe-specific messaging or special deals to the content mix of our site.

Of course, there's a big difference between real-time, near real-time, and historic (or even forensic) data analysis, but insight is insight, even for the little guy, because everyone from the small retail shop owner up to the head of a multinational organization now has at their disposal more data than they could ever consume in 20 lifetimes.

Put simply, the best data analysis tools allow the human operator to sift through great quantities of data and reveal patterns, trends, and relationships in that data. Are there certain days when our store makes more sales of certain products? Are there certain times of the day in which more sales occur? Do we make more profit from certain types of work, or when we work with certain types of customers?

Many data analysis tools are used to gain insight into customer behavior and understand what competitors are doing. Many of these tools are plugins or add-ons to existing software like spreadsheet and database programs.

Why is data analysis so crucial? Because good analysis is hard to come by, with too many organizations relying mostly on gut or instinctual calls. Think about the last time any company you worked for went about redesigning their Web site. Did they rely in any part on data analytics? For example, did they optimize for Firefox over Internet Explorer (as revealed by the preponderance of Firefox-using visitors in their log files)? Did they include information on their Web sites based on the actual needs, problems, questions, or issues posed by their customers?

Probably not — they most likely made a series of subjective decisions about their design and hoped for the best. The same is probably true about most of the millions of dollars spent on TV advertisements, but that's a whole other book.

Data munging

Data munging has a long and storied history in the information technology industry. Munging is all about writing simple scripts that do a specific task quickly and heuristically. Some malign munging as being the "poor man's ETL," but sometimes you can't get to the more sophisticated tools without first playing around a bit. And sometimes, there's just no point in building some perfectly automated tool that removes all human involvement from every step of the process.

In other words, sometimes all you need is a shovel, not a backhoe. You might need to open 10 log files with 10,000 lines of data each, scan each line for a certain character sequence, and if found, pull out the first 50 characters of each matching line. Then you might need to put all those matching lines into another file where another script, run every night at midnight, might compare those lines to the previous night's log. If a pattern or trend emerges, then that occurrence might cause

another script to load certain data into a spreadsheet that is e-mailed to an analyst for the next morning's analysis and briefing session.

Yes, your organization could go out and buy a huge enterprise system that does all this, or you could build the same system using Perl, UNIX shell scripts, AppleScript, Automator, or any other "simple" tools.

In fact, many organizations do just fine with data munging, mostly because they involve small pieces that can be mixed, matched, and adjusted many times without too many adverse consequences. In fact, the example I gave two paragraphs ago might be too involved, as many data munging efforts usually involve extracting data from certain files and then spitting out a result set that matches some simple search criteria. In Figure 1.1, I've included a look at a script that converts different image files to PDF files. Another popular munging activity is transforming one set of data, such as a comma-separated data file, into something else, like a set of SQL queries for database loading.

FIGURE 1.1

Sometimes automation is about converting from one format to another.

A lot of what you're going to do in this book could be categorized as munging. That's okay. I don't consider munging a pejorative term, and neither should you, particularly if a little bit of munging saves you hours of boring work.

Data extraction

Extraction is the E of ETL (extract-transform-load), and it usually involves handling either binary or unstructured data files. In other words, you start with a bunch of images, audio files, word processing documents, calendar events, e-mails, and other data and end up with structured data (database records, XML, SGML) at the other end of the process.

You normally see data extraction working in concert with other steps (such as data transformation, which I'll talk about next), but it's important to understand that automating extraction can be a valuable, beneficial exercise.

For example, one of the more time-consuming extraction exercises is pulling text out of Web pages or PDFs. Many tools (both commercial and open source) have emerged that allow users to quickly scrape the text out of Web pages and PDFs. In fact, Automator offers actions that allow you to do either with relative ease and comfort.

Instead of spending hours manually copying and pasting text from PDFs, now you can use Automator (and other tools) to painlessly grab what you need and then hand it all to another script, action, or tool.

Data transformation

Transformation is the "T" of ETL. Just like extraction, transformation is quite a big deal. In many cases, what you're trying to do is transform one kind of data (comma-separated values or HTML) into another format (SQL queries, XML, and so on). In other cases, you're not worried about converting, but about adding metadata, like you did with the Automator workflow in the Quick Start chapter. Sometimes all you'll be doing is mapping from one type to another — for example, you might have one spreadsheet with certain columns and rows that needs mapping to another structure.

Regardless of the kinds of transformation, you will probably see more munging activities than pure transformations, but that's okay.

Data integration

Whenever you have data coming from more than one source (like databases or applications) and then present that data in a unified view, you have a data integration opportunity. A typical use is to grab data from different data sources (such as XML files or database tables) and then create a data view on a Web or application dashboard.

You'll have plenty of opportunities to perform data integration tasks, as you pull from different data sources and combine them into custom views. Figure 1.2 shows a snippet of AppleScript that uses SOAP to get a stock quote. SOAP stands for Simple Object Access Protocol, and it's used to provide access to services or data without opening up the entire application stack to outsiders. For example, to use the AppleScript stock quote script, you don't need to know anything about how the stock quotes are stored on the remote server. All you need to know is how to get to the stock quote and how to retrieve the ones you're interested in knowing more about.

FIGURE 1.2

Sometimes automation calls for the use of SOAP or other Web services to get data.

Launching applications and scripts

One of the easiest and most common automation tasks involves launching applications and scripts. For example, you might want to open and close different applications at different times of the day; starting every day at 8am, you might want to launch Mail and iCal, then turn off all applications and put your Mac to sleep at 6pm.

The same thing goes for custom AppleScripts. You may want to launch a script manually, using a workflow action in Automator, as part of an iCal event, or as the result of another AppleScript that is itself scheduled to run at a certain time. You'll have ample opportunity to explore all these possibilities.

Figure 1.3 shows a snippet of AppleScript that opens the Accounts pane in System Preferences.

FIGURE 1.3

Sometimes automation simply opens a window or application.

Communications

A useful candidate for automation is communications — whether by sending scheduled e-mails to a client database (such as a weekly newsletter every Monday morning) or responding to or sending e-mails based on certain events or criteria — for example, setting up an autoresponder for when users send messages to a certain e-mail address.

In this case, think of Automator and AppleScript as extremely powerful additions to Mail, Address Book, and iCal — together, they can be tied together into extremely powerful and customized solutions. For example, you could track customers by using groups and metadata in Address Book, use an AppleScript to pull out customers by group, schedule the AppleScript to run through iCal events (every two weeks, for example, you could e-mail an invitation to a free workshop), and then use an Automator workflow to process the data from users who do attend the workshops.

Figure 1.4 shows a rather silly example, admittedly, of what happens when you use AppleScript to format the fonts in a mail message. You probably won't have much use for this kind of outcome, but you will most likely need to control the appearance of your messaging at some point, and so knowing about the capability is important.

FIGURE 1.4

Sometimes automation is just about having some fun.

Setting up reminders

One of the easiest things to automate is reminders, but don't fool yourself into thinking that only events in iCal have reminders. You can also use AppleScript to create reminders to revisit files with certain labels or Spotlight Comments (such as pending, draft, or in review). You can also create reminders that go far beyond the basic rules and flags associated with Mail, or tie together different applications to set up more custom reminders.

The idea is that you spend a lot of time working on files and data, but often not in ways that allow you to complete the work. It's also true that users tend to work with some files and data in an intermittent fashion, visiting the same folders and applications once a week or every quarter. The only way to stay sane is to set up reminders, and sometimes you need to go far beyond the simple reminders available in iCal.

Systems maintenance and backups

Mac OS X does an excellent job of providing system updates and backups (through Time Machine) on a regular basis, but there will come a time when you will need to create automated backups and maintenance programs of your own, even if all you're doing is mounting a network server once a day to check for updates or to back up files off site.

Why Automate?

Now that you have some idea of what can be automated, the question becomes, well, why should you automate? In some cases, automation would seem to require more time than would be required to set up an AppleScript or Automator workflow.

That supposition is only true if you consider that your work is a one-off deal, never to be repeated again. Given that view, yes, if you spend half an hour to automate what would normally take you 15 minutes to do manually, then you're wasting time. However, if you're going to perform the same workflow every day for the next week, or five times a month for the next year, then you'd be remiss in not at least considering automation.

Larry Wall, the inventor of Perl, said that one of the virtues of a great programmer is laziness. If you had to do the same thing more than once, a good Perl programmer would probably write a script and then be done with it. It might take them an afternoon to write the original processing script, but after that, the manual task that took an hour takes only two or three seconds.

Regardless, there are only a few good reasons for automation, but you're bound to hit one or another when you stop and consider the task before you. The reasons, broadly speaking, are as follows (and in no particular order):

- You want to save time
- You want to save effort
- You want to simplify a process
- You want to reduce errors
- You want to save on manpower
- You want to save money

Save time

Saving time is a primary reason for automating a task. Instead of slogging through hundreds of images, documents, or e-mails by hand, you could build a workflow or script to zip through the pile in mere seconds.

Figure 1.5 shows a simple Automator workflow that creates thumbnails of images — a process that takes a lot of time when done manually.

FIGURE 1.5

A simple Automator workflow that creates thumbnails can save a lot of time.

Save effort

Saving effort is not the same as saving time, not at all. You might have a process that takes just a little bit of time, but it might involve a lot of different people or different applications, or different types of interactions at different times of the month. Any of these combinations will quickly ramp up your level of effort.

Figure 1.6 shows an Automator workflow that extracts text from a Web page and then speaks that text out loud — normally, extracting raw text from a Web page is a laborious process, but Automator simplifies it easily.

FIGURE 1.6

Another Automator workflow might extract text from a Web page and speak it out loud, saving a lot of effort.

Simplify a process

Simplification is another easy win. If you have ten different manual steps, automating even half those workflow steps will create a good return on your investment. Even if your efforts result in three or four different workflows and scripts working together, you end up with a much simpler process.

Reduce errors

Given enough steps in a manual workflow, or enough time spent doing manual tasks, eventually it will lead to one or more errors. It happens to everyone. Errors happen because workers get tired, or get distracted by a phone call or a visitor, or because they lose interest. Maybe they didn't understand the initial instructions ("Oh, I thought you wanted all these images cropped to 400 x 400!").

In any case, setting up a repeatable workflow or script turns the task over to the computer, and the computer never gets tired, never gets distracted, and never loses interest. Whatever instructions you give it, rest assured that those instructions will be followed to the letter.

Figure 1.7 shows an Automator workflow for properly labeling files based on a user's selection in the Finder.

FIGURE 1.7

Another Automator workflow might, in two easy steps, label files for you.

Save on manpower

If you've got five people working on a manual task (processing thousands of files by hand) and you automate that task, maybe all you need is one person to analyze the results of the automated processing, and the other four can be retasked to more important jobs.

Save money

At the end of the day, just about any automation will save you money, because it will generally simplify processes, reduce the amount of time it takes for task completion, and reduce the number of errors that cause so much pain. The point is, how much money will you save? For example, if you reduce a task from 50 person-hours a month to 1 person-hour a month, you've just saved at least $2,400, if you count each person-hour at $50/hour.

Other tasks, like reducing the number of steps in a marketing workflow involving communications with customers, might be harder to tally up, but after a while you'll learn how much time is spent by marketing staff after automation compared to before automation, and you'll be able to work out the savings.

Summary

In this chapter, you learned a bit about automation — what it is, what kinds of things you can automate, and why you should do it. Your Mac sits at the tail end of a long history of automation — you now have more power at your fingertips than entire corporations had 40 or 50 years ago.

CHAPTER

2

Automation from a User's Perspective

Now that you've taken the tour of automation, it's time to take a look at automation options in Mac OS X. If you're looking for the specifics of automation from a user's point of view, then this is the right place. This information should help guide you, whether you're a regular user, a designer or developer, or a small-business owner.

IN THIS CHAPTER

- **The Mac user's viewpoint**
- **The professional's viewpoint**
- **The small business owner's viewpoint**

The Mac User's Viewpoint

Apple products are known the world over for their elegant design, clean user interfaces, and superb engineering. Mac OS X and the Aqua interface present the user with beautiful, easy-to-use screens and user interface elements. The typical Mac user is thought to be someone in the creative fields (for example, designer, photographer, videographer, or podcaster), but many others who never do the typical creative work of designing Web sites or enhancing photographs are also Mac enthusiasts.

In fact, when you look at the growing number of Mac users, you find a great deal of stay-at-home moms, retirees, students, and other regular folks. They use their Macs to surf the Web, send and receive e-mail, keep track of their calendars, listen to music, play movies, and perform all kinds of routine computing tasks.

None of these people are using their Macs to solve complex mathematical problems, take on global warming, or run a globe-spanning corporation. If you're one of them, don't despair, because automation, and particularly the tools already built in to your Mac for free, can still make your life a lot easier.

Typical activities to automate

If you're an everyday, common, regular Mac user, you probably haven't given much thought to tasks that can be automated. You've also probably not even thought about a better way of doing things because you haven't had much of a chance to. For example, you might go months or even years without even having to rename more than two or three files at one time.

However, at some point, you will be confronted with a big project, something that, without a doubt, makes you feel tired just thinking about it. You might have to wade through hundreds of photos. You might need to rename a thousand files. Or you might need to organize an entire group of files and folders.

Here are some ground rules to consider when thinking about automation. With any luck, these rules will help you figure out what to automate.

- **If you need to do anything more than once, consider automating the process.** If you need to rename one file, then you probably don't need to automate the process. If you need to rename twenty files, fifty files, a hundred files, then you definitely need to consider automation. Even if you think you're only going to rename two files right now, but will have to do it daily for the next month or year (or however long) then you need to consider automation. It doesn't matter if you write a script or put together a workflow; you will save time, reduce errors, or stave off mind-numbing boredom (or all three at the same time). Some examples include:
  - renaming files;
  - creating thumbnails of many photographs;
  - adding metadata or Spotlight Comments to numerous files;
  - adding labels to files; and
  - finding or filtering mail messages, audio files, images, files, contacts, or events.
- **If the task involves many little steps, then consider automation for the sake of simplifying your life.** Any task that takes at least three steps is a prime candidate. For example, if you find yourself searching for certain types of files, renaming them, and then archiving them, then you'd probably be well served by either a quick Automator workflow or AppleScript. These tasks might include:
  - archiving or backing up files;
  - converting files from one format to another;
  - extracting data from one application and making it available to another application; or
  - combining data from different sources.
- **If you find yourself doing the same task or series of tasks today that you did three months ago, then automate it.** There's nothing more frustrating than running a series of actions or tasks manually while thinking to yourself, "This is so tedious, I thought that I'd never have to do this again," particularly if the task is boring or prone to a lot of errors. These tasks might include:

  - adding audio files to playlists;
  - working with QuickTime movie files;
  - finding and printing out events and to-do lists; and
  - setting up groups for e-mails.

- **If you find yourself not wanting to do a task, then it's a sure sign you need to automate it now and avoid future pain.** Any task that makes you shudder, groan, or procrastinate is a good candidate for automation. Why grind yourself to bits doing stuff that computers are good at? Some avoidable tasks include:
  - any system administration task;
  - any updates, installs, or maintenance; and
  - any task involving a utility script.

- **If you find yourself repeatedly forgetting to do something, then automate it.** If nothing else, you can use iCal to send you a simple e-mail reminder. More likely, you'll be using iCal to kick off a certain workflow at different times or on certain days of the week. For example, if you're forgetful like me, you can set up little programs to help you remember what items to plant in your garden at different times of the year. (See, it doesn't all have to be about computing!)

Taking inventory

Even the most casual or intermittent Mac user has more data at their fingertips than an entire family would have seen 50 or 100 years ago. The typical Mac is stuffed with iTunes, iMovie, iPhoto, and iCal data; crammed with files in all kinds of formats ranging from plain old text files to word processing documents; and teeming with address book contacts, e-mails, and other productivity-related information like to-do lists.

Here's a quick inventory list that will help you ascertain what tasks you might need to automate.

- **Open a Finder window to your Home directory.** What files and folders need special archiving? I'm not talking about items that are covered by Time Machine (you are using Time Machine, right? If not, invest in a backup drive or a Time Capsule — it's cheap compared to losing everything). I'm talking about items that are no longer being used and could be stored in ZIP archives. Or maybe you need to keep track of recently modified items (see Figure 2.1). If you're not set up with Time Machine, don't worry; Automator can help you back up different folders to different locations, like your MobileMe account or an FTP server.
- **Open Mail and take a tour of your e-mail and folders, and think about your e-mail activities.** Can some of those messages be combined, stored as text, and removed from your folders? If so, it's time to think about using Automator to pull those e-mails out. Do you find yourself sending a lot of e-mails that contain the same content? If so, it's time to automate the process, which you can do with a simple Automator action that lets you fill in what you want to say in your response (see Figure 2.2). Are you distracted by e-mail?

Then it's time to use Automator and iCal together to start up and shut down Mail at certain times of the day, allowing you to get your productivity back.

FIGURE 2.1

Even a simple workflow (like finding files modified in the previous two weeks) can make life easier.

FIGURE 2.2

Workflows can help you create e-mails.

- **Open iCal and your Address Book, and take a quick inventory.** Do you find yourself needing to group different contacts or events, but now it just looks hopeless because there are several hundred in there? Automator and AppleScript can help. Do you need a summary of the last two months' events (see Figure 2.3)? Do you want to find everyone with an e-mail address but no phone number so that you can send them a note asking for that phone number? You can do all these things with Automator and AppleScript.

- **Open iPhoto and take a look at all your images; do the same with photos stored on your hard drive and backup drives.** Are they organized the way you want them? Do you want to add ratings, comments, or other metadata? Do you need to see your photos in contact sheet format for easier editing? Do the images require cropping or resizing (see Figure 2.4)? Do you need to export your images to another application (like Keynote or iMovie)? Then you can automate this process with a simple script or workflow.

FIGURE 2.3

Workflows can help you summarize event data in iCal.

FIGURE 2.4

Cropping images is a perfect task for automation.

- **Open iTunes and survey your music files.** Are there music files that have missing metadata (see Figure 2.5 for an example of finding "live" music)? Do you have unrated items in the list? Are there a bunch of miscellaneous podcast files mixed in with your music? Are you trying to recategorize many different files at once? You've heard me say it many times, but you can hear it again: Automator and AppleScript can help.
- **Open Safari and take a quick look at your RSS feeds and bookmarks.** Are there easier ways to consolidate the information you already consume? Can you automate your morning's reading into an easy-to-digest form that consolidates your reading (see Figure 2.6)? Automation can help here too, even if all you're doing is opening up certain Safari windows.

FIGURE 2.5

Finding live music in iTunes and adding comments

FIGURE 2.6

Automator can simplify your morning's Web reading.

The Professional's Viewpoint

If you're a software developer, Web developer, iPhone developer, designer, videographer, or podcaster, then your needs will go above and beyond the needs of the casual user. You'll still be trying to figure out the same thing as the casual user, only more often: How can you make yourself more productive?

If you're an employee, you'll want to get more out of your day — you're busy enough with all your meetings, projects, and sundry interruptions. If you're a freelancer or independent, every little bit of productivity you squeeze out of your day translates into more profits and happier clients. Either way, if your Mac is trundling along doing basic, repetitive, mundane tasks for you, then you're free to apply your brainpower, talent, and intuition in other ways.

The software developer's needs

If you're a serious software developer, then you probably have some kind of source code repository up and running (such as Subversion), and some kind of process for checking your code in and running a test environment.

However, even if you do have all of that, you can still use AppleScript and Automator to help automate various tasks:

- **Setting up watch folders that warn you if something has changed.** This is great for watching log files.
- **Integrating AppleScript and Automator with shell scripts.** This can help you chain together any number of custom scripts that work with your development environment (see Figure 2.7).

FIGURE 2.7

Automator can also run shell scripts.

- **Creating scripts that schedule automated testing.** Remember, you could use AppleScripts or Automator workflows to simulate a visitor to a Web site.

- **Create scripts that automate package creation or builds.** This kind of automation can save a developer a great deal of time and effort, especially if 'they're dealing with numerous builds or packages a day (see Figure 2.8).
- **Creating specialized backups by integrating with FTP clients.**
- Automating the process by which data is requested and sent using Web Services protocols like SOAP or XML-RPC.

FIGURE 2.8

Automator can help developers to automate package building.

The system administrator's needs

Just because you're on a Mac doesn't mean that system administration tasks will take care of themselves. Fortunately, you can schedule all kinds of workflows using iCal or UNIX tools like *launchd*. For example, you could do the following:

- **Search through log files on a regular basis to make sure nothing is amiss.**
- **Delete files on a regular basis using a scheduled workflow (see Figure 2.9).**

- **Schedule the activation and deactivation of firewall rules at different times.**
- **Provide more dynamic monitoring of system parameters by integrating with different utilities and shell scripts.**

FIGURE 2.9

Automator can help you find files that are very old and place them in the Trash.

The creative's needs

If you're a photographer, videographer, podcaster, or visual designer, then you need raw processing power. You often have to deal with hundreds of large binary files, such as movies, photographs, and audio files. Some may be in common formats like JPEG, MP3, or MOV, while others might be in more specialized formats.

You can automate:

- processing large numbers of files all at once;
- processing many different steps across many files at once (see Figure 2.10);
- periodic review of files to match your publishing workflows;
- transformation of files into different formats;
- annotation and metadata tagging of many files at once; and
- filtering, finding, labeling, and organizing many files at once.

FIGURE 2.10

Automator can help creatives to master tasks that have many tedious steps.

The Small Business Owner's Viewpoint

There's a myth out there that Macs are only for fun and not suitable for business. However, millions of small business owners who rely on a Mac's vaunted stability, performance, and virus-free operation can tell you that Macs aren't just for home movies.

If you're running a small business, automation opportunities lurk around every corner. In fact, you could even set up a workflow action to run a separate workflow (see Figure 2.11). The question is: Where do you start? Mostly it comes down to some of the things I mentioned in Chapter 1: How do you save effort, time, and money? In other words, how can you stretch your dollars and avoid costs by using automation techniques?

FIGURE 2.11

Automator can run self-contained workflows built by you or others.

What processes need automating?

You can start thinking about this by looking at all the processes you have in your business that need automating.

- **Marketing and Sales.** Most businesses need to get the word out to customers, and then respond to customer requests for information. You can easily automate e-mail responses, for example, as well as requests for portfolio items that go above and beyond whatever you might already have on your Web site. Using AppleScript, you could tie together Address Book, iCal, and Mail applications into a very workable customer relationship management tool. You can also use AppleScript and Automator to help you tie in to Keynote, Pages, and other applications.
- **Finance and Accounting.** AppleScript already allows you to tie into many of the functions made available by Numbers and Excel. Other tie-ins also exist for other well-known billing, hourly time tracking, and accounting software. If you want to build your own custom applications, it's possible to create a time-tracking script that prompts you for billable hours at the end of each business day, and then stores this information in an easy-to-use format like Excel.
- **Production.** Every business produces something of value, whether it is advice, delicious meals, or on-time delivery of couriered documents. Automation can help you in all areas of production, for example, helping you keep track of data that goes in a final report, or tracking your numbers on profitability while your employees work to deliver product. It's up to you to imagine the possibilities.
- **HR and Benefits.** Like finance and accounting functions, myriad opportunities exist for automation. For example, you can automate the way your employees request sick leave, and use a simple system for keeping track of this time. You could automate reminders of upcoming national holidays, special workshops and training dates, or special observances.
- **Facilities and Security.** In a modern small business, you may only have to take care of a single office, but you may also have customers coming in and out and a small warehouse that needs watching. Automation allows you to be in more than one place at a time, if you work it right. Different systems could be set to send in regular reports on temperature, events involving motion-detection cameras, or badge access that keep you up to date on what's going on. You can also use different AppleScripts, shell scripts, and Automator workflows to make sure the right data is brought to your attention.

What files and data need automating?

If you look at your business files and data, it's easy to start thinking up ways to automate how you process them. For example:

- **Spreadsheets need hard numbers and financials.** Where do those numbers originate? Other spreadsheets? User input? Databases? How can you automate the sifting of all this data?
- **Databases are everywhere, and their data can be reused in many contexts.** For example, it's very easy to set up an Automator workflow that converts CSV (comma-separated values) data from a spreadsheet into SQL queries (see Figure 2.12). You can also create SQL queries and execute them against SQL Lite using Automator (see Figure 2.13).

FIGURE 2.12

Automator can help non-experts convert CSV data to SQL.

- **Memos, reports, and other documents are often based on templates.** For example, if you're a consultant, your assessment reports are typically structured in very similar ways across industries and clients. So, too, are any white papers, brochures, or eBooks that your business creates. Also think about how to use automation to extract content from your blog and repackage it as information products. The only thing keeping you from this new line of revenue is the time it takes to do this manually.

FIGURE 2.13

Automator can execute SQL queries.

- **Presentations often contain many images, and these can be gathered in an automated fashion (see Figure 2.14).**

FIGURE 2.14

Automator can help you gather images for Keynote presentations.

- **E-mails in any business are usually overwhelming in number.** It's time to figure out how to make automation work for you, and let it level the playing field a bit. For example, a simple AppleScript might find different messages sent by certain people with certain subject lines. These could become the raw data for a variety of workflows that might convert these e-mails into spoken-word files that you can play on your iPod during your commute.

Where's the ROI?

The bottom line in business is the bottom line. If your automation effort provides return on investment of some kind, then you're almost morally obligated to try it out. However, don't limit your imagination just because the task seems small or inconsequential — you might need to repeat the very same task intermittently (say, once every five to seven months over the next ten years) or for a certain kind of client (for example, a public, non-profit organization that needs their reports done in a certain way) or because you've hired a certain kind of worker (for example, the temporary contract iPhone developer versus your full-time staff developer).

Always remember the programmer's mantra: If you have to do it more than once, it's a good candidate for some kind of automation. Just make sure that you get the kind of return on your investment that you want. How can you tell? There's lots of ways to calculate ROI on automation, but an easy rule of thumb can guide you: take the amount of time that it would take you to do the job manually and multiply it by your hourly rate — now think to yourself, is it worth it to automate?

Think about how many times you would have to do this manual task over a longer period, like a quarter or year. That total amount of money you're spending on the manual task is your opportunity cost, if you will.

For example, let's say that on most work days, you have to spend an hour extracting text from client PDFs. Let's say you do this on average twice a week all year round. That's 100 times a year, multiplied by your rate of $100 dollars an hour. That's $10,000.

Once you have your opportunity cost, you have to ask yourself, what could I do with that 100 hours a year that I'm stuck doing this manual extraction of text from PDF files? Would I be able to market my business more effectively? Could I take that time off as vacation hours (that's two weeks vacation — think about it!).

Is it worth it to spend a few hours to automate this task using Automator or AppleScript? A few hours up front might reduce your involvement to just five minutes a day, a 12x reduction of time. You can still keep billing the client for your expertise, of course, but now you're spending 8-9 hours a year instead of 100.

You could argue that it would be worth it to spend up to 99 hours that first year just to figure out how to not have to manually extract text from PDFs, because if for nothing else you might get even larger ROI in the following year.

That's how you should think about automation.

Automator and AppleScript are not the answers to everything. You may find yourself in a situation where you simply can't get something to work the way you want. If you're an experienced Perl scripter, for example, you may find AppleScript to be too limiting. Or you might find yourself in a situation where you can only automate 30 percent or less of a procedure. Or you may find yourself knowing so little about something that you stumble through it manually for many days or weeks before you get around to automating anything.

Don't let these scenarios bother you. No situation will be 100 percent perfect or ideal. And believe me; it's smart to work through a process manually a few times, especially if you don't completely understand it. Otherwise, you'll find yourself in a situation where you automate a bad process, or unknowingly insert a number of errors. Then all you've accomplished is more and more chaos, but this time implemented at the speed only a computer can achieve.

Summary

In this chapter, you were introduced to a lot of ways that you might automate on a Mac — whether you're just a stay-at-home mom, hobbyist, freelance photographer, Web developer, or small business owner. With AppleScript and Automator, you're really only limited by your imagination. Every successful automation project always begins with some brainstorming; hopefully this chapter has stoked some creative thinking on your part.

CHAPTER 3

Automator Basics

In this chapter, you'll learn about Automator: its interface, how to put together workflows, what actions are, and many of the same topics covered in the Quick Start chapter, but with more detail. By the end of this chapter, you'll also learn how to save your workflows in different configurations. All of this will prepare you for Chapter 4, in which you'll learn about more advanced topics like variables and loops.

IN THIS CHAPTER

- Opening Automator for the first time
- The Automator interface
- What are workflows?
- What kinds of workflows are possible?
- What are actions?
- What else can I do with actions?
- How can workflows be saved?
- What are Templates?

Opening Automator for the First Time

You can find Automator under Applications➪Automator. When you open it for the first time, it appears in your Dock. For simplicity's sake, and to make working with it much easier, right-click the Automator icon in the Dock (it looks like a little robot holding a pipe) and choose Keep in Dock from the pop-up menu.

Here's something else that you'll notice when you open Automator: It asks you to select a template. Don't worry about any of that now; just select Workflow and click Choose. It's time to take a quick tour of the Automator interface.

The Automator Interface

This section takes a closer look at the Automator interface. You got a very brief rundown in the Quick Start chapter, and I'll be covering some of the same ground here, but this time, I'll go at a slower pace.

Figure 3.1 shows the entire Automator interface. It's a shot that's pretty similar to the one you saw in the Quick Start chapter.

FIGURE 3.1

The Automator interface

The Automator interface is extremely simple. It consists of a toolbar that makes up the entire top panel. Below that toolbar are two panes: The left pane is where you choose your actions and variables, and the right pane is where you drag those selected items to make your workflows. Last but certainly not least, there's the footer, which contains special controls for listing variables, your workflow logs, and other functions. I will describe each part of the interface in the following sections.

The Toolbar

The Automator Toolbar is shown in Figure 3.2. It contains five buttons:

- **The Hide Library button, when clicked, allows you to show only your workflow.** This is a handy button to have when you're trying to debug a workflow and need the entire workspace.
- **The Media button, when clicked, shows your music, photo, and other media files stored on your Mac.** You can then easily add media files to your workflow — for example, this is a good way to select songs or photos that need processing.
- **The Record button allows you to manually record workflows.** This button is offered just in case there aren't any available actions that meet your needs.
- **The Stop button stops a running workflow.**
- **The Run button plays a workflow.**

FIGURE 3.2

The Automator Toolbar

The Actions library

Figure 3.3 shows the Actions library. As you can see, there is a list of actions (under Library) categorized by function, followed by a list of other Smart Groups like Most Relevant, Most Used, and Recently Added.

Notice that below this pane is an information pane that provides a description of the current selection. Currently, there is no selection, and so the information pane contains general instructions for working with workflows.

Furthermore, clicking any group in the first column displays a list of appropriate actions in the right column of this pane. Figure 3.4 shows what happens when you click the Calendar group. Using the search box also displays a list of actions. Figure 3.5 shows actions displayed for a search on **scale** — appropriately enough, you see actions related to scaling images.

FIGURE 3.3

The Actions library of Automator

FIGURE 3.4

Clicking the Calendar group in Automator

FIGURE 3.5

Searching for actions in Automator

In most cases, the application groups are pretty self-explanatory. The Calendar actions allow you to work with events. The Contacts actions allow you to work with items in your Address Book. Files & Folders actions relate to actions that tie in to the Finder and Spotlight. Internet actions relate to Safari and RSS. Mail actions allow you to work with e-mail messages. Music actions allow you to work with iTunes files and other audio files. PDFs actions allow you to manipulate PDF documents. Photos actions allow you to work with iPhoto and image files. Text actions allow you to manipulate text.

The Variables library

To switch to the Variables library, simply click Variables, as shown in Figure 3.6. Please note that this is not a complete list of the possible variables in Automator; it is only a list of system variables that come pre-loaded. You can create your own variables as needed (and as described in the Quick Start chapter — but don't worry; I'll cover this topic again in Chapter 4).

Just as with the Actions library, variables come in groups, which you can use to filter the list (see Figure 3.7).

You can also run searches on variables and get back a more manageable list (see Figure 3.8).

FIGURE 3.6

The Variables library

FIGURE 3.7

Selecting only the Date & Time variables in Automator

The Workflow paneThe Workflow pane is where you build your workflows. Figure 3.9 shows an empty workflow pane waiting for you to drag and drop actions onto it. Because a good part of this book deals with actions in this pane, I'll leave it at that for now. You'll get plenty of coverage as the book goes on, starting in the next chapter.

FIGURE 3.8

Searching for variables in Automator

FIGURE 3.9

An empty Workflow pane waiting for actions

The Footer

The Footer is really two footers, one at the bottom of each pane. Figure 3.10 shows the footer under the left pane. The footer has three buttons or controls on it. The first button allows you to create, edit, and remove groups and Smart Groups for your actions and variables. This, in effect, allows you to reorganize your actions and variables (and create dynamic groupings with Smart Groups) as you see fit.

FIGURE 3.10

The left footer in Automator

What's the difference between a group and a Smart Group? A group is hard-coded, containing only those actions or variables that you put in that group. For example, you might create a new group that contains all actions that start with the letter B (for whatever reason, let's say that you need those all in one group). Whatever you manually place in that group stays in that group.

A Smart Group (see Figure 3.11) is very similar to a Smart Mailbox in Mail or a Smart Folder in the Finder. However, they only work with actions, not variables. Let's say you want to have at your fingertips the most frequently used iPhoto-related actions. In this case, you'd select New Smart Group and then add your criteria with the sheet that appears, making sure that you select iPhoto for Application. Then you would add Times Used and make sure it says greater than 3 (or whatever you think will allow your iPhoto-related actions to make the list).

FIGURE 3.11

Creating a Smart Group in Automator

Without too much work, then, you've created a custom grouping that will help you keep organized.

The second control is a downward-pointing arrow. Click it, and the information box that accompanies each action or variable you select will disappear, giving you more room to view your listings.

The final control, over to the right edge of the left pane, allows you to resize the actions/variables pane. You can also hover the mouse pointer over the line dividing the two main panes, wait until the cursor turns into a resize cursor, and just drag left or right to resize. Being able to resize your panes may seem like a small thing right now, but eventually, you'll be working with fairly complex workflows, and you'll want more room to work with the right pane.

The footer for the right pane has two buttons side by side. The first button shows three horizontal blue lines, and the second shows two horizontal blue lines. The first button reveals the Workflow Log (see Figure 3.12).

This log will come in handy as you build and troubleshoot complex workflows, as it shows you a breakdown of every action and variable in your workflow. The second button reveals the Variable list (see Figure 3.13), which shows all the variables you're using (whether system variables or ones you've defined) in your workflow. In the Variable list, the name of the variable is listed on the left, and its corresponding value shows up on the right.

In both cases, you can simply hide the Workflow Log or Variable list by clicking the button that opened the view. You can also double-click the dot on the bar on top of each list to make the view disappear.

FIGURE 3.12

Showing the Workflow Log

FIGURE 3.13

The Variable list in Automator

What Are Workflows?

Think of workflows as Automator documents. Just as your word processor creates a certain type of document, and your spreadsheet creates another type, the workflow is simply a collection of instructions (called actions) that are arranged in a special order to help automate a process.

A workflow can contain just one action or many dozens of actions, and can even loop or reference other workflows. Furthermore, you can start workflows in one of three ways:

Note

If you're still running Tiger, note that its Automator is more limited than Snow Leopard's. Looping, for example, isn't available in Tiger.

- by starting Automator the old-fashioned way and starting a workflow from scratch;
- by using a Templates to help you get started quickly; or
- by right-clicking a list of Finder items and starting a workflow that way.

Workflows are built by dragging actions or files into the workflow pane and arranging what happens in a specific order so as to effect a certain outcome. For example, you may have 100 image files that you need to rename, adding today's date to the end of the filename. You may also want to replace all spaces in the filename with an underscore (_). Oh, and at the last minute, your boss decides that he wants a thumbnail added to each image when you're all done.

The workflow in Figure 3.14 starts by allowing the user to make their own Finder selections — presumably, the user will select images. You can't see it from the screenshot, but I've placed the starting point in the Pictures folder. Once the user makes their selection, the workflow copies the images to a new folder (one I've set up on the Desktop, but you can choose whatever target you want). Then I put the files through two Rename actions. The first action adds the date and time to the end of the filename. Notice that I'm using the current date and time with an underscore separating the first part of the filename from this new data I'm adding. Also notice that I'm using the Year Month Day format, without separators — again, you're free to use whatever you think is necessary.

The second Rename action simply removes any spaces from the filename, replacing them with an underscore. Finally, I add a thumbnail to each image, and I'm done. The workflow is now useable by anyone with a Mac who has enough talent and skill to select images to begin the process.

Speaking of sharing workflows with others, I'll touch briefly on saving your workflows — don't fret, I'll cover it in more detail later in this chapter, and you'll get a lot more exposure to these options as you read on. There are three main ways to save your workflows. The simplest and most common is just to save them as workflow documents. I place mine in a workflow folder in my home directory so that I don't lose track of them. You can also save them as stand-alone applications and as plugins for other applications, such as the Finder or the Print menu item.

FIGURE 3.14

A sample workflow that processes images

What Kinds of Workflows Are Possible?

In order to create an Automator workflow, you have to meet two conditions:

- **Do actions exist for what you're trying to do?** Many applications come with actions already created for them; others, however, may not have any actions or just a limited array of options. You can create your own custom actions with AppleScript and share them as you like.
- **Do the actions work together properly?** For example, it's important to chain together actions such that the output (or results) of one action becomes the proper input for the next action.

Those are about the only restrictions you will normally run into — well, there might be the whole issue of "how much memory does your Mac have" or "how much time do you have to think up a workflow" but those are really issues of performance/capacity and training. I seriously doubt you'll run into any memory limitations, even if you are putting together really big or complicated workflows. And, of course, by the time you get done with this book, you should be thoroughly versed in all manner of workflows.

In Part III, where you'll embark on 50 automation projects, you'll learn how to work with the following:

- **Files and folders,** including processing specific files, renaming them, creating aliases, and working with disks;
- **Music and audio files,** including automating your iTunes playlists, adding songs, playing specific songs, and converting audio to text;
- **Photos and images,** including applying color changes, cropping images, and creating thumbnails;
- **Text files,** including opening them, converting data to text, adding text to existing files, and working with BBEdit; and
- **Custom projects,** including tasks that involve Address Book, iCal, Safari, and Mail.

For each section, you'll learn how to create basic Automator workflows, then learn how to do the same thing (or similar things) with basic AppleScript, and then learn how to combine the two to create variants of the basic workflow. Along the way, you'll try out different save options (like creating Finder plugins, for example) and different options for running the workflows (manually, using AppleScript, iCal, and so on).

Here's an important question that needs some coverage: What can't you automate with Automator? It's very simple, really: Automator is no good at making decisions for you. It does exactly what you tell it to do, without fail, but it can't make any complex decisions based on output. It is not a scripting language, just a visual tool for automating tasks. You'll find it's useful in about 50 percent of situations — for the rest you'll need AppleScript.

Feel free to investigate what's possible by going to `www.apple.com/downloads/macosx/automator` (Appendix A also contains a list of other Web sites to visit). There you'll find a growing list of Automator actions provided by other developers (see Figure 3.15). You'll find Automator workflows and actions for specific applications (such as Adobe Illustrator, InDesign, Photoshop, and others), general workflows for specific tasks (adding Google Analytics code to your iWeb sites, for instance) and other handy tools that activate screen savers, convert text to different audio formats, and more.

When you download these actions, feel free to deconstruct them and figure out how they work. By the time you finish reading this Bible, you'll have more than enough expertise to know how to do that.

In fact, as you get better at making workflows, feel free to submit your creations to the community by clicking the submit downloads icon on the sidebar of the Automator actions page (see Figure 3.16). Simply log in (or register as a member of the Developer Connection community) and submit your work.

FIGURE 3.15

A short list of Automator actions available for download at `apple.com`

FIGURE 3.16

You're free to submit your work back to the community.

What Are Actions?

Actions are the building blocks of workflows. Most of the time, you'll be working with pre-built actions provided by Mac application developers. For example, if you want to automate tasks in Keynote (a presentation tool similar to PowerPoint), then you'll use the actions created by the Keynote developers and made available to you. You can also create your own custom actions using AppleScript, and download others at the Apple site (`www.apple.com/downloads/macosx/automator`).

Each action usually performs a single task. That task might be writing text to a file, downloading a URL, cropping an image, or waiting for the user to select a list of files to operate on.

Some workflows may consist of just one or two actions, but in most cases, you'll find that you need five or more actions strung together sequentially in order to get the job done. You've already seen many examples of how this is done, but the most important part of creating a workflow is figuring out beforehand what it is you want to accomplish. That being said, however, many people find that working with Automator in a heuristic fashion (try-fail-try again) not only lets them learn about the system but also gives them pretty good results.

In a workflow, an action can usually get input from a previous action, and pass along output (or results) to a successive action. For example, a typical workflow might do the following:

- select text files from a specific folder;
- archive those files; and
- attach the archive to an outgoing e-mail.

This workflow features specific points where results passed from one action become the input for the very next action. The list of selected text files is input for the archive action, which then results in an archive file that serves as the input for the new mail message action in the next step of the workflow (see Figures 3.17 and 3.18).

FIGURE 3.17

The list of text files becomes the input for the archive action.

FIGURE 3.18

The created archive becomes the input for the new message action, which then attaches the archive.

The result is what you'd expect: a new e-mail message window with whatever text you've filled in (in this particular case, just a subject line of **here's the backup!**) and the attached archive in ZIP format (see Figure 3.19).

FIGURE 3.19

The resulting e-mail message with the attachment

What Else Can I Do with Actions?

I noted in the Quick Start chapter that each action has a row of buttons along the bottom: Results, Options, and Description (see Figure 3.20).

- **The Results button allows you to see the results of running an action.** You can see results in three views: icon (which shows your results as a list of thumbnails), table (which displays your results in neat rows), and list (which shows your results as an AppleScript list).
- **The Options button allows you to get input from the user.** By checking the box next to Show this action when the workflow runs, you can create a situation in which a user makes a selection or has to click a button to continue. This is perfect for actions that require a user to enter a value or make a selection.
- **The Description button lets you see a description of the action.** This is the same description that shows up in the information pane when you select an action in the left pane.

FIGURE 3.20

The Results, Options, and Description buttons on an action

Another handy trick involves collapsing individual actions in your workflow so that you can better see what's happening. This is a good ability to have if you're trying to debug a recalcitrant workflow, or if you need to know what's happening at a glance (see Figure 3.21).

To collapse an action, click the downward-pointing arrow (it's called the disclosure triangle) to the left of the Action's name in the workflow. The arrow turns over on its side, and the information window for the Action slides up and out of view. To uncollapse an action, simply click the arrow again — it changes back to a vertical orientation and the information window slides down into view.

FIGURE 3.21

Collapsing actions in the workflow allows you to see what's happening at a glance.

One more note about actions and then I'll move on. You can easily and quickly disable an action in a workflow at any time. Why would you want to do this? Well, perhaps you are testing different parts of a workflow, and you don't want a certain action to run (for example, the action might involve copying hundreds of files from one directory to another, and you're not ready for that to actually happen).

To disable an action, just right-click the action's name and select Disable from the pop-up menu (see Figure 3.22).

FIGURE 3.22

Disabling actions in your workflow

As you can see from Figure 3.22, you can also use this method to duplicate, move, or rename an action — not to mention ignoring input, which allows you to run an action but disregard any input that is fed to it. You'll learn more about all of these options in Part III as you build out the 50 Automator options.

How Can Workflows Be Saved?

You have three options for saving your workflows. The first and most common option is to save as a workflow (.workflow) file. The second is to save as an application that you can double-click or drag files and folders on. The third is to create a plugin for another application or context (such as the Finder, the Script menu, or the Print dialog).

Save as workflow

Saving your workflow as a workflow document is the most common way to go (see Figure 3.23). When you save your workflow, place it in a special folder (I keep all of mine in a folder called Workflows in my home directory).

FIGURE 3.23

Saving as a workflow

To save your workflow, follow these steps:

1. **Press ⌘+S to save your work, or select File ➪ Save from the menu bar.**
2. **Enter a name for your workflow in the field provided.**

3. **Select a folder in which to save your workflow.**
4. **Make sure that you've selected Workflow from the File Format pop-up menu.**
5. **Click Save.**

You can then open your saved workflow document by double-clicking it, dragging it to the Automator icon on the Dock, or selecting File➪Open from the Automator menu bar. In any case, to run your workflow, click Run on the toolbar.

Save as application

Saving a workflow as an application lets you run the workflow without having to open Automator first (see Figure 3.24). This can be a good thing if you want a user to double-click the application icon or drag files and folders on top of it (for example, a folder full of images that need processing by your workflow).

To save as an application, follow these steps:

1. **Press ⌘+S to save your work, or select File➪Save from the menu bar.**
2. **Enter a name for your workflow in the field provided.**
3. **Select a folder in which to save your workflow.**
4. **Make sure that you've selected Application from the File Format pop-up menu.**
5. **Click Save.**

FIGURE 3.24

Saving as an application

Save as plugin

Saving a workflow as a plugin is probably the most flexible and useful option you have (see Figure 3.25). Think of it as a way to add exciting new features to your existing applications and to the Finder. For example, if you save a workflow as a Finder plugin, then you can run the workflow by selecting a series of files, right-clicking your selection, and choosing to run the Automator plugin directly from the shortcut menu.

You're not limited to this kind of interaction, of course. You can create workflows that integrate directly with other applications (such as iCal), or run when certain triggers occur (like when you add a file to a folder), or as options on the print menu.

As you work through the projects in this book, you'll get the opportunity to work with plugins. For right now, though, you can save your workflow as a plugin by following these steps:

1. **Select File➪Save as Plug-in from the menu bar, or press ⌘+Option+S.**
2. **Enter a name for your plugin in the field provided.**
3. **Select what application your plugin will work with.** For now, choose Finder, but in Part III of the book you'll be creating other types of plugins.
4. **Click Save.**

FIGURE 3.25

Saving as a plugin

Once you've created a plugin, you can access it in different ways. For example, Finder plugins appear when you right-click a file or folder and then choose More➪Automator from the shortcut menu. Once you do that, you should see all the plugins you've created (see Figure 3.26).

You might have also noticed that accessing the contextual menu is a good way to start a workflow. In Part III, I'll ask you to create more than your fair share of workflows by selecting files in the Finder and using the contextual menu. It'll become second nature to you.

Note

Plugins are for Leopard only. In Snow Leopard, they are known as Services and have a slightly different configuration.

FIGURE 3.26

Accessing Finder plugins

What Are Templates?

Remember back when the chapter started and I mentioned something called *templates*? Templates are what you see when you first open Automator (in Leopard they were known as Starting Points). A template is just a convenient way to create a kind of workflow (see Figure 3.27).

FIGURE 3.27

Templates can help you get started quickly.

Templates were new to Automator in Snow Leopard. To get a workflow started, simply select a template that matches the kind of work you want to do. For example, to create an iCal alarm, start with an iCal Alarm template.

Summary

In this chapter, you've learned the following basics of Automator:

- How to start Automator
- How to make sense of the interface
- The difference between actions and variables
- How to view logs
- What actions are and how to chain inputs and outputs
- How to save your workflows
- What templates are

At this point, you know the basics but not much else. You should be able to create just about any basic workflow with the knowledge you've gained from Chapters 1, 2, and 3.

CHAPTER

4

Advanced Automator Topics

In Chapter 3, you learned the basics of the Automator interface and the basics of workflows and actions. You learned how to put together some basic workflows and how to use the Automator interface to your advantage. In this chapter, you're going to learn a little bit more about some advanced Automator concepts. I'll be introducing you to variables, loops, plugins, scheduling workflows, recording manual events, and setting up watch folders.

The goal of this chapter is simple: By the time you're done reading it, you'll know just about everything you need to know to create any workflow you want. After this chapter, you'll be spending quite a bit of time learning how to work with AppleScript. After you've mastered that skill, you'll be working on 50 automation projects in Part III. In just about every project in that section of the book, you'll get a chance to look at a specific automation problem, apply some basic Automator skills, add AppleScript to the mix (if needed), and really customize a solution. Right now, though, it's time to learn some advanced tricks in Automator.

One quick note: If you are using Tiger (Mac OS X 10.4), then most of the stuff in this chapter won't work unless you upgrade to Leopard (Mac OS X 10.5) or Snow Leopard (Mac OS X 10.6). Variables, loops, and manual recording only work in Leopard or higher!

IN THIS CHAPTER

- **Working with variables**
- **Working with loops**
- **A closer look at plugins**
- **Recording manual events**

Working with Variables

In the Quick Start chapter, you got the chance to add a very simple variable to a workflow, and then used that variable to hold the name of a folder. In this section, you're going to learn a bit more about variables and how to use them. Although you may not use them all that much, variables can be a powerful tool for helping you automate tasks.

What is a variable?

The simplest way to define a variable is that it is a container that holds a bit of data. If you're a programmer, then you're pretty used to the concept of a variable. One thing you don't have to worry about, though, is doing anything special with a variable before using it. For example, you don't have to say, "This variable will hold a date" or "This variable will hold an integer" or "This variable will hold a file path." For the most part, all of this is taken care of when you create a variable — the Computer Name variable most likely holds an alphanumeric string, whereas the Movies variable holds a file path to where your movies are stored.

What can you do with variables besides storing data in them and retrieving data from them? Not much, really. That's the point. You can ask a user to enter some information, capture it in a variable, and then reuse that same data throughout your workflow. For example, you might ask a user to enter their name, capture that data in a variable called username, and then use that variable to create a folder on the desktop (see Figure 4.1).

Variables can be used in other contexts as well — they don't just have to be something that a user types in a dialog. For example, you could capture a filename from a process, use a system-generated variable (like today's date) in different contexts, and so on.

If you have any doubt, just think of variables like this: They only have values once the workflow runs. They're not hard-coded. For example, you could create a workflow that sends an e-mail every single morning to your mother or father saying hello. Instead of hard-coding the date each and every day, you could simply use the variable called Today's Date. There, I just simplified the process of e-mailing your mom.

How much data can fit inside an Automator variable? It varies, but don't expect to store a lot of data. I'm talking about a single IP address, a username, a folder name, or quick responses from the user (like their name, their favorite color, or what have you).

FIGURE 4.1

Using a variable to capture a user's name

Can the values associated with a variable change once you've set them? Absolutely not. Once you set a variable, it stays that way until the workflow ends. In Figure 4.2, I've added a second round of clarification, seeking information from the user. I ask for their name twice before I set up the folder on the desktop.

At first, I tell the workflow that my name is Billy. Then I change my mind and say my name is Bill. As you can see by viewing the value of the username variable in the list, the value remains as `Billy`.

This is a good thing to know and will keep you from getting too frustrated. Don't worry, though, as you'll only be using variables in your most advanced workflows, and you'll likely not have to worry about changing values. However, if you have very complex workflows, or workflows that call other workflows, you might end up with variables with the same names.

FIGURE 4.2

Variables don't change once you set them in a workflow.

Once you start getting into the more complex areas of use, you might want to settle on some kind of naming convention for your variables that avoids this problem. A good prefix or suffix on variable names will keep you organized.

Types of variables

Like actions, variables are organized into a number of categories (see Figure 4.3), namely:

- Date & Time
- Locations

- System
- Text & Data
- User
- Utilities

FIGURE 4.3

Variables are organized into categories.

You're not limited to this list. Remember, you can create your own custom variables like I did with the username variable in the previous section's example.

Date & Time variables

Date & Time variables allow you to add such information as the current time, today's date, the current month, or the current year to your workflow. For example, you could create a new folder with today's date in the folder name. To do that, simply drag the Today's date variable to the New Folder action (see Figure 4.4).

FIGURE 4.4

Using the Today's date variable to create a new folder

Locations variables

Locations variables let you easily reference common folders on your Mac, such as your user's Home folder, the Pictures folder, the Sites folder, and so on. There's also a New Path variable that lets you establish a new location mapped to the folder of your choosing. In Figure 4.5, I've dragged the Music variable into the Start At field of the Ask for Finder Items action. Users can select music files and folders from that directory, which are then fed to the Create Archive action.

FIGURE 4.5

Using a Locations variable to specify a starting point for archiving

System variables

The System variables allow you to retrieve current system information and use that information in your workflows. For example, you may want to know the system's current uptime, its IP address, operating system version, or computer name — all of these are available as variables. In Figure 4.6, for example, I use the IP address and Operating system version variables, along with the Today's date variable, to name a text file dynamically.

FIGURE 4.6

Using System variables in a workflow

Text & Data variables

You can use Text & Data variables to store and retrieve text values, file and folder paths, iPhoto items, e-mail messages, and other items. In most cases, you'll find yourself using these variables whenever you want to store and retrieve action results.

In the sequence starting with Figure 4.7, I add a New Text variable to a workflow by double-clicking it. It appears in the Variable list at the bottom of the right pane.

FIGURE 4.7

Adding a New Text variable to the list of variables

Next, I double-click the New Text variable again and give the variable a new name and a value (see Figure 4.8). As you can see, I've made the variable name `FavoriteColor` and the value `Blue`.

Next, I drag the variable into my workflow, where it becomes a Get Value of Variable action. I then add a New Text File action to the end of that action to get a simple workflow that will place my favorite color (`Blue`) into a text file. Figure 4.9 shows the workflow action steps.

FIGURE 4.8

Setting a new name and a value for the New Text variable

All you have to do is check the Desktop for a `test.txt` file and open it to confirm that Automator placed the text `Blue` (or whatever your favorite color happens to be) in that file (see Figure 4.10).

FIGURE 4.9

Using the FavoriteColor variable in a workflow

FIGURE 4.10

The results of my simple workflow

User variables

User variables allow you to retrieve information about the current user: information like their first name, last name, homepage URL, e-mail address, and so on. In Figure 4.11, for example, I use the Short user name variable to pull out `myerman` (that's what my friends call me!) and put that text into the subject line of a new e-mail address.

FIGURE 4.11

Using the Short user name variable to populate a subject line

Utilities variables

The Utilities variables allow you to create random numbers and identifiers, as well as AppleScript and UNIX Shell Script variables. You can use the random numbers, for example, to keep filenames unique (as I do in the simple workflow in Figure 4.12 that takes a screenshot and then creates an image slide from it in Keynote). You can use the Shell Script variable to help execute code at various points in a workflow (see Figure 4.13).

FIGURE 4.12

Seeding a filename with a random number to ensure uniqueness

FIGURE 4.13

Using the Shell Script variable to execute code in your workflow

Adding variables to workflows

The best way to add a variable to your workflow is to either search or browse for the variable you need, and then double-click to add it to the list of available variables. You can also right-click inside the variable list and choose New variable from the shortcut menu (Figure 4.14).

FIGURE 4.14

Using the shortcut menu to add a new variable

Taking this route presents you with a Variable Options dialog similar to the one used for the New Text variable (see Figure 4.8).

Just remember that newly added variables don't have a value associated with them until you run a workflow. For example, as shown in Figure 4.15, the My name variable has no value until the workflow runs.

FIGURE 4.15

The value of a variable fills in only after a workflow runs.

Using variables in a workflow

Once you have a variable, you can simply drag it to the workflow (thus creating a Get Value of Variable action, as shown in Figure 4.15) or you can drag it to an Action element, such as a pop-up menu (folder location, for example) or text field (filename, for example).

In Figure 4.15, you can see how a variable name (in this case, `My name`) resides in a pop-up menu. Earlier in this chapter, in Figure 4.11, you saw how a variable could be pulled into the subject line of an e-mail.

All in all, variables offer a powerful and easy way to add more dynamic flexibility to your workflows. Instead of hard-coding folder names or other elements, you can use variables to help you work smarter.

Working with Loops

So far, I've only described one type of workflow: the one that runs through a series of actions once, and then stops. Sometimes, you need a workflow that repeats a series of tasks over and over. For example, you might want to process a series of files in one folder, then move on to another folder, and so on. Or you might want to save data to a series of log files.

Although you could create scheduled workflows to do the same thing, sometimes a loop simplifies the process for you — not only keeping your workflows more concise but also more manageable. Loops also give you some other benefits: They allow you to specify how many times (or for how many minutes) a loop should run.

What is a loop?

If you're a programmer, then you should be pretty familiar with a loop. A *loop* is simply a logical construct that says, "Repeat a series of tasks X number of times." If you're a shell scripter or Web developer, then you're used to `while` loops, `for` loops, and `foreach` loops.

In some ways, Automator loops are similar to these, except for an important distinction. Don't think, for example, that you can feed 500 images to a loop and have it work the way you might expect. An Automator loop doesn't open each file in the list of 500 images and process them one at a time — no, instead, it opens all 500 images at once, processes them at once, and so on. When you stick this kind of task in a loop, each time through the loop, it processes the 500 images.

In other words: nightmare. Imagine how many CPU resources it takes to do this operation — opening 500 images, processing 500 images, saving your work, then repeating again and again until the loop ends. If you've got 500 very small images, you might get lucky. If you've got 500 medium or large images, then you're going to run into trouble.

The proper way to use an Automator loop is to think in terms of either periodic updates or doing something a set number of times. For example, instructing your iSight camera to take a series of pictures and saving those pictures to a folder is a perfect use for a loop. So is searching for a random photo and copying it to a directory on your desktop, and then repeating that process a set number of times, or every few minutes.

Another great use for a loop is to do something in iterative stages. Loops in Automator can either work off the original data you started with, or continue working with modified data. For example,

if you told it to process 10 images by scaling them down 25 percent each time through the loop, and then looped through the files three times, you'd end up with successively smaller images. You'd have your original set at 100 percent, a second set 25 percent smaller than the first set, a third set 25 percent smaller than the second, and a fourth set 25 percent smaller than the third.

The best way to explain this is to walk you through creating a loop yourself.

Setting up a loop

Let's say that you're a real photography nut (or enthusiast, as we shutterbugs like to think of ourselves). You're constantly scouring Flickr to see what's new on the site. You have the URL for the latest photos bookmarked — it's `www.flickr.com/photos`. You go there constantly during the day, feeding your fix.

Well, it's time to use a loop to get some of your day back and continue to indulge your need for crowd-sourced photography. You're going to learn how to set up a simple workflow that pulls down all the images from a URL, and then imports those images into iPhoto for your later consumption.

Open up Safari and go to the URL indicated above. Make sure that you can also see an Automator workflow window. Simply drag the URL from the Safari window to the Automator workflow window. When you release the mouse, Automator automatically creates a Get Specified URLs action with the URL in question already filled in (see Figure 4.16).

Once you have the URL, add two more basic actions. The first is Get Image URLs from Webpage, which captures all the images you see. The second is Download URLs, which requires you to specify a target destination. I've created a folder titled `interesting` in my Photos folder, but you can create any folder anywhere; it doesn't matter. Figure 4.17 provides an illustration of where you're at so far.

FIGURE 4.16

Dragging a URL into Automator creates a Get Specified URLs action.

FIGURE 4.17

Once you have the URL, actually downloading the images is no trouble at all.

Once you've pulled down all the images into your folder, use a Get Folder Contents action to grab the contents of that folder. Then simply insert an Import Files into iPhoto action to do the actual import. I've opted to create a new album each time this runs, but you could easily add the photos to an existing set. I've also opted to clean up after myself by deleting the files. Figure 4.18 provides an illustration of where you should be so far.

FIGURE 4.18

Getting folder contents and importing into iPhoto

Finally, you're ready to add the looping mechanism. However, you need to implement a delay of some kind; otherwise, your loop will just keep repeating. No good! You want to have some kind of delay, like 15 minutes or so. Add a Run AppleScript action and place a delay 900 command inside the template provided by Automator (see Figure 4.19). The delay 900 command basically tells the workflow to pause for 15 minutes before continuing.

The final step is to add a Loop action — you can find it in the Utilities group, or by searching for loop. Loops can run automatically or sit and wait for user intervention or permission. In this case, you want the loop to just run automatically — there's no need to wait around for someone to click a mouse button. Furthermore, you can also tell loops to run a specific number of times or for a certain number of minutes — in my case, I want this loop to run 10 times. Because it's running every 15 minutes, that gives me a solid 3 to 4 hours worth of pictures from Flickr.

Finally, I want to opt to use the original input only. There's no point in reprocessing what's there.

FIGURE 4.19

Using AppleScript to delay the workflow before looping back up to the beginning

As you can see from Figure 4.20, the first time the workflow ran, it successfully pulled in all the images (including some GIFs that were really more for navigation on Flickr) into iPhoto, creating an album called `flickr newest`. All is good with the world, as the workflow will loop every 15 minutes, ensuring that I stay on top of the latest photos posted to Flickr.

FIGURE 4.20

The workflow successfully grabbed the latest images from Flickr.

A Closer Look at Plugins

In Chapter 3, I mentioned that you had various options for saving your workflows — you could opt to save them as workflows (and therefore run them inside Automator), as applications (allowing you to run them independently), or as plugins.

In this section, you're going to learn about plugins. The first question running through your mind is probably, "Well, what's a plugin?" A *plugin* is just a simple way to share what you've done in Automator with a completely different application.

For example, you may create a very useful workflow that creates thumbnails out of any list of images you designate. You could run this workflow inside Automator, or create an application out of it so that you can drag and drop whatever images you want onto it for processing. Similarly, you could also create a Finder plugin, making your workflow available whenever a user right-clicks a list of images in the Finder.

In this section, I'll only briefly cover the possibilities, as you'll get plenty of hands-on interaction with each of these in Part III.

Finder plugins

A workflow saved as a Finder plugin can be found in the Finder's contextual or shortcut menu, allowing users to take advantage of the workflow from anywhere in the Finder. For example, you

might want to process a list of files directly from the Finder instead of firing up Automator each time. Save your workflow as a Finder plugin. When you need to use it, simply select the files you want to process, right-click, and select More ➪ Automator ➪ *Your Workflow* (substituting the name of your workflow, of course). In Figure 4.21, you can see this process in action with a workflow I created for processing photos into a contact sheet.

All the files that I've selected for this particular workflow are passed into the first action as input. From there, the workflow continues as intended.

FIGURE 4.21

Processing files directly from the Finder using a Finder plugin

Calendar plugins

You can save your workflows as iCal Alarm plugins. When you do that from Automator, iCal automatically opens and you get the option to configure your workflow like any other kind of event — you can repeat it daily or weekly, you can add alarms to it, and so on.

For example, in the previous section, you created a workflow that would download the latest Flickr photos every 15 minutes. You could save this workflow as an iCal plugin by choosing File ➪ Save as Plugin from the menu, giving it a unique name (like `get-flickr-photos`), and choosing iCal Alarm from the pop-up menu (see Figure 4.22).

Note

Plugins are for Leopard only. In Snow Leopard, they are known as Services and have a slightly different configuration.

FIGURE 4.22

Saving a workflow as an iCal plugin

As soon as you save your plugin, iCal opens and creates an event that matches the name you gave the plugin. In this case, the event is called `get-flickr-photos` (see Figure 4.23). To configure this event, click Edit and treat it as you would any other event.

FIGURE 4.23

The iCal plugin as it appears in iCal

For example, you could set it to start up every day at 7:15am to make sure you start your day off right with new Flickr images.

Other plugins

In Part III, you're going to get a lot of additional hands-on experience with Image Capture, Print, and Script plugins. Briefly, though, here they are:

- **The Image Capture plugin option allows you to extend the Image Capture application that's built in to your Mac.** This can be very useful if you import a lot of photos from a digital camera. The possibilities here are pretty endless — any workflow that crops, color syncs, or processes images in any way can be added to the Image Capture contextual menu.
- **The Print plugin option allows you to extend the capabilities of the Print dialog.** You can already send documents to a printer or convert them to PDF using the PDF menu button in the Print dialog. Using Automator, you can create other, more useful possibilities as well, like being able to add PDF metadata when you convert to PDF.
- **The Script plugin option allows you to add your Automator workflows to a system-wide menu of scripts.** This menu is not visible by default. To make it visible, you need to enable it using the AppleScript Utility application. Once you have enabled the Scripts menu, it appears near the upper-right corner of your Mac's screen (see Figure 4.24).

FIGURE 4.24

The Scripts menu

Recording Manual Events

Sometimes, using loops, variables, shell scripts, AppleScript, and all the other possible options isn't enough. Either you don't have the time to figure something out, or you just need to walk through it once to see how it might be turned into a workflow. Or maybe you're just not the type to do things by the book (even this book). Instead, you want to just play around a bit, doing tasks in your normal way and having Automator set them all up correctly.

Don't despair, because the development minds behind Automator have thought of just about everything. Starting with the Leopard release of Automator, you can manually record workflows by having Automator watch what you do and record your mouse clicks and other activities.

First things first: Set up accessibility

Before you can have Automator record your actions, you have to set up Accessibility options on your Mac. To enable accessibility, you must first be an administrator on your Mac. Once you've made sure you have those privileges, follow these steps:

1. **Choose System Preferences from the Apple menu or click its Dock icon.**
2. **In the Systems Preferences window, click the Universal Access icon (see Figure 4.25).**

FIGURE 4.25

Setting up Universal Access in the Systems Preferences window

3. **Select the Enable access for assistive devices checkbox (see Figure 4.26).**

FIGURE 4.26

Check the assistive devices checkbox.

Recording a manual event

To start recording a manual event, you can start with either an empty workflow or a workflow that already has actions in it. Either way, you don't have to actually insert a new action from the Action library. Simply choose Workflow➪Record from the Automator menu, press ⌘+Option+R on the keyboard, or click the Record button in the Automator toolbar (see Figure 4.27).

FIGURE 4.27

Using the Record button

Once you click Record, you see a small window that displays a message indicating that recording is happening. You can now start working with your desired application (such as a text editor, the Finder, or another tool) and start doing things. While you work, Automator watches what you're doing, keeping track of keystrokes and mouse clicks.

Keep in mind that while you work, the application you're recording must support accessibility on the Mac. Even so, sometimes Automator doesn't record every single action you take. Furthermore, Automator may also record some events that you don't want or need in your workflow — you're free to delete them later.

Tip

Remember that Automator is watching what you're doing, so don't hurry through your tasks. Pretend that you're showing a task to someone else for the first time.

To stop a recording at any time, click the Stop button provided in the Automator record window, or choose Workflow➪Stop from the menu, or press ⌘+. on the keyboard.

Let's record a simple series of tasks (finding files in the Finder and opening them in a text editor). Here it is, step by step:

1. **First, start with a blank workflow in Automator.** (This is optional, of course, but easier for discussion purposes.)
2. **Click Record and wait for the window to appear that indicates that Automator is in record mode.**
3. **Open a Finder window and type** .txt **in the Search field.**
4. **Make sure that you're in your Home directory and that you are searching for filenames.**
5. **Sort the columns by names ascending, putting the list in alphabetical order.**
6. **Select the first ten files or so, right-click, and choose Open With➪TextEdit (see Figure 4.28).**

FIGURE 4.28

Selecting files in the Finder

7. **The workflow is now complete; click Stop.** You should now have a Watch Me Do action in the workflow with all the steps you've recorded (see Figure 4.29).

FIGURE 4.29

The Watch Me Do action holds a record of what you've done.

As you can see, this initial record isn't perfect — there appear to be some repetitions. Notice that if you click any event in the Watch Me Do action, you get different details in the right pane. Did you also notice the blank event? That's where I sorted the list by filename in ascending order. It'll be interesting to see if this actually works on playback. Also notice that my selection of the first 10 files named `_folderinfo.txt` was translated into three Finder clicks.

When I finished recording this action, I ended up with 10 text files open on my Mac. What happens when I play it back? Well, the first problem I ran into was the fact that I was auto-hiding my Dock. Automator simply couldn't find it at all, so I turned off auto-hiding.

Once it could find the Dock, Automator performed fairly well. It opened a Finder window, did a search in the search field, clicked File Name, sorted the filename column properly, and then selected the first 10 files as I asked. The first time, it failed because it couldn't figure out what I'd done with TextEdit; as a result, I went back and made sure that TextEdit was launched and then ran my workflow again. It still didn't work.

I tried again, this time dragging my selected files to the TextEdit icon on my Dock. Still no luck. The third time was the charm, though: I re-recorded the task, this time only asking for selected .txt files. Then I added an Open Finder Items action (see Figure 4.30). That seemed to make it work.

What's the takeaway on all this? First of all, the recording feature isn't perfect. It won't be able to match everything you do, so be patient with it. However, it can be a very good teaching tool to keep you moving forward with your own projects. Very soon you'll find yourself just building stuff from scratch because your recorded workflows will be too cumbersome.

FIGURE 4.30

Adding an Open Finder Items action

Summary

In this chapter, you've learned some advanced tips and tricks related to Automator:

- How to work with variables
- What loops are and how to use them
- How to create basic plugins
- How to record manual events

At this point, you know just about all you need to know when it comes to Automator. It's time to learn AppleScript.

Part II

A Detailed Look at AppleScript

IN THIS PART

Chapter 5
AppleScript Basics

Chapter 6
AppleScript Objects and Dictionaries

Chapter 7
Working with Variables and Properties

Chapter 8
Operators, Expressions, and Statements

Chapter 9
Conditionals and Loops

Chapter 10
Handling User Input

Chapter 11
AppleScript Subroutines

Chapter 12
Applets and Droplets

Chapter 13
Folder Actions

Chapter 14
AppleScript Studio

CHAPTER

5

AppleScript Basics

Now that you've had a thorough introduction to Automator, the next ten chapters will introduce you to AppleScript. By the time you're done with these chapters, you'll be able to create just about any script you need for just about any situation you might encounter.

This knowledge of AppleScript will also come in handy as you continue to work with Automator. The projects in Part III will involve some tweaking of workflows using AppleScript. In some cases, as your expertise grows, you might find it easier or faster to simply create automation scripts directly in AppleScript instead of working with Automator. Or you may use both Automator and AppleScript in conjunction.

IN THIS CHAPTER

- **What is AppleScript?**
- **A brief history of AppleScript**
- **Why AppleScript?**
- **The AppleScript Editor menus**
- **The Script menu**

What Is AppleScript?

AppleScript is a scripting language built into Mac OS X. If you're looking for some kind of parallel to other systems, it's much like shell scripting in UNIX or Windows Batch programming, although in many respects it is different from both of these.

Here are the essentials that you need to know about AppleScript:

- Starting with AppleScript 2.0, the character set is Unicode.
- You can only use upper- and lowercase letters, numbers, and the underscore (_) character when you name your variables, objects, and other items. All of these identifiers must also start with a letter to be valid.
- AppleScript is not case-sensitive; myVariable is equivalent to myvariable and myVaRiAble.

- The basic data types supported in AppleScript are Boolean, text, number, list, record, and object.
  - Booleans can be either true or false.
  - Text can contain alphanumeric characters, as well as various other characters
  - Numbers can contain positive and negative whole numbers (for example, `0`, `-2`, `200`), floating point numbers (for example, `3.14`, 0.773353), and exponential notation (for example, `9.3E+10`).
  - Lists can contain different data types, including other nested lists (for example, `{"Beethoven", 9, -3.15353, true}`).
  - Records are lists of unordered key-value pairs (for example, `{product:"pencil", price:0.99, color:"yellow"}`).
  - Objects are instantiations of a class definition, including built-in AppleScript objects, Mac OS X objects (for any scriptable part of Mac OS X, such as the Finder), Application objects for third-party objects, and objects that you define yourself.
- AppleScript offers standard control flow with `if/else` constructions and `repeat` loops (`for`, `while`, `in`).
- Variables are instantiated when you use them, and they're not strictly typed.
- Script objects can have methods and data, and can inherit behavior from parent scripts.
- You can use the `tell` construct to identify targets of your scripts.

In the next ten chapters, you'll learn a lot about each of these items, with an emphasis on a hands-on approach.

AppleScript is designed as an end-user scripting language, offering an easy and intuitive way for Mac users to control applications, automate tasks, and access and modify information without having to actually work through the graphical user interface (GUI).

The main use of AppleScript is to automate workflows, thus reducing the time it takes to perform certain tasks, reducing the number of errors, and increasing the consistency of both input and output. AppleScript does this by taking advantage of the fact that most Mac programs tend to use standard Apple Events to communicate with the underlying operating system — you can think of AppleScript as the glue that binds applications, data, and the operating system together.

A Brief History of AppleScript

The AppleScript project was a direct result of the HyperCard project. HyperCard was a hypermedia authoring environment that predated the World Wide Web. It featured an English language-based scripting language called HyperTalk, which HyperCard programmers used to embed logic and behavior within a HyperCard stack.

A HyperCard stack, by the way, was very much like a collection of Web pages, except that it all ran as part of a desktop application, like the one shown in Figure 5.1. You could create links and program buttons to perform certain actions. In many ways, HyperCard was an important forerunner of all the hypermedia projects and tools that followed, including CD-ROMs and the Web. In fact, the only thing that kept HyperCard from becoming the first Web browser was its focus on local stacks of cards, as opposed to network-oriented stacks.

FIGURE 5.1

Beethoven's Ninth is an example of a HyperCard stack.

Because HyperTalk (an example page from the Quick Reference Guide is shown in Figure 5.2) was deemed generally useful as a working part of any application, Apple engineers decided to spawn a new project, AppleScript, and make it part of the System 7 release. The first release of AppleScript was in 1993, as part of the System 7.1.1 (System 7 Pro) release. One of the first major software applications to support AppleScript was QuarkXPress, which gave the programming language a lot of credibility in the publishing and prepress world.

FIGURE 5.2

A page from the HyperTalk Reference Guide

The move to Mac OS X and its Cocoa frameworks has given AppleScript a much-needed boost, as AppleScripts can now work with Cocoa applications. The addition of AppleScript Studio and the availability of an improved Script Editor make creating, debugging, maintaining, and deploying AppleScripts even easier.

Why AppleScript?

If automation is the name of the game (at least as far as this book is concerned) and you know something about Automator, then why all the fuss about AppleScript? What's the point, you may ask?

Well, Automator will get you pretty far. In fact, I'm willing to bet that you could probably use nothing but Automator for most of your tasks. Eventually, though, you're going to run into a situation where you can't simply drag and drop an Action onto your Workflow. You'll find that you need to create a script, and that by creating a script, you create a reasonable, concise solution to a specific task or problem.

You may also find yourself in a situation where you need your automation to make decisions based on output — in other words, that you'll need support for conditional branching. Or that you'll need much finer control over looping, and variables, and many other domains that come through actual scripting. Or you'll need to take control of an application's user interface, or for that matter, start working with an application that isn't fully compatible with Automator.

Another thing to consider is your learning style. You may be highly visual and so find a lot of comfort in the visual metaphor offered by Automator. Or you may be highly verbal, and find the most comfort in using a scripting language. Or you'll come from another scripting environment, such as UNIX shell scripting or Perl, and you just feel better about task automation when it involves scripts.

I'm not trying to say that you'll always use one tool versus another. What you'll find very quickly is that for some tasks, it's a lot more intuitive (and faster) to create an Automator workflow, save it as a Finder plug-in, and use it as part of a right-click or contextual menu. In other cases, it'll just be easier for you to create a concise five-line AppleScript to do what you need. And in yet other cases, you'll probably find yourself using those little AppleScript gems in your Automator workflows.

The goal of this book is to give you an introduction to all of the options available to you, and then stand back and let you decide on the solutions that meet your needs.

AppleScript Editor

In the Quick Start chapter, you received a very quick introduction to AppleScript Editor. In this section, I'll slow down a bit and give you a much closer look. It's worth taking the time to learn something about this tool, as you'll be spending most (if not all) of your AppleScripting time here. Yes, it is possible to create and maintain AppleScripts with TextEdit (or other text-editing tools),

and yes, there are other Integrated Development Environments (IDEs) available that do a fine job, but for the most part, AppleScript Editor is all that you need.

Please note that in Leopard, the AppleScript Editor was called Script Editor. It was renamed in Snow Leopard and moved under Applications ➪ Utilities.

Starting AppleScript Editor

You can find Script Editor under Applications/Utilities/AppleScript Editor. Do yourself a favor, if you haven't already done this: as soon as you open AppleScript Editor, right-click (or Ctrl+click) the AppleScript Editor icon in the Dock and choose Keep in Dock; that way, you always have it at your fingertips (Figure 5.3).

FIGURE 5.3

Keep AppleScript Editor in the Dock for easy access.

The AppleScript Editor GUI

AppleScript Editor should launch as soon as you call it up. It's a very simple interface, as you can see in Figure 5.4.

The interface is broken up into five main areas:

- The top toolbar contains some important buttons, such as Record, Stop, Run, and Compile. You'll learn more about these as you go.
- Next comes a windowpane that you'll use to enter your code.
- Below this first windowpane is an array of buttons that control what you see in the second windowpane: events, replies, or results. Result is selected by default
- The next pane can show different kinds of information, such as the results of your code execution, a description of a particular item, or a log of events in your script.
- The final piece of the interface is a footer that allows you to select what either a description or event log.

FIGURE 5.4

The AppleScript Editor GUI

Basic operations

Generally speaking, whenever you write a script, you simply open AppleScript Editor, write your script in the top pane, click the Run button, and see the results in the bottom pane. For example, Figure 5.5 shows what happens when you write and run a very simple script: this example is asking for the version of the Finder application. The answer you get back is very simple: 10.6, which also happens to be the version of Mac OS X I'm running.

Comments and error messages

Many of the scripts you'll write will be very simple, but some will be very complex. Either way, AppleScript Editor offers a way for you to enter comments (so you can document your code) and error messages (to tell you when things have gone wrong).

To enter a comment, simply start a line with two dashes (--). You can create multi-line comments (for more extensive documentation) by using `"(*"` at the beginning and `"*)"` at the end of your comment block. Both options are shown in Figure 5.6.

FIGURE 5.5

Running a very simple AppleScript in AppleScript Editor

FIGURE 5.6

Adding single- and multi-line comments to AppleScripts

If you do something wrong, such as misspell a command or make some other kind of mistake, AppleScript provides a helpful error dialog to let you know what happened.

For example, in Figure 5.7, I've changed the `version` command to `versions`. AppleScript doesn't know what I'm talking about, so it stops the script immediately, shows an error message, and highlights the part of the code that it finds problematic.

FIGURE 5.7

The AppleScript Error dialog

Saving your scripts

Once you've perfected a script, you'll want to save it so you can use it over and over again. Although my little script that prints out the version of Finder isn't particularly useful, I'm going to save it anyway (see Figure 5.8).

To save a script, you can select File➪Save from the menu or press ⌘+S on the keyboard. By default, AppleScript Editor prompts you to enter a name for your AppleScript and offers a default location for the script (in ~/Library/Scripts/Finder Scripts). You can save your script to this location or choose some other location, such as your desktop or in your documents folder.

FIGURE 5.8

Saving your AppleScript

You also get a default file type of Script, which means not only that you have a working script, but that you can also edit it again. However, in order to run your script, you have to run it from AppleScript Editor. You can also choose to save your work as an Application (an Applet that you can double-click), as a Text File (in case you can't get it to compile but need to save anyway), and in other ways, such as droplets. As you learn more about AppleScript in these upcoming chapters, you'll explore the different options available to you.

For right now, you'll be saving your work as a Script.

The AppleScript Editor Menus

There are quite a few options available to you through the AppleScript Editor menus. In this section, I'll go over a few of the more important ones. The top-level menu items, shown in Figure 5.9, are as follows:

- AppleScript Editor
- File
- Edit
- View
- Script

- Font
- Format
- Window
- Help

FIGURE 5.9

The AppleScript Editor top-level menu choices

AppleScript Editor menu

The two most important choices on the AppleScript Editor menu are:

- AppleScript Editor➪Quit AppleScript Editor, which allows you to quit working with the application (⌘+Q does the same thing); and
- AppleScript Editor➪Preferences, which opens the Preferences dialog (or press ⌘+,).

The Preferences dialog is something you probably won't get to at first, but it's worth exploring right away. The dialog consists of five tabs:

- **General**. This is where you set your default language (almost always AppleScript) and set preferences related to the Script menu. Figure 5.10 shows this screen.

FIGURE 5.10

Select Preferences➪General to show this screen.

- **Editing.** This is where you tell AppleScript Editor whether to wrap lines, set your tab widths, and use Script Assistant (Figure 5.11). Brand new in Snow Leopard is a Develop section where you can enable/disable the "tell" application popup menu.

FIGURE 5.11

Select Preferences ➪ Editing to show this screen.

- **Formatting.** This is where you set your fonts, sizes, and colors for different options. For example, by default, comments are shown in gray Verdana 12 pt, but you might want another configuration (Figure 5.12). Don't worry about changing things, as you can always revert to the default settings.

FIGURE 5.12

Select Preferences ➪ Formatting to show this screen.

- **History.** This is where you set your Event Log history (Figure 5.13). Although this doesn't make much sense to you now, to be a good debugger, you have to have some way of looking over your previous runs. You can set either to unlimited entries or to some kind of maximum, such as 10.

FIGURE 5.13

Select Preferences➪History to show this screen.

- **Plug-Ins**. This is where you manage third-party plug-ins for AppleScript Editor (Figure 5.14). Different ones are available on the Internet, although it's likely that your screen will look exactly like Figure 5.14: blank.

FIGURE 5.14

Select Preferences➪Plug-ins to show this screen.

File menu

The File menu is where you go when you need to open a script, close a script, or save your work. The most important menu items you need to know now are:

- File➪Open (or ⌘+O), which prompts you to select a script for editing;
- File➪Open Recent, which shows you a list of the last ten scripts you opened in AppleScript Editor;

- File ➪ Open Dictionary, which allows you to open a specific dictionary from a dialog listing all the available application dictionaries.
- File➪Close (or ⌘+W), which closes the file you currently have open;
- File➪Save (or ⌘+S), which saves the script you're working on; and
- File➪Print (or ⌘+P), which prints out the source code of the script you're working on.

Edit menu

The Edit menu is where you go when you need to copy, paste, undo some kind of action that you've taken (such as an accidental deletion), or find a string in your code. The most important menu items you need to know now are:

- Edit➪Undo (or ⌘+Z) to undo your last action;
- Edit➪Cut (or ⌘+X) to cut highlighted text from your script to the Clipboard;
- Edit➪Copy (or ⌘+C) to copy highlighted text in your script to the Clipboard;
- Edit➪Paste (or ⌘+V) to paste cut or copied text from the Clipboard back into your script;
- Edit➪Select All (or ⌘+A) to select all content in your script; and
- Edit➪Find ➪ Find (or ⌘+F) to search for a specific string.

View menu

The View menu helps you control what you see in AppleScript Editor. You won't be using it very often, but there are some important menu items that you may need to access from time to time, for example:

- View➪Show Description (or ⌘+1) to toggle the description mode in the bottom pane;
- View➪Show Result (or ⌘+2) to toggle the result mode in the bottom pane; and
- View➪Show Event Log (or ⌘+3) to toggle the event log mode in the bottom pane.

One more thing to note here: View➪Hide Toolbar allows you to hide the toolbar that runs along the top of AppleScript Editor, and View➪Show Toolbar makes it visible again.

Script menu

The Script menu gives you additional access to all those functions represented by buttons in the toolbar across the top. These menu items (and their associated keyboard shortcuts) can come in pretty handy, particularly if you hide the toolbar. Here they are:

- Script➪Record (or Shift+⌘+R) to record a script. You can record scripts with AppleScript Editor in much the same way as you can record with Automator. You'll learn more about recording in Part III, when you work on some projects.
- Script➪Run (or ⌘+R) to run a script.

- Script➪Stop (or ⌘+.) to stop a script that is currently running.
- Script➪Compile (or ⌘+K) to compile a script.

Font menu

The Font menu allows you to configure the fonts in your Scripts. You won't be doing much with fonts at first, but there are two very useful menu commands to know, especially if, like me, you spend a lot of time at the computer and your eyes get tired:

- Font➪Bigger (or ⌘++) increases the font size of your scripts.
- Font➪Smaller (or ⌘+-) decreases the font size of your scripts (in which case, you're just a show-off).

Format menu

The Format menu allows you to configure the alignment of highlighted lines of code, for example:

- Format➪Align Left (or ⌘+{) aligns lines to the left.
- Format➪Align Right (or @cmd+}) aligns lines to the right.
- Format➪Center (or ⌘+|) centers the lines.

Window menu

The Window menu helps you manage your window environment. Right now, you won't be using it much, but if you have many windows open, it's good to know that all your script windows are listed at the bottom, making it easier for you to switch from one to the other (Figure 5.15).

FIGURE 5.15

The Window menu

Help menu

The Help menu contains a search interface that makes it easy to find commands in the menu hierarchy (Figure 5.16). It also contains three extremely useful menu items:

- Help➪AppleScript Editor Help, which opens a tutorial on AppleScript Editor.
- Help➪AppleScript Help, which opens the online AppleScript documentation (Figure 5.17). This online help guide allows you to browse or search for AppleScript-related information.
- Help➪Show AppleScript Language Guide, which opens a reference guide on AppleScript.

FIGURE 5.16

The Help Search interface

FIGURE 5.17

The AppleScript Help documentation

The Script Menu

In Chapter 4, I mentioned the Script menu, and showed you how to enable it (Figure 4.24). This isn't really a menu in the AppleScript Editor application; it's more like a globally available set of scripts on your Mac. Once enabled, it usually shows up as an icon on your Mac's top menu bar. Just click anywhere on the desktop to make that default menu bar appear.

Any script saved to /Library/Scripts or ~/Library/Scripts (in other words, inside your username's Scripts folder under the Library folder) should appear in this convenient location. One more note before leaving this topic: the scripts in this area aren't limited to AppleScripts, but can also include shell scripts, Perl scripts, or any other kind of script.

Summary

In this chapter, you got some background on AppleScript and learned about:

- the basic data types available to you as a programmer;
- what AppleScript Editor is;
- how to work with AppleScript Editor;
- what your options are for saving scripts; and
- what kinds of menu options are available to you.

CHAPTER

6

AppleScript Objects and Dictionaries

In Chapter 5, you learned the basics of the AppleScript environment — AppleScript Editor, the Script menu, and other basics of the tools involved. If it seemed like a lot to take in at first, don't worry; by the time you're done with this Bible, it'll all be second nature to you.

In this chapter, you're going to learn some extremely high-level concepts related to AppleScript. Specifically, you're going to learn about objects and dictionaries. Although this information won't seem that useful to you right now, after you've been working with AppleScript for a while, you'll realize that everything comes back to two things: objects and dictionaries.

So, it's time to begin. First, I'll get into objects.

IN THIS CHAPTER

- **What are objects?**
- **What are dictionaries?**
- **How to use dictionaries**
- **A quick tour of dictionaries**
- **Looking up an AppleScript**

What Are Objects?

Everything in AppleScript is an *object*. Period, end of story, move on to the next thing. Check, please!

Wait, too fast? Okay, I'll explain. First, let me quote from the *AppleScript Language Guide*, available at `developer.apple.com/documentation/applescript`:AppleScript is an object-oriented language. When you write, compile, and execute scripts, everything you work with is an object. An object is an instantiation of a class definition, which can include properties and actions. AppleScript defines classes for the objects you most commonly work with, starting with the top-level script object, which is the overall script you are working in.

Note

Please note that you will need to join the Apple Developer Connection to get access to this documentation. Signing up is free; you can do that at developer.apple.com.

If you write a simple script that plays a random song from iTunes, the playlist is an object, and so is the track. So is the command that actually plays that track. So is the iTunes application. For that matter, so is the entire script. Each object has different properties and actions that you can access. For example, to play a random song, you specify the `some` property of the track object.

Let's take that quote from the AppleScript Language Guide one bit at a time.

AppleScript is an object-oriented language

If you are coming from an object-oriented world, such as Python or object-oriented PHP, this part of AppleScript will be a cinch. If you have limited exposure to object orientation, then here's a quick primer on that world.

Each object in an object-oriented world is actually an instantiation of a *class*. A class is like a definition for an object, a master object if you will. Let's pretend that in your world, you wanted to have lots and lots of dogs. The way to do that in AppleScript would be to have some kind of Dog class. This Dog class would include some basic *properties* and *actions*.

What would be some properties for dogs? Well, for one thing, each dog would have fur, and four legs, and a tail, and ears, weight in pounds, and so on. Think of properties as nouns.

What would be some actions for dogs? They could run, sit, fetch, bark, eat, and so on. Think of actions as verbs.

Not all dogs are the same, of course, but they each share certain characteristics. If I wanted to make a small dog, like my Yorkshire terrier, Marlowe, I would instantiate a new Dog. This new Dog object would inherit all the properties and actions from the Dog class, but after that, I'd be free to make some changes to make her more Yorkie-like.

For example:

- Her weight would go down to 6 to 8 pounds.
- She would have a short tail, big ears, and silky fur.
- Her excitement and energy levels would be extremely high.
- Her need for food would be very low.
- She would be more prone to bark than the average dog.
- She would be more prone to play like a puppy even into her senior years.
- Her life expectancy would be longer than that of the average dog, 16 years or so.

Another dog would be entirely different from this Yorkie-like object, even another Yorkie. Each object is an individual that is cast from a mold known as a class. In this way, inheritance and individuality play an important role in making you a more productive, intuitive programmer.

Once you understand how classes and objects work, then you've gone a long way toward understanding exactly how AppleScript works.

For the most part, you'll be working with predefined objects, for example:

- AppleScript objects, such as scripts, text, numbers, and so on.
- Mac OS X objects, such as the Finder, System Events, and so on.
- Third-party application objects, which are defined by vendors to help you work with their software.

You won't just be working with predefined objects, however, because you'll also be able to create your own objects that contain the properties and actions that you need to get the job done. The most common type of object you'll be able to create is called a script, but more on that later.

Everything you work with is an object

It's worth stating again: Everything you work with in AppleScript is an object. If you set a variable, that variable is an object. If you're coming from the world of shell scripting or Perl, for example, you're probably thinking right now, "Nah, that can't be, a variable is just a container. It contains a simple value that I assign to it at some point in the script, then throw away."

Well, yes, that might be true, but in AppleScript, that simple variable is all that *and* an object. It just so happens that it's a simple object, but an object nonetheless.

Here's a list of objects you'll likely encounter on any given scripting day (check out Figure 6.1 too):

- Any applications you invoke
- Any windows you open inside those applications
- Any text fields inside those windows
- Any pop-up menus inside those windows
- Any keystrokes you invoke
- Any menu items in those applications
- Any text you select
- Any characters in text
- Any words in text
- Any items in a Finder listing (folders, files, aliases)

- Any smart folders in Finder
- Any tracks or playlists in iTunes
- Any messages in Mail
- Any mailboxes in Mail
- Any smart mailboxes in Mail
- Any accounts in Mail
- Any images in iPhoto
- Any events in iCal
- Any date
- Any person or group in Address Book

I could go on and on, but I think you get the idea.

FIGURE 6.1

All of these things are objects

Objects include properties and actions

I've already mentioned this quite a bit, but each object you work with has properties and actions. A *property* is a characteristic of an object with a single value and label. For example, a date will have a year property. Each property must be uniquely named within a class — you can't have two names for a window or two year properties for a date.

By the way, when you work with properties, it's also important to know if it's read/write or read only. Not knowing this can cause the beginning AppleScripter a lot of grief. You'll waste a lot of time trying to set a property, not realizing that it's read-only. One way to learn whether a property is read/write or read-only is to make use of the AppleScript dictionary — which I'll get to shortly.

An *action* is equivalent to a method in other object-oriented languages. In other words, an action allows you to do something with that object. If the object were a dog, it could bark, sleep, or play. Different AppleScript objects have different actions; for example, you can:

- **play** a track
- **show** an event
- **eject** a disk
- **sort** a list of files
- **add** a person's contact information
- **e-mail** contents of a Safari tab or window
- **check** for new e-mail
- **forward** e-mail
- **delete** (files, words, messages, and so on)
- **count** (files, words, lines, messages, characters, and so on)
- **print**

When I start talking about dictionaries shortly, you'll learn that some actions and properties are part of the Standard Suite (see Figure 6.2).

In other words, they are commonly shared across Mac OS X applications, while other actions and properties are locally defined by certain applications or class of applications, such as the database suite, spreadsheet suite, and so on. In most cases, particular applications have their own sets of unique actions and properties; however, occasionally you'll run across a situation in which a Standard Suite action or property has been redefined.

FIGURE 6.2

A sample of actions from the Mail dictionary

Even the script you're working with is an object

That's right, you read correctly. When you fire up AppleScript Editor and write a script, as simple as that script may be, you are working with a special kind of object called a *top-level script object*. It's known as a top-level script object because nothing at all can be above a script object — it contains all other objects.

What makes up a script? Well, to begin with, every script comes with an implicit `run` handler built into it. You don't have to add it because AppleScript understands it to be there even when it isn't. An implicit `run` handler just tells AppleScript, "Hey there, run this script, okay?"

For example, the following very simple script runs very well by itself:

```
beep
```

This is because AppleScript wraps the entire script with an implicit `run` handler that actually runs the script. Of course, you could add an explicit `run` handler as shown in Figure 6.3, like this:

```
on run
    beep
end run
```

Later on, when you're building applets and droplets, you'll find that it's a good idea to put in explicit `run` handlers and `open` handlers to keep things straight, but for right now, just know that the implicit `run` handler is your friend.

FIGURE 6.3

An explicit run handler: Beep, I say, Beep!

What else can you put inside a script object? Here's a list, most of which will be explained as you move through the chapters of this Bible:

- Property and variable definitions
- Additional run handlers, which are calls to subroutines (either built-in ones, or custom subroutines that you write yourself)
- Script objects (which I don't use; these are nonetheless important enough to discuss in the next subsection)

A quick tour of script objects

Script objects are objects that you define inside your scripts. Each script object is made up of properties (each of which holds some kind of data) and actions (such as custom handlers and even other nested script objects).

I don't use script objects very much in the main part of this Bible, but they're worth talking about because they offer you a great deal of flexibility. To illustrate, I'm going to create a very simple script object and then build on it.

Here's a very simple script object that does one thing — it returns the current date:

```
script mydates
    on sayToday()
        return current date
    end sayToday
end script
```

By using the `script` command, I've defined a script object by the name of `mydates`. This `mydates` script object contains one thing, a handler called `sayToday`. When run, the `sayToday` handler returns the current date.

To use this script object, I have to tell it to `sayToday`, like this:

```
tell mydates to sayToday()
```

When I run this script, I get the following returned value:

```
date "Saturday, July 4, 2009 7:11:41 AM"
```

I could add other subroutines to this basic script object, making it more useful. For example, I might want to return the current date as a string instead of an object reference. To do that, I would just add a new subroutine called `sayTodayString` and tell it to return the current date as type string:

```
script mydates
    on sayTodayString()
        return (current date) as string
    end sayTodayString
    on sayToday()
        return current date
    end sayToday
end script
```

I can then invoke it like this:

```
tell mydates to sayTodayString()
```

And I get back this:

```
"Saturday, July 4, 2009 7:46:04 AM"
```

The final script would look like Figure 6.4.

FIGURE 6.4

The final script object

What Are Dictionaries?

Dictionaries are repositories of AppleScript knowledge. What kind of knowledge? Well, everything you ever wanted to know about objects, properties, and actions. If you want to do something with AppleScript, then it is recorded in one of the dictionaries, probably the one associated with the application you're trying to work with. And just as you look up a word in Merriam-Webster to find out what it means, you look up commands in an AppleScript dictionary to find out what you can do with them.

If you can't find what you need in a dictionary, or if there isn't a dictionary for the application you're dealing with, it doesn't necessarily mean that you can't script your way through a problem. You can almost always use a special call to the System Events application to manipulate menus and simulate keystrokes — in fact, a few times in this book you'll have to do just that to make certain applications work for you.

How to Use Dictionaries

The easiest way to open a dictionary is to choose File➪Open Dictionary from the AppleScript Editor menu. You can also press ⌘+Shift+O (that's an "oh," not a zero).

When you do that, the system prompts you to choose an application from the list (see Figure 6.5). To open a dictionary, click the appropriate application and then click OK. If you're working with Address Book, then that's what you want to see, as shown in Figure 6.6.

FIGURE 6.5

Choosing a dictionary from a list

FIGURE 6.6

The Address Book dictionary

Another way to access dictionaries is via the Library window. To open this window (see Figure 6.7), choose Window➪Library from the AppleScript Editor menubar, or press ⌘+Shift+L on the keyboard. To open a dictionary, all you have to do is double-click the appropriate icon.

FIGURE 6.7

The Library window

Notice that the Library window has add and remove buttons on the top. These buttons allow you to add or remove dictionaries to and from the list. I had to do this the other day — I had just installed a brand-new application that I wanted to automate. It wasn't on the list when I checked the library, so I clicked the add button and selected it from the list of applications in the Applications folder. Within seconds, the Library added the dictionary that had been so helpfully provided by the developers of the application.

A Quick Tour of Dictionaries

Once you've got a dictionary open, what exactly do you do with it? Well, most people, when they first open a dictionary, find it a bit confusing. That's a natural reaction, as there's a lot of information in them and you have to know how they are structured or else it's easy to get lost.

In this section, I'm going to take you through a quick tour of the Safari dictionary. It's a fairly common (if not universal) application on the Mac, and it contains some pretty well-understood objects, properties, and actions that you should not have any trouble with.

The first step is to open the Safari dictionary using one of the methods described in the previous section. Once you've got it open, it'll look like Figure 6.8.

FIGURE 6.8

The Safari dictionary

Along the top of the Dictionary window are three panes that allow you to drill down into more and more detail.

- The leftmost pane contains a list of suites (which is the top-most level of organization within a dictionary, like chapters in a book).
- The second pane holds information about objects, actions, and properties of a selected suite.
- The third pane holds information regarding any objects selected in the second column.

The bottom pane is reserved for content. Whatever you've got selected in the top panes displays there. Figure 6.9 shows how I've navigated for specific content, in this case, the application object in the Standard Suite. As you can see, an application can contain windows and documents, and has three properties: frontmost, name, and version.

You now know how to access the name of a window — via the application object.

Here are some other important details that should jump out at you:

- The icon that looks like an S inside an orange square denotes a Suite.
- The icon that looks like a C inside a blue circle denotes a Command (that is, a verb).
- The icon that looks like a C inside a purple square denotes a Class (that is, a noun).
- The icon that looks like a P inside a purple square denotes a Property (that is, an adjective).

FIGURE 6.9

Navigating for content in the Safari dictionary

You can drag the horizontal divider between the top panes and bottom content pane, giving yourself more or less room as needed to either navigate or view content. What a lot of people miss, however, is that another draggable divider exists, a vertical one, over on the left side of the navigation pane. In Figure 6.10, I've dragged out that pane and marked both dividers for easier identification.

FIGURE 6.10

Resizing the dividers

This secondary navigation pane only shows you items by Suite, which is what you'll probably use most of the time. However, there are other ways to view a dictionary (specifically, by *containment* and *inheritance*), and you can easily switch views by clicking one of the three View buttons in the header (in Figure 6.11, you can find these buttons in between the Text Size and the Print buttons).

FIGURE 6.11

Switching views with the View buttons

Viewing by suite

Viewing commands by suite is probably the most intuitive way to look at them. Each application dictionary hooks into the Standard Suite, which is shared across all applications. These are commands and options such as delete, move, duplicate, print, quit, and so on. Also included in this suite are common object classes such as application, document, window, and so on.

Another suite you might see is the Text Suite (if the application in question does a lot of text handling). Usually, though, there is a suite devoted to the application in question (as in the Safari Suite) that forms the core of the functionality you're after.

In the case of Safari, the Safari Suite contains certain actions (such as `do JavaScript` and `show bookmarks`) and object classes (such as tab) that are unique to Safari and make it easier for you to automate the application. For example, if you want to view the source code of a document loaded in a tab, you'd access it via Safari Suite ➪ tab ➪ source (as shown in Figure 6.12).

Viewing by containment

The containment view shows you what objects can be contained inside other objects. This view is limited to object classes and can help you visualize how things are placed in the hierarchical AppleScript world.

Continuing with the Safari example, you can see that the top-most container is the application class, which can contain both documents and windows. A window, as you can see, is part of the Standard Suite (Figure 6.13).

Why would you want to even view information in this manner? Easy answer: Occasionally, you will need to know how to reference an object with its containment path. For example, you might want to know the name (that is, the title) of the first window:

```
tell application "Safari"
    get name of window 1
end tell
```

FIGURE 6.12

Viewing the source code

FIGURE 6.13

Viewing the Standard Suite in the navigation pane

Viewing by inheritance

Remember the little discussion about creating a Dog class and then instantiating different Dog objects based on that class? Remember how important inheritance was to object-oriented coding and how your individual Dog objects could inherit the properties and actions you put in the Dog class?

Well, the inheritance view in the dictionary allows you to view how any particular object sits in the inheritance tree. Go far enough back, and you'll always reach the item class — everything in AppleScript derives from this class. In and of itself, an *item* isn't very remarkable, as it is only a scriptable object of a certain class with certain properties, which are kept in a record data type.

For example, the paragraph object, which divides text on the screen into paragraphs, directly inherits from the item class (see Figure 6.14).

FIGURE 6.14

The paragraph object directly inherits from the item class.

Looking Up an AppleScript

Okay, so now that you know about AppleScript dictionaries, how do you look something up? It really depends on what you're trying to do. Let's say that you have a specific need in mind; for example, you want to tell Safari to set document 1 to a certain URL. You want this to work regardless of how many tabs or windows you've got open — you just want Safari to jump to a URL in that first document.

The next step is to open the Safari dictionary, as you know you'll be working with that set of commands. Click the Safari Suite in the leftmost column, and then click Document in the second column. This reveals three choices:

- source
- text
- URL

You know you want to work with the URL, so you click on that, and you see the definition of the document class, with URL highlighted (see Figure 6.15).

FIGURE 6.15

Looking up the URL property

So now you know how to change the URL — you have to reference it via the document object. Here's the simple AppleScript that does just that:

```
tell application "Safari"
    set URL of document 1 to "http://www.cnn.com"
end tell
```

Summary

In this chapter, you learned about:

- AppleScript objects
- AppleScript dictionaries
- Using objects and dictionaries

At this point, it's time to learn more about the details of AppleScript, starting with variables.

CHAPTER

7

Working with Variables and Properties

In Chapters 5 and 6, you got an overview of AppleScript and its history, a good look at AppleScript Editor, and an introduction to objects and dictionaries. In this chapter, you're going to learn the details you need to know to start creating your first scripts.

Also in Chapter 5, you learned about the basic data types. In this chapter, you're going to build on this knowledge by learning how to use these data types with properties and variables. At first glance, properties and variables seem pretty similar, as they each allow you to store some kind of value, but they are different in key ways.

IN THIS CHAPTER

- **What are variables?**
- **What are properties?**
- **Creating three simple scripts**

What Are Variables?

If you're a programmer, you know that a *variable* is a simple way to store some kind of value so that you can use it again. That value might be a number, a list, or some text (hence the importance of knowing something about supported data types).

For example, you might want to store a number (like your age) or a text string (your name) or a list of items (such as a grocery list), and then reuse it numerous times during your script. Or you might want to find something out (like how many lines exist in all the text files in a certain directory) and then store that value in a variable. The next time you need to know how many lines there are, you don't have to run the command over (unless you need to); you can just look up the value stored in the variable.

If you're not a programmer, then just remember back to those algebra classes you had in high school and college. Remember when the teacher would give you an equation such as 2 + y = 13 and then have you solve for y? In algebra, the y is a variable, and although it isn't exactly the same as in AppleScript, it's the same idea.

Creating variables

You have two basic ways of creating variables in AppleScript. The first involves using the `set` command to create a variable with a value, for example:

```
set myBigDog to "Kafka"
set myLittleDog to "Marlowe"
set myDogs to {"Kafka", "Marlowe"}
set myDogs to {myBigDog, myLittleDog}
```

In the first line of code, I'm using the `set` command to create a variable called `myBigDog` and setting it to the value `Kafka` (mostly because I thought it would be cute to use my actual dog's names in this book, so indulge me a little).

In the second line of code, I create a second variable with `set`, this time `myLittleDog`, and store the value `Marlowe` in it. Again, I'm going with actual dog names for the cuteness factor. In the third line of code, I use `set` to create a third variable called `myDogs`, but this time I use the list data type to store both dogs' names as text strings. I could have also just done it like I did in the fourth line of code, which populated the list with the two variables I'd already created.

The second way to create a variable is to use the `copy` command to copy a value from another variable. So, for example, if I wanted to have another dog named Kafka, I would do this:

```
set myBigDog to "Kafka"
copy myBigDog to anotherDog
display dialog anotherDog
```

Now I have a variable called `anotherDog` that contains the value `Kafka`. Notice the use of `display dialog` in the last line of code? Using `display dialog` is an easy way to see what's inside most variables, as you can see in Figure 7.1.

Before moving on, you need to know a few things about variables. For one thing, and this bears repeating, variable names are not case-sensitive. The variable `myBigDog` is the same as `mybigdog`. If you're coming from another tradition, like Perl or shell scripting, this will certainly make you pause.

When you name a variable, you can only use numbers, letters, and the underscore (_) character. Here are some other rules:

- don't use spaces;
- don't start a variable name with a number;

- don't use any of the reserved AppleScript terms as a variable name, such as:
  - tell
  - say
  - display
  - the
  - repeat
  - if
  - while

FIGURE 7.1

Using `display dialog` to see values

There are many others, as well — you can find a complete list in Appendix C of this book. To be on the safe side, just name your variables with semantically appropriate identifiers. For example, a variable name of `x7dfdjladjadf_123` is perfectly valid, but it doesn't make much sense. You can't look at that variable and remember what it was for. On the other hand, `wife_birthday_party_location`, although long, is more semantically appropriate.

Getting the value of a variable

Earlier in this chapter, I showed you how to use `display dialog` to display the value of a variable. You can also use another command, `get`, to retrieve the value of a variable. For example:

```
set myBigDog to "Kafka"
copy myBigDog to anotherDog
get anotherDog
```

This displays "`Kafka`" in the Result window, as shown in Figure 7.2.

FIGURE 7.2

Results of the `get` command

Working with lists

I want to go back to the earlier example, where I set up a list, as I want to show you how to retrieve items from a list. You could run the `get` command on the entire list, as I do in Figure 7.3.

Sometimes, however, you just want to retrieve a single item from a list. To do that, you use the `get item` *x* construct, where *x* stands for the index of the item you want to retrieve.

Tip

Unlike other programming languages, like Perl, lists are not zero-indexed, but start with item number 1. This is an important little detail to remember if you are trying to pull out item zero and not getting any results!

In the following example, I want to retrieve the string `"Marlowe"` from the `myDogs` list, so I want to get item 2 (Figure 7.4).

Now that you know how to get an item from a list, you can also reset a particular item. All you have to do is use the `set` command and an item number to reset a value, like this:

```
set myDogs to {"Kafka", "Marlowe"}
set item 1 of myDogs to "Vladimir"
get myDogs
```

FIGURE 7.3

Using `get` to retrieve an entire list

The result should be that the first item is changed from Kafka to Vladimir, as shown in Figure 7.5.

FIGURE 7.4

Getting an item from a list

FIGURE 7.5

Resetting a value in a list

Of course, you might not want to replace values in a list, but instead add to the end of the list. To do that, you can use the `copy` command to add items to the end or beginning of a list.

```
set myDogs to {"Kafka", "Marlowe"}
copy "Vladimir" to the end of myDogs
copy "Oscar" to the beginning of myDogs
get myDogs
```

The result, as shown in Figure 7.6, is to have a list that consists of Oscar, Kafka, Marlowe, and Vladimir.

Now that you have a list, you can perform a variety of operations on the items. For example, to know how many items are in a list, use the `count` command:

```
set myDogs to {"Kafka", "Marlowe"}
copy "Vladimir" to the end of myDogs
copy "Oscar" to the beginning of myDogs
count myDogs
```

The answer is shown in Figure 7.7.

FIGURE 7.6

Adding items to the beginning and end of a list

Retrieving certain items can be done with a simple `thru` command:

```
set myDogs to {"Kafka", "Marlowe"}
copy "Vladimir" to the end of myDogs
copy "Oscar" to the beginning of myDogs
get items 2 thru 4 of myDogs
```

FIGURE 7.7

Counting items in a list

The result is a list that contains Kafka, Marlowe, and Vladimir, as shown in Figure 7.8.

To pull out a random item, just use the `some item` command:

```
set myDogs to {"Kafka", "Marlowe"}
copy "Vladimir" to the end of myDogs
copy "Oscar" to the beginning of myDogs
get some item of myDogs
```

Every time you run this script, you should get a random dog name! Because you only have a few dog names in your list, you'll probably get a high number of repeated names, but they're random enough for most purposes.

FIGURE 7.8

Getting a range of items from a list

Working with records

A *record* is a list of unordered key-value pairs. You can't just get item 2 of a record because there is no such thing — what you need to know instead are the names of the keys in the record. If you're coming from another programming language like Perl or PHP, a record is a lot like a hash or associative array.

I'm going to continue with the dogs, but this time I'm going to create a record that helps me store the breed of dog. The keys will consist of breeds, and the values will be the names of the dogs.

```
set myDogs to {yorkie: "Marlowe", mutt: "Kafka"}
```

The result of this command should be the following:

```
{yorkie: "Marlowe", mutt: "Kafka"}
```

To retrieve a value from this list, use the key name. For example, to get the value `"Marlowe"`, you need to get the `"yorkie"` item:

```
set myDogs to {yorkie:"Marlowe", mutt:"Kafka"}
get yorkie of myDogs
```

As you can see from Figure 7.9, this retrieves the value `"Marlowe"`.

FIGURE 7.9

Retrieving a value from a record

Global variables

Unless you explicitly say so, all variables in AppleScript have a *local scope*, which means that setting a variable in one part of your script and then expecting to use it in a subroutine (for example) will make that proposition impossible.

However, you can use the `global` command to set them as global variables.

```
global dogOne, dogTwo
set dogOne to "Kafka"
set dogTwo to "Marlowe"
```

The question is, why would you want to use a global variable? To go back to what I said earlier, you might want to set a global variable at the top of your script, then change that value in different subroutines or other places in your script.

For example, setting dogOne to Kafka and then allowing a subroutine to later rename dogOne to some other value depending on circumstances (for example, let's say you wanted to temporarily refer to dogOne by another name — my wife and I have a nickname for Kafka, and that's "coffee-pot;" it's a long story) would be possible through the use of a global variable.

What Are Properties?

A *property* is a special kind of variable. You'll be using properties a lot once you start working with applications. Each application with AppleScript support has a dictionary of properties available to you as an AppleScripter. However, for the moment, it's only important to know that once you change a property, the change is permanent. You'll see what I'm talking about in a minute.

Usually, you set your properties at the top of a script, using the `property` command:

```
property yorkie : "Marlowe"
get yorkie
```

So far, so good. As you can see from Figure 7.10, the result of the `get` operation is very similar to what you'd expect from a variable.

FIGURE 7.10

Setting and getting a property

Now here comes the interesting part. Create the following script to see how changes affect the use of property values:

```
property yorkie : "Marlowe"
display dialog yorkie
set yorkie to "Vladimir"
display dialog yorkie
```

What happens when you run this script? The first time through, you get a dialog that displays the value `"Marlowe"` and then another one that displays the value `"Vladimir"`. If you run the script again, though, you will see two dialogs, and both will contain the string value `"Vladimir"`. That's because once a property has been changed with the `set` command, you can't change it again unless you use another `set` command.

Most of the time, you'll run into properties when you work with AppleScript dictionaries. You'll get a little information here and there about properties in this chapter, and then again in Part III.

That covers enough of the basics. There's a lot more to learn, of course, but right now it's time to create some simple scripts that will hopefully illustrate what I've been talking about.

Creating Three Simple Scripts

Now that you've learned something about variables and properties, it's time to take what you've learned and create a set of simple scripts.

Script 1: Play random iTunes track

If you're anything like me, your Mac is not just where you work, but where you have fun, too. I have several thousand songs loaded into iTunes, and most of them have been put into some kind of playlist, rated, tagged, and what have you.

I particularly like that '80s sound (and no, I'm not going to apologize for my love of '80s pop, as I graduated high school in 1989), so I decided to write a very simple AppleScript that plays a random track from my *80's Music* playlist.

Here's the script:

```
tell application "iTunes"
    tell (some track of playlist "80's Music") to play
end tell
```

When you look at this script, a couple of questions probably come up right away. You might be thinking to yourself, "Why is there only one end tell on this thing?" The end tell that you see there at the end of the script is finishing the block that started with tell application `"iTunes"`. This is known as a compound tell statement or a tell block. Compound statements are useful because they can contain other statements with multiple clauses and nested multi-line statements.

The other tell statement, the one that plays a random track from the 80's Music playlist, is known as a simple statement. A simple tell statement starts with tell followed by a reference to an object (in this case some track of a certain playlist) followed by a command.

When you run this script, it follows the instructions explicitly:

1. **The `tell application` command opens iTunes.** You'll be using this command a lot, by the way, as it's the way you get an application to do a task for you.
2. **The *80's Music* playlist becomes a list.** The `some track` part of the command is equivalent to grabbing a random item from any kind of list.
3. **That random song is played using the `tell…to play` syntax.**

That's it, really. Click the Play button and you'll get some '80s goodness, at least on my machine. Right now, I'm jamming to Mellencamp's *R.O.C.K. In The U.S.A.*, and all it took was a simple little AppleScript (see Figure 7.11). Life is good.

FIGURE 7.11

Rocking out with Mellencamp and AppleScript

Script 2: Copy files from one directory to another

This script won't be as much fun, but it'll certainly make you productive. Let's say you need to copy a number of files from one directory to another. You can certainly just open Finder and copy the files manually, but what's the fun in that? Write a script to do it instead.

Because there's no fun in writing a single-purpose script of this kind, you'll want to let the user select a directory to copy from and a directory to copy to. You haven't learned too much about user input, but don't worry, as there isn't much to cover here.

Here's the code. Notice that I'm using the continuation character (Option+Return) to keep my lines flowing correctly, even with breaks in the code:

```
tell application "Finder"
    set sourceFolder to choose folder with prompt ¬
    "Choose the source folder"
    set destFolder to choose folder with prompt ¬
    "Choose the destination folder"
    duplicate files of sourceFolder to destFolder
end tell
```

What am I doing here? Well, I'm using the `set` command to create two variables, each of which is determined by some kind of user input. The first variable is called `sourceFolder` and will contain a folder name chosen by the user with the prompt, `"Choose the source folder."`

Similarly, the second variable, called `destFolder`, will contain a folder name chosen by the user with the prompt, `"Choose the destination folder."` In either case, the user can browse for a folder or create a new folder.

Once they choose their two folders, I use the `duplicate files` command to copy whatever is in `sourceFolder` over to `destFolder`.

Now you have a very simple script that lets you duplicate any set of files — provided that you have the permission to read from the source folder and write to the destination folder!

Script 3: View today's events in iCal

Okay, because you did so well with your duplication script, how about writing another one that's just as useful? How about a simple script that opens iCal to today's date and shows you what appointments you have?

All you need to do is use the `tell application` command to open iCal, and then from there, you tell iCal to view the current date's calendar in day view mode:

```
tell application "iCal"
    view calendar at (my (current date))
    switch view to day view
end tell
```

The `current date` variable is built in to the system — it always has the current date built in to it, so there's no need to hard code or otherwise calculate a date. The `switch` and `view` commands are both understood by iCal because they're both in the iCal dictionary.

Each scriptable application, if you'll recall, comes with a dictionary, which is no more than a list of properties and commands that are exposed to AppleScript. To view a dictionary, choose Window➪Library (or Shift+⌘+L), and then select an application from the list.

Figure 7.12 shows the dictionary entry for iCal's `switch view` property. As you can see, this particular property accepts various views (day, week, month), giving you a bit of flexibility when you launch your view. If you wanted to, you could easily change the script to give you either a weekly or monthly view.

You'll be referencing this dictionary as you work through AppleScript, particularly at the beginning of your scripting career.

FIGURE 7.12

The Dictionary for iCal

Summary

In this chapter, you've learned some advanced tips and tricks related to AppleScript variables and operators, including:

- how to work with variables and operators;
- how to work with various data types, particularly lists and records;
- how to build some simple scripts; and
- how to work with dictionaries.

CHAPTER

Operators, Expressions, and Statements

In Chapter 7, you learned quite a bit about variables and properties, and about how you can use the different data types that are available to you. In this chapter, you're going to learn about operators and expressions.

IN THIS CHAPTER

- **What are operators?**
- **What are expressions?**
- **What are statements?**
- **Writing scripts**

In any language, operators, expressions, and statements play a very important part in getting things done. Like variables, they make programming tasks easier. In this chapter, you're going to learn how to use them to help you put together effective AppleScripts.

Think of these three elements (operators, expressions, and statements) as the proper building blocks of any AppleScript. All AppleScripts are made up of one or more statements. Statements consist of expressions, which in turn may contain operators, variables, objects, and other elements. Because you've already learned about variables, it's time to learn about operators, so that's where you'll start in this chapter.

What Are Operators?

An *operator* is any symbol, word, or phrase that derives a value from another value or pair of values — and that comes straight from the AppleScript Language Guide. But what does this really mean? Put simply, an AppleScript operator usually has one or two operands (or parts).

Those operators that call for two operands are called *binary operators*. Those operators that operate on a single operand are known as *unary operators*.

For example, one of the most common operations you'll work with is a *concatenation* (see Figure 8.1). In many programming languages, concatenation is used to combine one string value with another, and AppleScript is no different.

For example:

```
"Austin " & "TX" -- results in "Austin TX"
```

FIGURE 8.1

A simple concatenation operation

AppleScript has both logical and mathematical operators, as well as operators for containment, for concatenation, and for obtaining a reference to an object.

The next section summarizes all the operators that AppleScript uses, and also shows the order in which AppleScript evaluates operators within expressions. Once you've got a good understanding of all the operators, you'll learn more about expressions.

Logical operators

The following sections provide some insight into each of the different types of logical operators:

- Logical conjunction
- Logical disjunction
- Negation

- Equality
- Inequality
- Greater than
- Less than
- Greater than or equal to
- Less than or equal to

Logical conjunction (and)

You can use the `and` operator to combine two Boolean values. The result is true only if both operands evaluate to true. AppleScript checks the left-hand operand first, and if it is false, ignores the right-hand operand. This is known as a short-circuit evaluation, by the way. The result is always a Boolean.

For example:

```
true and true -- evaluates to true
true and false -- evaluates to false
false and false -- evaluates to false
false and true -- evaluates to false
```

Logical disjunction (or)

You can also use the `or` operator to combine two Boolean values. The result is true if either operand evaluates to true. AppleScript checks the left-hand operand first, and if it is true, ignores the right-hand operand. The result is always a Boolean.

```
true or false -- evaluates to true
false or true -- evaluates to true
true or true  -- evaluates to true
false or false -- evaluates to false
```

Negation (not)

Negation is a unary logical operator, and as such, only requires one operand. If the operand to its right is false, it returns true. If the operand to its right is true, it returns false. For example:

```
not true -- returns false
set myvalue to true -- returns true
not myvalue -- returns false

set myvalue to false -- returns false
not myvalue -- returns true
```

Pay attention to what I did in the last two lines. If you set `myvalue` to false and then run `not myvalue`, you're going to get a Boolean true as the return value. That's because true is the negation of false.

Equality

Although the most common way to compare equality is with the = operator, `is equal`, `equals`, and `is equal to` also work the same (see Figure 8.2). You'll use these operators to determine if two values equal each other. For example:

```
"beaumont, tx" is equal to "beaumont, tx" -- true
"beaumont, tx" = "beaumont, tx" - true
9 equals 8 -- false
```

FIGURE 8.2

A check for equality

Inequality

As with equality, you have a lot of options for testing inequality. Basically, if your two operands are different, the inequality result is true (which is what you'd expect). You can use ≠ (Option+= on the keyboard), is not, isn't, isn't equal to, is not equal, doesn't equal, or does not equal. For example:

```
"beaumont, ca" is not "beaumont, tx" -- true
"beaumont, ca" does not equal "beaumont, tx" -- true
"beaumont, ca" doesn't equal "beaumont, tx" -- true
9 isn't 4 -- true
```

Greater than

The greater-than operator results in a true return value if the left-hand operand is greater than the right-hand operand, or follows the right-hand operand in whatever collating sequence is in effect. You can use the most common symbol, >, as well as is greater than, comes after, is not less than or equal to, or isn't less than or equal to, depending on your needs. For example:

```
9 > 4 -- true
11 > 34 -- false
12/1/2009 > 11/30/2009 -- true
"Jethro Gibbs" > "Zeva David" -- false, but only in AppleScript!
```

Less than

The less-than operator results in a true return value only if the left-hand operand is smaller than the right-hand operand, or precedes the right-hand operand in the current collating sequence. You can use the most common symbol, <, as well as is less than, comes before, is not greater than or equal to, or isn't greater than or equal to, depending on your needs. For example:

```
3 < 4 -- true
110 < 54 -- false
12/1/2007 > 11/30/2009 -- true
"Kafka" < "Marlowe" -- true from a alpha order sense
```

Greater than or equal to

The greater-than-or-equal-to operator results in a true return value only if the left-hand operand is greater than or equal to the right-hand operand, or follows the right-hand operand in the current collating sequence. You can use the most common symbol, ≥ (Option+. on the keyboard), as well as is greater than or equal to, is not less than, isn't less than, or does not come before, depending on your needs. For example:

```
5 isn't less than 4 -- true
10 is greater than or equal to 8 -- true
333 ≥ 321 -- true
```

Less than or equal to

The less-than-or-equal-to operator results in a true return value only if the left-hand operand is less than or equal to the right-hand operand, or precedes the right-hand operand in the current collating sequence. You can use the most common symbol, ≤ (Option+, on the keyboard), as well as is less than or equal to, is not more than, isn't more than, or does not come after, depending on your needs. For example:

```
5 is less than or equal to 10 -- true
3 isn't more than 4 – true
11 is not more than 3 -- false
11 ≤ 500 – true
```

Mathematical operators

The following sections provide some insight into each of the different types of mathematical operators:

- Multiplication
- Addition
- Subtraction
- Division
- Integral Division
- Modulus
- Exponentiation

Multiplication (*)

The multiplication operator multiplies the number to its left and the number to its right. For example:

```
9 * 8 -- returns 72
```

Addition (+)

The addition operator adds the number or date to its left and the number or date to its right. Only integers can be added to dates — AppleScript considers an integer as a number of seconds. For example:

```
3 + 4 -- returns 7
3 + -4 -- returns -1

current date + 10 -- will return current date plus 10 seconds into
   the future
```

Subtraction (-)

The subtraction operator subtracts the number to its right from the number or date to its left. As a unary operator, it makes the number to its right negative (as shown in the previous example). As with addition, only integers can be subtracted from a date, and AppleScript interprets it as a number of seconds. For example:

```
100 - 1 -- results in 99
1 - 100 -- results in -99
current date - 10 -- will return current date minus 10 seconds into
   the past
```

Division (/)

The division operator divides the number to its left by the number to its right. You can also use the ÷ symbol (Option+/ on the keyboard). For example:

```
88 / 3 -- results in 29.333333333333
3 / 8 -- results in 0.034090909091
122 / 0 -- error! can't divide by zero
```

Integral division (div)

The integral division operator is exactly like the division operator, except that it only returns the integral part of the result — so you'll only see an integer.

```
88 div 3 -- results in 29
3 div 8 -- results in 0
```

Modulus (mod)

The remainder operator divides the number to its left by the number to its right and returns the remainder. For example:

10 mod 3 -- results in 1Exponentiation (^)

The exponentiation operator raises the number to its left to the power of the number to its right. For example:

```
10 ^ 3 -- results in 1000.0 (10*10*10)
6 ^ 67 - results in 1.3681501912022E+52
```

Other operators

The following sections provide some insight into the remaining operators:

- Concatenation
- Containment
- Object Reference

Concatenation

You've already seen the concatenation operator in action earlier in this chapter. Using the & symbol, you can concatenate strings, lists, and records. For example:

```
"6" & "6" -- results in "66"
6 & 6 -- results in {"6", "6"}
{yorkie:"Marlowe"} & {mutt:"Kafka"} - results in {yorkie:"Marlowe",
   mutt:"Kafka"}
```

Notice how there are different results for the different data types? This has something to do with type coercion. You'll learn more about type coercion after you learn about operator precedence. For now, be aware that AppleScript takes whatever is on the left-hand side of an operator and tries to coerce the right-hand operand into the same data type if it can.

Containment

There are a whole bunch of operators in AppleScript that can help you determine if what you're looking for `starts with` a value, `ends with` a value, `contains` a value, `does not contain` a value, `is in` a container, or `is not in` a container. For example:

```
set mylist to {4, 5, 6}
mylist starts with 4 -- true
mylist contains 3 -- false
6 is in mylist -- true
mylist ends with 8 -- false
23 isn't in mylist -- true
```

By the way, containment operators are great ways to figure out if text strings contain substrings. For example:

```
"Kafka" contains "fk" -- true
"arlo" is contained by "Marlowe" - true
```

In Figure 8.3 I've created a simple grocery list with common items (like eggs and milk). I then check to see if the grocery list starts with the item eggs. The value returned is a Boolean true — the list does start with eggs.

FIGURE 8.3

A simple grocery list

Object reference

You can use the object reference operator to return a reference to just about any data object. For example:

```
set mylist to {4, 5, 6}
set listref to a reference to item 3 of mylist
contents of listref -- results in 6
```

Operator precedence

AppleScript, like most other programming languages, follows a set of rules for operator precedence. If you use more than one operator in an expression, AppleScript will evaluate them according to a precedence order. If you paid attention in your high school math classes, this order won't come as too much of a surprise.

For example:

```
10 + 12 * 3 / 2 -- results in 28
```

Here, AppleScript first multiplies 12 * 3 (to get 36), then divides that by 2 (to get 18), and finally adds that number to 10 (to arrive at 28). That's what Figure 8.4 shows.

FIGURE 8.4

A simple case of mathematical precedence

As with other programming languages, you can use parentheses to force a particular order in your calculations. For example:

```
(10 + 12) * 3 / 2 -- results in 33
```

Here's the order of precedence in AppleScript:

1. Parentheses for grouping
2. Unary + and –
3. Exponentiation (^)
4. * / div mod
5. + -
6. &
7. as
8. < ≤ > ≥
9. = ≠
10. not
11. and
12. or

What is coercion?

In AppleScript, the process of *coercion* involves converting an object from one class to another. You saw this in action with concatenation, when two different data types (such as an integer and string) result in a certain final form after concatenation. You'll run into coercion quite a bit in your AppleScript career, so it's good to get to know it now.

Most of the time, if you're working with operators, AppleScript automatically tries to coerce the right-hand operand into whatever type has been specified for the left-hand operand. For example, let's say that you are trying to concatenate a text value of "6" and the integer 6. The result is a string, "66":

```
"6" & 6 -- result is "66"
```

Going the other way, starting with the integer 6 and concatenating it with a text value of "6", you end up with a list:

```
6 & "6" -- {6, "6"}
```

You can also use the `as` operator to specify a certain coercion. For example, you could tell AppleScript that you want a certain integer to be treated as text:

```
set myValue as 235 as text -- results in the string "235"
```

What's interesting is that using a mathematical operator on `myValue` (which now contains a string value) causes AppleScript to treat this string as a number. See for yourself:

```
myValue + 0 -- results in 235, and myValue is now an integer
```

Considering and ignoring

You can use `considering` and `ignoring` clauses in your operations to help you make better comparisons, particularly when you're dealing with mixed-case text. For example, you might have a situation where mixed case really matters (such as entering a case-sensitive password) and in other cases where you don't really care so much (a folder name).

For example, consider the following:

```
"kafka" = "Kafka" -- true
```

That seems reasonable enough, as you're dealing with the same letters in the same order. But what if you wanted to be a little bit stricter? In that case, you could use `considering case`, as I've done here:

```
considering case
    "kafka" = "Kafka" -- evaluates to false
end considering
```

You can also use the `ignoring` clause to help you ignore different elements, such as white space and hyphens. For example, both of the following comparisons would evaluate to false:

```
"foot loose" = "footloose" --sometimes you can't escape the 80s
"flux-capacitor" = "flux capacitor" -- ditto
```

In order to be less strict, I've used `ignoring white space and hyphens` here:

```
ignoring white space and hyphens
    "foot loose" = "footloose" --sometimes you can't escape the 80s
    "flux-capacitor" = "flux capacitor" -- ditto
end ignoring
```

Now both of these return a value of true.

You can even combine various consider/ignore clauses with `but`. For example, you might want to consider case but ignore white space:

```
considering case but ignoring white space
    "Foot loose" = "footloose" -- false
    "footloose" = "foot loose" --true
end considering
```

You can even do the same with diacritical marks (you know, all those little symbols you see in French, German, and Scandinavian words, not to mention many other languages).

For example, if you were to compare anime and animé, you would get a return value of false:

```
"anime" = "animé" -- false
```

This is mostly because the Unicode value of é is different from plain old English e. This is great to know, but sometimes you don't need that level of consideration, so you can use `ignoring diacriticals` to be less strict:

```
ignoring diacriticals
    "anime" = "animé" -- true!
end ignoring
```

One more thing, and then I'll move on. By default, AppleScript considers punctuation when it makes comparisons. So the following two strings would return false when compared:

```
"hello world!" = "hello world" -- false
```

This doesn't make a lot of sense in some situations, so you can use ignoring punctuation in order to be less strict:

```
ignoring punctuation
    "hello world!" = "hello world"
end ignoring
```

What Are Expressions?

An *expression* in AppleScript is any series of lexical elements (such as operators and object specifiers) that have a value. The simplest expressions, called *literal expressions*, are representations of values in scripts. You can have more complex expressions that combine literals, variables, operators, and object specifiers.

AppleScript converts all expressions into values at runtime. That's why, when you run a simple expression such as:

```
14 * 10
```

what you see in the results pane is the value:

```
140
```

An object specifier is a special type of expression that specifies some or all the information needed to find an object. The difference between an object and an object specifier is like the difference between a domain name and a Web site. The Web site contains all the content, whereas the domain name just points the way to the content. The content might change, but the domain name stays the same.

For example, in the following code listing, you can see that I'm asking the Finder to retrieve all the items found in the Documents folder in my home area (in UNIX path terms, I'm asking for all files and folders that live in ~/Documents).

```
tell application "Finder"
    activate
    get items of folder "documents" of home
end tell
```

In this case, the parent object is the Finder application, which has a container called home. This home container always points to a user's home directory. Inside this container is any number of folders, but I've specified a particular container called documents. Finally, I've asked for all the items of that documents container.

The result is a list that contains all the items:

```
{folder "EmailCampaigns" of folder "Documents" of folder "myerman" of
   folder "Users" of startup disk of application "Finder", folder
   "ia_examples" of folder "Documents" of folder "myerman" of folder
   "Users" of startup disk of application "Finder", folder "Macware"
   of folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder", folder "mail_archives" of
   folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder", folder "marketingtips" of
   folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder", folder "Microsoft User
   Data" of folder "Documents" of folder "myerman" of folder "Users"
   of startup disk of application "Finder", folder "presos" of
   folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder", folder "proposals" of
   folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder", folder "resumes" of folder
   "Documents" of folder "myerman" of folder "Users" of startup disk
   of application "Finder", folder "SVN" of folder "Documents" of
   folder "myerman" of folder "Users" of startup disk of application
   "Finder", folder "TDDM" of folder "Documents" of folder "myerman"
   of folder "Users" of startup disk of application "Finder", folder
   "Web Receipts" of folder "Documents" of folder "myerman" of
   folder "Users" of startup disk of application "Finder", folder
   "writing" of folder "Documents" of folder "myerman" of folder
   "Users" of startup disk of application "Finder", folder "writing_
   mafia" of folder "Documents" of folder "myerman" of folder
   "Users" of startup disk of application "Finder", folder "www" of
   folder "Documents" of folder "myerman" of folder "Users" of
   startup disk of application "Finder"}
```

I could just as easily have asked for a particular item:

```
tell application "Finder"
    activate
    get item 8 of folder "documents" of home
end tell
```

When I run that code, I get a return value like this (it so happens that item 8 of my documents folder is another folder called proposals, which contains all the proposals I've written for clients--but enough about me):

```
folder "proposals" of folder "Documents" of folder "myerman" of
   folder "Users" of startup disk of application "Finder"
```

Notice that I'm talking about containers here. A *container* is an object that contains one or more items. A folder contains files and other folders. A text document contains paragraphs, and paragraphs contain words.

You can use the keywords `of` or `in` to specify a container:

```
tell application "Finder"
    first item of first folder of startup disk
end tell
```

with the result being:

```
application file "Address Book.app" of folder "Applications" of
   startup disk of application "Finder"
```

Furthermore, the command you just ran, the one that contains `first item of first folder of startup disk`, is a relative object specifier — there isn't enough information in it for AppleScript to complete the job. The object specifier must be wrapped in a `tell application "Finder"` command.

Other specifiers that have enough information are known as absolute object specifiers. For example:

```
version of application "TextEdit" -- returns 1.6 on my MacBook Pro
```

What Are Statements?

A *statement* is any series of commands, variables, keywords, operators, constants, or expressions that follow AppleScript syntax. Every script, in fact, consists of statements, and when AppleScript executes a script, it processes the statements in order and carries out the instructions in these statements.

Every time you program in AppleScript, you build up a series of statements. A statement might contain a variable assignment, a loop, an object specifier, an operation, or some other lexical statement.

Writing Scripts

Now that you've had some time to understand operators, expressions, and statements, it's time to write a few scripts.

Script 1: Checking for e-mail validity

In this section, I'll write a very simple script that checks to see if a user-provided e-mail address is valid. I'll be using some containment and logical operators, of course, but also an example of gathering user input. Although you're not going to learn about user input until Chapter 10, this should be pretty straightforward, like Script 2 in Chapter 7 that involved choosing from a list of folders. The script will also contain some `if` statements, which 'aren't covered in detail until Chapter 9, but you'll get the idea.

In this script, I'll consider an e-mail to be valid if it follows all of these rules:

- It contains an @ symbol.
- It ends with .com, .net, or .org.
- It has at least one character to the left of the @ symbol.

This is by no means a complete definition of a valid e-mail address (especially the `ends with` containment, but you can add others as you see fit), but it will do for now.

Here's the entire script:

```
display dialog ¬
    "enter an email address:" default answer "test@example.com"
set emailAddress to text returned of result
if not ((emailAddress contains "@") and ¬
    ((emailAddress ends with ".com") or ¬
        (emailAddress ends with ".net") or ¬
        (emailAddress ends with ".org")) ¬
        and ¬
    ((offset of "@" in emailAddress) > 1)) ¬
    then
    display dialog "Oops, that doesn't look right" with icon 0
else
    display dialog "Looks okay for now!"
end if
```

The first part of the script asks for user input using the `display dialog` command. For now, it's important to understand that if you were just to put in the command `display dialog` without the `default answer` keyword, you'd end up with just a dialog that says "enter an email address" and no user input field.

```
display dialog ¬
    "enter an email address:" default answer "test@example.com"
set emailAddress to text returned of result
```

Of course, if you just wanted an empty field in the dialog, you can use the following construct, using empty quotes:

```
display dialog ¬
    "enter an email address:" default answer ""
set emailAddress to text returned of result
```

Cross-Reference

In Chapter 10, you'll learn more about all the options you have for dealing with user input, including choosing folders and files, choosing from a list, choosing colors, and much more.

Notice that I'm setting a variable called `emailAddress` to whatever value comes back from the user. Once I have that value stored, I can start checking to make sure it conforms to what I think is a good e-mail address. I can do that with a mixture of containment (`contains` and `ends with`) operators and a logical (greater-than) operator.

All of those tests are wrapped in a negation operator (not), which allows you to easily flip the expected value to the negative (for example, if it does not contain X, Y, Z):

```
if not ((emailAddress contains "@") and ¬
    ((emailAddress ends with ".com") or ¬
        (emailAddress ends with ".net") or ¬
        (emailAddress ends with ".org")) ¬
    and ¬
    ((offset of "@" in emailAddress) > 1)) ¬
```

Essentially, if the `emailAddress` value does not contain an @, doesn't end with .com, .org, or .net, and doesn't have at least one character before the @ symbol (an offset of 1 would indicate that the @ symbol is at the beginning of the string), then you have a problem:

```
then
    display dialog "Oops, that doesn't look right" with icon 0
```

If there is a problem, display a dialog with an appropriate error message and icon 0 (a red octagon that looks like a stop sign).

Otherwise, display a dialog with a success message and end the `if` test:

```
else
    display dialog "Looks okay for now!"
end if
```

The complete script and user input dialog are shown in Figure 8.5.

FIGURE 8.5

Checking for e-mail validity

Script 2: Inspect the Trash

AppleScript has a whole series of named destinations that you can use, and one of them is `trash`, which points to your Trash folder. You can use the named location to perform all kinds of activities. For example, you can get a list of items in the Trash folder by doing this:

```
tell application "Finder"
    get items of trash
end tell
```

What you get back from AppleScript is a list of files and folders in the Trash folder, something like this:

```
{document file "3469404669_67a7964f53_t.jpg" of trash of application
   "Finder", document file "3469404693_6dc795c91c_t.jpg" of trash of
   application "Finder", document file "3469404711_eaa522b279_t.jpg"
   of trash of application "Finder", document file "3469404789_
   ea4d208806_t.jpg" of trash of application "Finder", document file
   "3469404807_b5b0ffdabf_t.jpg" of trash of application "Finder",
   document file "3469416959_382fc25d60_t.jpg" of trash of
   application "Finder", document file "3469416991_ccd524a909_t.jpg"
   of trash of application "Finder", document file
   "3469417005_76c4ea2fca_t.jpg" of trash of application "Finder",
   document file "3469417019_772f0e0661_t.jpg" of trash of
   application "Finder", document file "3470216444_c16ce1a1e7_t.jpg"
   of trash of application "Finder", document file
   "3470216580_4905e072ce_t.jpg" of trash of application "Finder",
   document file "3470216584_ecedf37220_t.jpg" of trash of
   application "Finder", document file "3470216588_22368c063e_t.jpg"
   of trash of application "Finder", document file "3470216602_
   a64c342956_t.jpg" of trash of application "Finder", document file
   "3470216620_2cc7dc8827_t.jpg" of trash of application "Finder",
   document file "3470216622_d132d80387_t.jpg" of trash of
   application "Finder"}
```

To get a count of how many items are in the Trash folder, you could do this:

```
tell application "Finder"
    get count of items of trash
end tell
```

The result might be 0, or 150, or 20, or however many items are in the Trash folder. As simple-minded as this little script is, bear in mind that you now have a nice little AppleScript that you can use in an Automator workflow to see if the Trash folder has items in it.

In Figure 8.6, you can see that at the moment, I have 47 items in my Trash folder.

How about something a bit more complicated? Why not check to see if the count of items in the Trash folder has reached a certain threshold, and then prompt the user to empty the Trash folder? Again, this script uses `if` statements and user input (including a check on which button is pressed in the dialog), but I think you'll be able to follow:

```
tell application "Finder"
    set countTrashItems to count of items of trash
    if (countTrashItems > 100) then
        display dialog "Time to empty trash?" buttons {"Yes", "No"}
        if button returned of result = "Yes" then
            empty the trash
```

```
                display dialog "Just took out the trash!"
            else
                display dialog "OK, I'll leave the trash alone!"
            end if
        end if
    end tell
```

FIGURE 8.6

Counting items in the Trash folder

In the first part of the script, I set a variable named `countTrashItems` to the count of items found in the Trash folder. If that variable is higher than 100, then I use a simple dialog to ask the user, "Time to empty trash?"

Notice that on that dialog, I specify two buttons: one that says Yes and another that says No (see Figure 8.7).

By specifying buttons, I've overridden the default setup, which calls for OK and Cancel buttons. From there, it's a simple test to see which button was pressed:

- If the Yes button is pressed by the user, then empty the Trash folder and display an appropriate dialog.
- If the No button is pressed by the user, just show a message without emptying the Trash folder.

Although you're using all kinds of elements here (the `if` statements, user input dialogs, variable settings), the key to this script is the greater-than test. Nothing happens until that Trash folder count goes above 100.

FIGURE 8.7

Taking out the trash

Summary

In this chapter, you've learned some more about the AppleScript language, specifically:

- how to work with operators;
- how to ignore or consider case, white space, hyphens, diacriticals, and punctuation when making comparisons;
- how to work with expressions;
- what statements are; and
- how to write scripts that work with standard operators.

CHAPTER 9

Conditionals and Loops

In the last two chapters, you learned how to work with both variables and operators. You've built a few scripts, some of which go way beyond what you've already been formally introduced to. In this chapter, you're going to learn about conditional tests and loops. If you're already a programmer, these concepts will be easy for you to learn. If you're not a programmer, don't worry; you'll learn it very quickly.

IN THIS CHAPTER

- **What are conditional tests and loops?**
- **Conditional tests**
- **Repeating loops**
- **Writing scripts**

What Are Conditional Tests and Loops?

A conditional test is very simple: It's an expression that tests to see if something (usually a value or variable) in your code passes some kind of test. Once the test is passed, the code can set a value, perform an action, or do something with that information. If the test isn't passed, then you might have a simple default solution, or a more detailed response.

In other words, conditional tests are all about decisions, and your use of them is basically teaching AppleScript how to make good decisions — hopefully, the kinds of decisions you would make as a human doing them manually.

For example, you might have a randomly selected number created by AppleScript. You could write a conditional test to check to see if that number is higher than 10. If it is higher than 10, then you might display a dialog. If it isn't higher than 10, then you might just have the Mac beep or something.

That's a fairly simple conditional test. A more complex one might involve checking to see if the random value is higher than 10, as before. If it is, it might take a certain action such as displaying a dialog. If the number is in fact lower than 10, then you might have various other checks to see if it is between 7 and 9, between 4 and 6, or less than 4.

Each decision in a conditional test is called a *branch*, and in this chapter, you'll learn how to create all kinds of conditional tests that use progressively more complex branches.

A *loop* is an expression that lets you do something over and over again. There are loops that allow you to do something a set number of times, other loops that will continue until some kind of test is passed (for example, the number of seconds since the loop started or the number of files processed), and even loops with defined start and stop values.

The second part of this chapter will be concerned with loops. By the time you're through with this chapter, you'll know all about loops and how to use them in your AppleScripts.

Conditional Tests

In AppleScript, you use a variation of the `if` expression to help make good decisions. Using the `if` expression, you can test to see if a value is greater than or less than another value, or if a certain amount of time has passed, or some other kind of value. Furthermore, you can use `if-else` and `if-else if-else` tests to put together fairly complex tests of any kind.

The if test

The basic `if` test is very simple. It begins with `if` and ends with `end if`:

```
set myYorkie to "Marlowe"
if (myYorkie = "Marlowe") then
    display dialog "Hooray!"
end if
```

Here I've created a very simple variable called `myYorkie` and populated it with the value `Marlowe`. If indeed `myYorkie` is equal to Marlowe at the time of the `if` test, then it displays a dialog that displays the word `Hooray!`

If you were to consider the example from the introduction of this section, one in which AppleScript creates a random number, and then you check to see if it is higher or lower than 10, then the `if` test would look like this:

```
set randNumber to (random number (20)) as integer
if (randNumber > 10) then
    display dialog randNumber
end if
```

In the first line, I'm using the `(random number(20)) as integer` expression to create a random number between 1 and 20 and return it as an integer. Then I use an `if` test to see if that value is higher than 10. If it is higher than 10, then I display that randomly generated number in a dialog.

Please note that you'll only see the dialog (as shown in Figure 9.1) if you actually get a number higher than 10. If you want to see other values, then you need to move on to the two-value `if` test.

FIGURE 9.1

The simple `if` test combined with a random number

The two-value if test

Sometimes you need more than just one possible outcome to a conditional test. Sometimes you need to know what happens if the test isn't passed. For that, AppleScript provides you with an `else` branch. You can easily add the `else` branch into your code this way (again, revisiting the random number example from the previous section):

```
set randNumber to (random number (20)) as integer
if (randNumber > 10) then
    display dialog "More than 10: " & randNumber
else
    display dialog "10 or less: " & randNumber
end if
```

As you can see, I've tweaked what appears in the dialogs, as well as added the `else` branch. Now you're going to see a dialog message each time, but with some added context.

Figure 9.2 shows what happens when the random number generator gives you a number 10 or less.

FIGURE 9.2

Adding an `else` branch to the random number generator

The multi-value if test

Sometimes you need to know all kinds of outcomes to a test. Now you could go and create a bunch of `if-else` branches and then nest more `if-else` tests within each `else` branch, but that gets very confusing very quickly. For example:

```
set randNumber to (random number (20)) as integer
if (randNumber > 10) then
    display dialog "More than 10: " & randNumber
else
          if (randNumber ≥ 7 and randNumber ≤ 9) then
        display dialog "Between 7 & 9: " ¬
    & randNumber
    else
     if (randNumber ≥ 3 and randNumber < 7) then
        display dialog "Between 3 and 6: " ¬
        & randNumber
```

```
        else
           display dialog "It's a 0, 1 or 2: " ¬
           & randNumber
        end if
      end if
end if
```

This code works perfectly fine, but it sure isn't elegant. It doesn't take too long before your eyes start to cross and you have to use your finger to trace the code to figure out what's going on. So what is going on? Well, in the simplest terms, what the code is testing for is this:

1. **Is the randomly generated number greater than 10?** If so, display a dialog with the result.
2. **If not, do another test to see if it is between 7 and 9.** If so, display a dialog with the result.
3. **If not, check to see if the number is between 3 and 6.** If so, display a dialog with the result.
4. **If not, then the number is a 0, 1, or 2, so display a dialog with one of those three numbers in it.**

This sequence is illustrated in Figure 9.3, along with the result from the last branch of that code.

FIGURE 9.3

An inelegant approach to complex `if` tests

A much more elegant approach is to use an `if-else if-else` branch, in which you're free to use as many `else if` branches as you need to keep testing for different outcomes (see Figure 9.4). The `if` branch is still your main test, and the `else` branch becomes kind of a default location for any value that fails all other tests.

Here's the new code, delivered as a single `if` test with multiple branches:

```
set randNumber to (random number (20)) as integer
if (randNumber > 10) then
    display dialog "More than 10: " & randNumber
else if (randNumber ≥ 7 and randNumber ≤ 9) then
    display dialog "Between 7 & 9: " & randNumber
else if (randNumber ≥ 3 and randNumber < 7) then
    display dialog "Between 3 and 6: " & randNumber
else
    display dialog "It's a 0, 1, or 2: " & randNumber
end if
```

Even if you're totally new to the world of AppleScript, this just seems a lot easier. It's fewer lines of code, a lot less nesting, and a much more elegant approach that is easier to test, debug, and maintain in the future.

FIGURE 9.4

A more elegant approach to multiple-branch `if` testing

That's pretty much all you need to know about conditional testing for now. You'll be using these tests a lot in your AppleScript efforts, so don't worry; you'll be getting a lot of practice.

Repeating Loops

You'll often find yourself in a situation where you need to do the same thing over and over again, whether patiently waiting for a user to provide you with the proper input, processing 100 files in a directory, or making sure that a task is repeated until some conditional test returns a true value.

In AppleScript, you can repeat an action with a `repeat` loop. The basic structure of a `repeat` loop looks like this:

```
repeat
  -- do something
end repeat
```

In the upcoming sections, you'll learn all about the different kinds of `repeat` loops.

Repeating forever

Sometimes, you need to create a loop that runs forever. One such situation is when you're waiting for a specific event to happen — for example, you might be waiting for a certain file to be created in a folder, for certain data or calculations to emerge from other sources, or for a user to type in a certain value.

If that's the case, simply use the basic `repeat` loop. In the following example, I use a `repeat` loop to wait for a user to enter their first name into a dialog:

```
repeat
    display dialog ¬
        "Please enter your first name:" default answer ""
    set firstName to text returned of result
    if firstName is not equal to "" then exit repeat
end repeat
```

Using the dialog, I ask the user to enter their first name. Notice that I'm using a default answer of "" on the dialog, meaning that if they don't enter a value, I capture an empty string.

Once the user enters their information, I capture it in the variable `firstName`, which I can easily test to see if it does not equal "". If the test passes (in other words, the user has entered some value into the dialog), then the script can exit the `repeat` loop. Otherwise, the dialog remains in place.

Repeating a set number of times

Sometimes you want to repeat a loop a set number of times. For example, you may want to capture the user's first name, but you only want to give them a set number of times to actually enter a value. In that case, you would use the following format:

```
repeat x times
-- do something
end repeat
```

In this code, `x` would represent the number of times you want to repeat the code: 3, 5, 20, 2000 — it's up to you.

Here's what the original code would look like if I limit it to three attempts. Also notice that I've added a simple `if` test at the end that displays a dialog if the `firstName` variable is still blank.

```
repeat 3 times
    display dialog ¬
        "Please enter your first name:" default answer ""
    set firstName to text returned of result
    if firstName is not equal to "" then exit repeat
end repeat
if firstName = "" then display dialog ¬
     "You never gave us your first name!"
```

The entire run is shown in Figure 9.5.

FIGURE 9.5

This example repeats the process three times.

Repeating with defined start and stop values

In some cases, you might need to repeat a loop a certain number of times, but with certain defined start and stop values. For example, you might want to create five folders on your desktop, each with its own name. For that kind of situation, you use a `repeat with` statement, which has this format:

```
repeat with increment_value from starting_value to ending_value
-- do something
end repeat
```

In this example, `increment_value` is just a dynamic placeholder that changes each time through the loop, `starting_value` is where to begin the loop, and `ending_value` is where to end the loop.

For example, to create those five folders on your desktop, you could write a script like this:

```
tell application "Finder"
    repeat with incrementValue from 1 to 5
        make new folder at desktop with properties ¬
          {name:incrementValue as string}
    end repeat
end tell
```

In this case, each time through the loop, the value of `incrementValue` changes. The first time through the loop, its value is 1, and so the folder that is created on the desktop is named 1. The next time through the loop, the variable's set to 2, and so on. The result is five folders on the desktop, as shown in Figure 9.6.

There's no need to start at 1, of course. You can write scripts that use other start values:

```
tell application "Finder"
    repeat with incrementValue from 10 to 50
         make new folder at desktop with properties ¬
           {name:incrementValue as string}
    end repeat
end tell
```

This script would create 41 folders on the desktop instead of just 5, with the first folder named 10 and the last folder named 50.

FIGURE 9.6

Using a `repeat with` loop to create folders

To increase the increment value, simply add a `by` clause:

```
tell application "Finder"
    repeat with incrementValue from 1 to 50 by 5
        make new folder at desktop with properties ¬
         {name:incrementValue as string}
    end repeat
end tell
```

This creates folders named 1, 6, 11, 16, 21, 26, 31, 36, 41, and 46 on the desktop. Each time through the `repeat` loop, the value of `incrementValue` is incremented by 5, not 1.

To decrement a value, simply add a `by` clause with a negative number, but make sure that your starting number is higher than your ending number:

```
tell application "Finder"
    repeat with incrementValue from 10 to 1 by -1
       make new folder at desktop with properties ¬
          {name:incrementValue as string}
   end repeat
end tell
```

Please note that throughout all these examples I've been using the option-return character to create a continuation character (¬) in my scripts. This is a crucially important thing to remember, as the continuation character is different from a regular line ending that denotes the end of an expression. Without this character, the result would be an error message indicating a compiler failure, but not necessarily a specific reason for that error. In other words, you might end up chasing errors for hours if you don't use the character!

What you end up with when you run this script are ten folders, with the 10 folder created first, all the way down to the 1 folder.

If you are coming from PHP, C, or Perl, you're probably used to a `for` loop in which you initiate a variable, provide a test for ending the `for` loop (when the variable is set to a certain value), and then provide an increment/decrement factor. There isn't a `for` loop in AppleScript, but the `repeat with` expression is its nearest cousin.

Repeating until something is true

The `repeat until` statement is used to repeat a process or action until something is true. For example, you might set a variable to a certain value, and then repeat a loop until that value has reached a certain value, incrementing each time through the loop.

For example:

```
set loopCounter to 1
repeat until loopCounter = 30
    display dialog loopCounter
    set loopCounter to loopCounter + 1
end repeat
```

In this code, I set a variable named `loopCounter` to a value of 1. Then I use a `repeat until` loop to keep working until `loopCounter` is set to 30 (see Figure 9.7). Then, each time through the loop, I display the value of the variable in a dialog and then increment the `loopCounter` variable by one.

By itself, this code isn't very useful at all, but you can imagine how useful it could be. For example, you could rewrite the infinite `repeat` loop to something like this:

```
set firstName to ""
repeat until firstName does not equal ""
    display dialog ¬
      "Please enter your first name:" default answer ""
    set firstName to text returned of result
end repeat
```

This does the exact same thing as the infinite `repeat` loop, but it's a bit more elegant.

FIGURE 9.7

Using `repeat until` to loop through a process

Repeating while something is true

You can also set a `repeat` loop to operate while something is true. This is known as the `repeat while` loop and is used to test for a particular scenario to appear.

For example, let's say that you wanted to display a dialog if and when a particular file was created at a certain location. Here's a bit of code that waits until a file called `test.log` is written to the desktop. It uses a `repeat while` loop that keeps testing to see if that file has been created yet.

Once the file is created, the `repeat` loop exits and AppleScript displays a dialog.

```
set myLogFile to (path to desktop as string) & "test.log"
tell application "Finder"
    repeat while (file myLogFile exists) = false
    end repeat
end tell
display dialog "The log file is on the desktop!"
```

To make this work, I ran the script and then opened a Terminal window and ran the UNIX command `touch ~/desktop/test.log`. As soon as I did that, the dialog popped up, informing me that "The log file is on the desktop!" as shown in Figure 9.8.

FIGURE 9.8

Using `repeat while` to detect a log file

Repeating with a list

Sometimes, you'll need to go over a list of items, processing each item one at a time. For example, you might have a list of files, a list of to-do items, or a list of groceries that you need to buy.

AppleScript uses `repeat with` to achieve this kind of loop, with this format:

```
repeat with currentValue in listOfValues
-- do something
end repeat
```

Each time through the loop, the value of `currentValue` is shifted to another item in the list. So, for example, let's say that you have a series of common nouns in a list, and you want to create a series of folders on your desktop that match those names. Here's the code that would do it:

```
set myList to {"dogs", "cats", "cars", "planes", "food"}
tell application "Finder"
    repeat with noun in myList
        make new folder at desktop with properties ¬
         {name:noun as string}
    end repeat
end tell
```

This code is almost precisely the same as the other `repeat with` example, except that it contains a list of names that is then looped through to create the folders. Also notice that I use the more semantically appropriate variable name of `noun` in the loop — but it doesn't really matter; I could have used `foo`, `j`, or `somethingthatisreallylong`.

Figure 9.9 shows the code in action, including the folders created by the `repeat with` loop.

FIGURE 9.9

Using a `repeat with` loop to process a list

Writing Scripts

In this section, you're going to take what you've learned and create a pair of functional scripts that use conditionals and loops. That way you'll get some practical hands on knowledge before moving on.

- The first script will use a conditional check to see if a folder exists. You'll use that check within a larger context of creating a new folder.
- The second script will use a repeat loop to export all email addresses from Address Book to a text file.

Script 1: Check if folder exists

In this first script, you're going to put together a basic script that prompts a user to enter a text string. The script will then check to see if that folder exists in a certain location (the current user's home folder). If it doesn't exist, the script creates the folder and displays a dialog. If it already exists, the system displays an error dialog.

```
set destFolder to (path to current user folder)
display dialog "New Folder Name" default answer ""
set newFolder to the text returned of result
tell application "Finder"
    if (exists folder newFolder of destFolder) then
        display dialog "Sorry, that folder exists!"
    else
        make new folder at destFolder ¬
          with properties {name:newFolder}
        display dialog "New folder created!"
    end if
end tell
```

Figure 9.10 shows the results of running this simple script and getting a message that the new folder was created by the script.

FIGURE 9.10

Creating a folder with a simple script

Script 2: Export e-mail addresses

The second script uses a simple repeat loop to process each person in your Address Book and then extract every e-mail address for each person it finds. The script does this by getting a count of the number of contacts in Address Book, then uses that as the upper limit of the repeat loop (see Figure 9.11).

Once the script finds all the e-mail addresses, it prompts you to choose a file to save your information to, then saves it in text format.

```
tell application "Address Book"
    set emailsList to {}
    set peopleCt to (count every person)
    repeat with i from 1 to peopleCt
```

```
            set emailsList to emailsList & (get value of every email ¬
            of person i)
        end repeat
        set filename to choose file name with prompt ¬
        "Save emails as text:" default name "emails.txt"
        set outfile to open for access filename with write permission
        repeat with thisaddress in emailsList
            write thisaddress & ", " to outfile
        end repeat
        close access outfile
    end tell
```

FIGURE 9.11

Exporting e-mail addresses with a repeat loop

Summary

In this chapter, you've learned some advanced tips and tricks related to AppleScript, including:

- what conditional tests are and how to use them; and
- what loops are and how to use them.

CHAPTER 10

Handling User Input

So far, you've learned about variables, operators, expressions, and loops, and so you should be feeling pretty comfortable with AppleScript. In this chapter, you're going to learn about handling user input.

User input comes in many different forms, and this chapter will cover the most common types of input you'll encounter as you create AppleScripts.

IN THIS CHAPTER

- What is user input?
- Sending a message to the user
- Asking for text input
- Working with buttons
- Choosing folders and files
- Choosing from a list
- Choosing an application

What Is User Input?

User input involves any situation in which you ask the user for input: filling in a text field, pushing a button, choosing a folder or set of files, choosing a color, or making some other choice. Sometimes, you end up simply communicating with the user in a one-way fashion — showing a dialog with an alert or using the system beep. Although these aren't technically examples of user input, I'm going to cover them here.

Despite the example of one-way communication, user input almost always involves two parts:

- asking for the user input in some way; and
- capturing the user's response.

What you do with that response is entirely up to you, of course; however, you almost always have to match the kind of user input device to your stated goal. If you want the user to choose a folder, then use the folder prompt, not a text input field. If you want to capture a text string from the user, then don't present them with an option to choose a color.

Some of these user inputs have already been covered as examples in previous chapters. With a topic like AppleScript, this kind of situation is inevitable. Although it may seem strange to repeat that information here, it only makes sense to consolidate information in a book of this type.

Sending a Message to the User

Before getting into the proper methods of gathering user input, I'd like to show you three ways you can communicate with a user. They are, in increasing order of complexity:

- the `beep` command;
- the `say` command; and
- the `display dialog` command.

The beep command

Your Mac has a basic audio alert built into it — a beep. You can make your Mac beep simply by using the `beep` command:

```
beep
```

That's all there is to it. You can use the `beep` command at the completion of a command, or if certain scenarios arise, or in combination with other events. For example, if there's an error, you could display a dialog and have the system beep at the user.

You can beep more than once by adding an integer to the command, like this:

```
beep 2
```

By the way, you can also control the volume level with the `set volume` command followed by an integer. The `set volume` command accepts levels 0 through 10 as arguments:

```
set volume 8
```

Try the following system check:

```
repeat with myvol from 1 to 10
    set volume myvol
    beep
    delay 1
end repeat
```

As you can see, it's a simple `repeat with` loop that sets the volume incrementally, beeps once, and then delays one second before going to the next volume level. Without the delay factor, you'd just hear one beep, the last one.

The say command

Your Mac can also talk to the user in a variety of voices. To make your Mac say something, just use the `say` command:

```
say "AppleScript rocks!"
```

After a second, the Mac will say whatever string you just fed it as an argument.

Depending on what voice you've set, you'll get different effects. A full list of voices is available if you go to System Preferences, click Speech, and then click the Text to Speech tab. As shown in Figure 10.1, you can set a default system voice by selecting one from the pop-up menu.

FIGURE 10.1

The complete list of system voices

As you can see, there is a whole list of male, female, and novelty voices available for your use. I use the Alex voice at the moment, but could easily switch my default to some other voice, like Zarvox (if I'm in the mood to hang out with Cylons from the classic *Battlestar Galactica* series) or Princess (if I thought of my computer as a pre-teen girl).

In any case, you can switch between voices directly from your AppleScripts by adding a `using` clause to your `say` commands:

```
say "AppleScript rocks!" using "Zarvox"
```

Whether or not you use a voice is up to you. Some people hate using them, while others love them. Personally, I can't wait until there's a Mac voice that's like the voice of the computer on the U.S.S. Enterprise (original series, please!), but until then, I'll stay with Alex.

The display dialog command

The most common way to communicate with a user, and the way you've already seen it done a few times in this book already, is to use the `display dialog` command. Simply issue the command along with some text, and you get an all-purpose dialog:

```
display dialog "AppleScript rocks!"
```

Figure 10.2 shows you what this dialog looks like. Notice that you not only get the message, but also two standard buttons: Cancel and OK.

FIGURE 10.2

A standard dialog

What if you wanted different buttons? Just add a `buttons` clause followed by a simple list of options; each item in the list becomes a button:

```
display dialog "AppleScript rocks!" buttons { "Meh", "Right on!"}
```

Figure 10.3 shows both the code and the resulting dialog.

Furthermore, you could specify a default button using the `default` clause at the end of the `button` clause:

```
display dialog ¬
    "AppleScript rocks!" buttons {"Meh", "Right on!"} ¬
    default button "Right on!"
```

This causes the Right on! button to highlight when you run the script, as shown in Figure 10.4.

Two quick notes before I move on to icons. First, you could have also provided a button number to get the same effect. In the previous example, Meh is button 1, and Right on! is button 2.

```
display dialog ¬
    "AppleScript rocks!" buttons {"Meh", "Right on!"} ¬
    default button 2
```

FIGURE 10.3

Displaying a dialog with non-standard buttons

FIGURE 10.4

Setting a default button

Second, I mention the button number format only because you have to be careful using button names. You have to use the complete name, including punctuation, when you refer to a button by name, or AppleScript throws an error. For example, calling the Right on button (sans exclamation point) throws an error, as shown in Figure 10.5.

FIGURE 10.5

Getting a button name wrong throws an error

By the way, you can also add icons to your dialogs. You get three icons to choose from:

- the stop icon (icon 0);
- the note icon (icon 1); and
- the caution icon (icon 2).

You add them using a `with icon` clause at the end of the `display dialog` command. Figure 10.6 shows the three different icons in action, along with sample code.

FIGURE 10.6

The three types of icons

Asking for Text Input

Sometimes you need to ask a user for some text input. For example, you may need them to enter a username, password, or some other kind of detail. To do that, use `display dialog` but add a `default answer` clause to the end of it.

The default answer doesn't have to contain anything at all; just a "" will do. However, feel free to enter whatever appropriate text might help the user make a good entry. For example, here's a simple dialog that asks for the user's favorite color. Notice that the default answer clause is empty, and so is the accompanying user input widget (Figure 10.7):

```
display dialog "what is your favorite color?" default answer ""
```

FIGURE 10.7

Using a blank default answer for capturing text input

In the following example, I offer a default suggestion of blue:

```
display dialog "what is your favorite color?" ¬
default answer "blue"
```

As you can see in Figure 10.8, blue appears in the text box.

Of course, the default answer could have been `monkey` or `submarine`, as could the user input, but you get the idea.

FIGURE 10.8

Offering up a suggestion for text input

Working with Buttons

In a previous section, you learned how to add custom buttons to your dialog. You also learned that even if you don't specify any buttons, AppleScript provides them.

In some cases, you won't have to actually do anything with these buttons. You display a dialog to show the user a status message and that's it. The only reason you're doing it is to tell them that the script has now run its course.

Sometimes, however, you actually want to respond to whichever button the user has pressed. For example, if you go back to the example where I ask the user what their favorite color is and convert it to use buttons, you might end up with something like the following:

```
display dialog ¬
     "what is your favorite color?" buttons {"blue", "red", "yellow"}
   ¬
     default button "blue"
```

The result is shown in Figure 10.9: a dialog with three buttons, with the blue one highlighted by default.

Now, what happens when a user selects one of those buttons? At the moment, nothing happens, because I haven't added any way to handle the result. To capture the button that's pressed, I'll add a statement that captures the result of the button returned:

```
display dialog ¬
    "what is your favorite color?" buttons {"blue", "red", "yellow"}
   ¬
      default button "blue"
set myColor to button returned of result
```

FIGURE 10.9

Setting up an array of buttons to choose from

I now have whatever button is pressed saved in the variable `myColor` and can do what I want with it. I could set up a whole series of conditional tests to take different actions, or I could feed the value to another process, or anything else.

For the moment, all I want to do is display another dialog that confirms the value captured in the variable:

```
display dialog ¬
    "what is your favorite color?" buttons {"blue", "red", "yellow"}
   ¬
      default button "blue"
set myColor to button returned of result
display dialog myColor
```

Figure 10.10 shows the code and the resulting dialog when I click the blue button.

FIGURE 10.10

Capturing the value of a button

Choosing Folders and Files

Sometimes, you need to prompt the user to select something specific, like a particular folder, or a set of files. This way, you can capture what they've selected and then process that selection in some way — for example, you might copy all the files in one folder to another folder, or rename all the selected files in some way.

To prompt a user to select a folder, use the `choose folder` command, like this:

```
choose folder
```

This basic command opens a Finder window set to a default directory (in all likelihood, the folder where you're saving your AppleScripts, if you run the command directly from Script Editor).

If you want to add a custom prompt, add a `with prompt` clause:

```
choose folder with prompt "Choose a folder:"
```

And if you want to specify a particular folder, use the `default location` clause, as I've done in the following example, stipulating the desktop as the default location:

```
choose folder with prompt ¬
    "Choose a folder:" ¬
      default location (path to desktop folder)
```

For files, you can use the `choose file` command with a variety of similar clauses to help the user make a good selection. For example, you can change the prompt using `with prompt`, set a `default location`, restrict the type of file with `of type`, and allow `multiple selections`.

For example, the following code prompts the user to select one or more files from the Documents folder:

```
choose file with prompt ¬
    "Choose file(s):" ¬
      default location (path to documents folder) ¬
      with multiple selections allowed
```

To capture this selection, remember to set a variable:

```
set myFiles to ¬
     choose file with prompt ¬
        "Choose file(s):" default location (path to documents folder)
   ¬
      with multiple selections allowed
```

As you can see from Figure 10.11, this sample code results in a list of files saved to `myFiles`.

FIGURE 10.11

Saving a file selection to a list

Another useful command to know is `choose file name`, which you can use to prompt the user to save a file. You can set a prompt, set a default name, and even specify a default location for the file.

```
choose file name with prompt ¬
    "Please choose a file name:" default name ¬
    "logfile.txt" default location (path to desktop folder)
```

Figure 10.12 shows this code in action.

FIGURE 10.12

Saving a file with `choose file name`

Choosing from a List

Sometimes, you want to present users with a list of choices and then have them select one of those items from that list. All you have to do is set up the list and then use the `choose from list` command to let them choose:

```
set myList to {"red", "blue", "yellow", "green", "orange", "purple"}
choose from list myList
```

That code gives you the most basic list chooser available, as shown in Figure 10.13.

When the user makes a selection (for example, by clicking orange and clicking OK), the result is a list:

```
{"orange"}
```

Because the returned result is a list, you can capture that result by capturing item 1 of the result, like this:

```
set myList to {"red", "blue", "yellow", "green", "orange", "purple"}
choose from list myList
set myColor to item 1 of result
```

FIGURE 10.13

Setting up a basic list chooser

However, as with other choosers, you can add a whole bunch of extras. For example, you can change the prompt, allow multiple selections, allow an empty selection, and even set default items. In the following code, I've set up a custom prompt, set `blue` as the default color, and allowed multiple choices. I've also made sure to capture a list at the end, and not just an item from a list.

```
set myList to {"red", "blue", "yellow", "green", "orange", "purple"}
choose from list myList ¬
    with prompt ¬
      "Favorite color?" default items {"blue"} ¬
      with multiple selections allowed
set myColor to result
```

When I run the code this time, I choose a variety of colors from the list and end up with the following result:

```
{"blue", "yellow", "green", "orange"}
```

Choosing an Application

Sometimes you want to prompt a user to select an application. You can do this very easily with the `choose application` command. Here's the basic command:

```
choose application
```

As you can see in Figure 10.14, this command displays a list of applications.

FIGURE 10.14

Prompting a user to select an application

Whichever selection is made, the result looks like this:

```
application "Address Book"
```

Of course, you can also allow multiple selections, making the script a launch pad (notice that I've also changed the default prompt):

```
choose application with prompt ¬
    "Launch application(s)" ¬
    with multiple selections allowed
```

When I run the script, I make various selections, and the result comes back as a list:

```
{application "Chess", application "Address Book", application
   "BBEdit"}
```

Summary

In this chapter, you've learned some advanced tips and tricks for gathering user input, including:

- how to talk to or beep at the user;
- how to create dialogs;
- how to create file and folder choosers;
- how to allow users to create from a list; and
- how to choose from a list of applications.

CHAPTER 11

AppleScript Subroutines

So far, your AppleScript education has included learning about variables, operators, conditional statements, loops, and a few other details. However, you've only created simple scripts that run as standalone files — you haven't had the opportunity to create any scripts that are more complex, or any script that would require reusable code.

In this chapter, you're going to learn how to create and use (and reuse) subroutines to your advantage. Learning about subroutines will take your AppleScript knowledge to the next level.

Please note that many books on AppleScript refer to subroutines as *handlers*. Technically speaking, they're absolutely correct, but please note that in the AppleScript world, handlers can mean a lot of different things. For example, when you tell an application such as Safari to open, you're using an on handler.

Combining the on handler with a custom handler (or subroutine) that you've built yourself seems a bit confusing, especially if you have experience with shell scripting, Perl, Python, or Java, where custom subroutines are really a separate deal. For that reason, I insist on calling a custom handler a subroutine, just to avoid any possible confusion and to make it easier for programmers from other languages.

IN THIS CHAPTER

- What are subroutines?
- Defining a subroutine
- Running a subroutine
- Using loops, conditionals, and variables in subroutines
- Recursive subroutines
- Reusing subroutines
- Some useful subroutines

What Are Subroutines?

If you're an experienced programmer, then you're probably familiar with subroutines (or functions, procedures, or handlers, as they're often called) in other languages. Put simply, a subroutine is any bit of code that contains

instructions or logic that can be reused. A subroutine always has some kind of unique name that you can call from a script, and you always get the same (or similar) results from calling that subroutine.

For example, let's say you have a very simple AppleScript that adds two values together:

```
2 + 2 -- result is 4
```

This bit of code is so specific as to be almost nonreusable, so let's amend it to be a bit more general:

```
set x to 2
set y to 2
x + y -- still comes out to 4
```

Okay, the code is still very simple and still contains some hard-coded values, but at least now you're using some variables. But it still isn't a subroutine — in other words, I can't just call this script and have it add any two values together.

In the next section, I'm going to fix that by creating a very simple subroutine, one that will add two values together. I'll then be able to use that subroutine whenever I want.

In other words, subroutines let you do the following:

- **Organize your code better.** Writing scripts becomes a lot easier if you use subroutines. You can work your way logically through your script, thinking about major functionality: you need to add two numbers together, you need to validate a ZIP code, you need to check the count of files in different folders, and so on. Using subroutines, you can make it easier to not only write scripts, but also maintain them later. Instead of having five or six different places where you are adding two numbers together (or whatever you're trying to do), you now just have one place. Using subroutines makes it more intuitive to follow your code because of this new organization.
- **Work smarter, not harder.** Subroutines give you more flexibility, more efficiency, and more return on your time investment. Yes, it might take a bit longer to concoct a great subroutine, but once it's written, you can reuse it dozens (if not hundreds) of times over the course of its lifetime. Updating a subroutine means updating every bit of your script that uses your code, making it easier to script and rescript.
- **Reuse reliable code.** A subroutine is a reusable nugget of reliable functionality. It's something you don't have to bother with ever again if you don't want to. If you figure out how to get a certain result — for example, how to check for certain values in a log file — then you can reuse that code every time you need to run a similar operation. Even if you don't reuse the exact same code, you can use an existing subroutine as a starting point for the next project.

- **Plan ahead.** When you're stuck in a particular problem, you sometimes script for right now and not necessarily for tomorrow, next week, or next year. Writing a subroutine forces you to think and plan ahead. It may be faster to just put together some code that performs a one-off function, but it's smarter to think about how many times you might reuse that code in the future. Planning ahead makes you a better scripter.
- **Break tasks down into smaller, manageable chunks.** Good scripters and programmers are able to analytically break down a problem into its smallest parts. Small subroutines that attack a small problem give you incredible leverage over even the most complex of tasks.

Defining a Subroutine

A subroutine always has the same basic format:

```
on subroutineName (list of arguments)
     -- do something useful
end subroutineName
```

The subroutine I need is going to have a very simple name (`addTwo`) and it will accept two values (the two values you want to add together). It will return the added value and that's it. Here's the subroutine:

```
on addTwo(x, y)
    return x + y
end addTwo
```

This subroutine doesn't look like much, but believe me, large structures are built from small, beautiful components like this. But before moving on, let me talk about these arguments.

First of all, whatever names you give to your argument variables (the x and y in this example) are local to the subroutine. If you have an x or a y in the main flow of your code, they in no way impact the x and y inside your `addTwo()` subroutine.

The second thing to understand is that subroutines don't have to have arguments at all. A subroutine could be constructed to start a new Finder window, for example, or launch Safari, with no real value being returned from the subroutine. However, if you do choose to use arguments, then you must return a value of some kind, even if you're just returning true or false.

One more thing: If you choose to use arguments, then all arguments are required. None can be optional, and there's no easy way to set a default value for an argument as you can in other languages. You also need to specify your arguments in the same order as the subroutine specification. This isn't such a big deal with your simple little script, but in other cases, you might be building a subroutine that passes in a string followed by a number followed by another string, or some other combination of data types.

Running a Subroutine

Once you've built a subroutine, how do you run it? It's very simple: You run a subroutine by calling it, using its name followed by parentheses. If your subroutine requires arguments, place your arguments inside the parentheses.

Here's an example of calling the `addTwo()` function:

```
addTwo(20,50) --returns 70
addTwo(2,5) --returns 7
addTwo (-2,5) --returns 3
```

Figure 11.1 shows the subroutine in action.

FIGURE 11.1

Using the `addTwo()` subroutine

What's most interesting about even this very simple subroutine is that it accepts any two values that are numbers. For example, if you took a count of items in a list and added that number to a random number, you'd still get back a sum of the two:

```
set myListCount to count of {"eggs", "milk", "bread"}
set myRandom to (random number 100)
addTwo(myListCount, myRandom)
on addTwo(x, y)
    return x + y
end addTwo
```

Running this script over and over again, you get different values because of the random factor used to generate the second number. In Figure 11.2, one particular run results in a value of 67.

FIGURE 11.2

A more complicated use of `addTwo()`

There's something very important to know about that value being returned by the subroutine. In order to actually do something with that return value, you need to set some kind of variable that will capture it. As it stands now, both scripts that I've created don't really leave much room for future use of the result returned from the subroutine.

Here's an easy way to fix that:

```
set myListCount to count of {"eggs", "milk", "bread"}
set myRandom to (random number 100)
set mySum to addTwo(myListCount, myRandom)
on addTwo(x, y)
    return x + y
end addTwo
```

Now you've got `mySum` as a variable to work with.

If you've got a subroutine that doesn't return a value, then don't worry about setting a variable. For example, let's say you've created a subroutine that launches Safari. There's no need to have some kind of variable to store the results:

```
launchSafari()
on launchSafari()
    tell application "Safari"
        activate
    end tell
end launchSafari
```

If you're calling a subroutine within a `tell` block, then you have to add a `my` keyword in front of your subroutine call in order to make it work. For example, if you're using a `tell` block to open up the Finder and work with folders, and in the middle of that you decide to call `launchSafari()`, then you need to call `my launchSafari()` instead.

For example:

```
tell application "Finder"
    set myDesktop to (path to desktop as string)
    -- other activities
    my launchSafari()
end tell
on launchSafari()
    tell application "Safari"
        activate
     end tell
end launchSafari
```

Using Loops, Conditionals, and Variables in Subroutines

You already know how to use loops, conditionals, and variables inside your AppleScripts. In this section, I'm going to talk about how to use these and other valuable tools inside subroutines. You already know that any variable names you use for arguments stay local to the subroutine. Here are some other things you can do inside a subroutine:

- You can use any repeat loop you like.
- You can use conditionals.
- You can do just about anything you like, except define another subroutine within a subroutine.

If you want to use a variable from the main part of your script (in other words, from outside a subroutine), you have two choices:

- Pass that variable to your subroutine as an argument.
- Define the variable as a global variable or property.

Recursive Subroutines

A *recursive subroutine* is a subroutine that calls itself. One of the most commonly used examples is a subroutine that calculates a factorial. In mathematics, a factorial of a number is the product of all positive integers from 1 to that number. For example, 5 factorial is equal to 1 * 2 * 3 * 4 * 5, or 120.

In order to create a subroutine that calculates the factorial of a positive integer, you need the subroutine to call itself recursively until the sequence is done. So here is the code that does exactly that:

```
on factorial(x)
    if x > 0 then
        return x * (factorial(x - 1))
    else
        return 1
    end if
end factorial
```

To run the code, you give the subroutine a positive integer. If that integer is greater than 0, it returns a value of that integer multiplied by the result of the factorial of the integer minus 1, recursively until the integer is no longer zero. Because it's recursive, you get a return value of the combined result.

Figure 11.3 shows the results of `factorial(10)`. As you can see, it's not hard to generate some pretty big numbers with a simple factorial subroutine.

FIGURE 11.3

Running `factorial(10)`

Reusing Subroutines

As you can see, creating subroutines is the smart way to go, but it does you no good to have multiple copies of subroutines scattered throughout all your scripts. If you needed to update a certain subroutine, you'd have to search for half a dozen (or more) copies of the same subroutine.

The smartest thing to do is to create libraries of subroutines and then load those libraries when you need them. For example, you could create a math library that contains your `factorial()` subroutine, save it as a compiled script, and then use the `load script` command to make whatever subroutines are in that library available to other scripts.

In the following example, I've created a script called `mathlib`, copied the `factorial()` subroutine into it, and saved the whole thing as a script. If you look at this script, and then Ctrl-click it and choose Get Info from the pop-up menu, you'll see that it's saved with an .scpt extension.

In the script called `factorial`, the one you created earlier, I can then load the `mathlib` script and use a `tell` block to run the `factorial()` subroutine. In the following example, I give the full path to where I've saved my script, which will likely be different on your machine:

```
set mathlib to (load script file¬
"myerman:library:scripts:myerman:mathlib.scpt")
tell mathlib
    factorial(10)
end tell
```

As you can see from Figure 11.4, the result of `factorial(10)` is the same as it was before.

FIGURE 11.4

Using `load script` to load reusable libraries

From this point, you can see how useful it would be to have the mathlib file. You could add all kinds of generic and specialized subroutines — for calculating area, volume, angles, whatever. You'd have one repository for these types of subroutines, and you could load them wherever you needed them.

Some Useful Subroutines

It's time to practice what you've learned. In this section, you're going to build a series of subroutines that will make you more productive. Feel free to start your own libraries using these subroutines.

Subroutine 1: Create a folder

The first subroutine you'll work on is a simple one that creates a new folder. You'll be able to use this kind of subroutine in a lot of different ways and situations. For example, you could simply state in a script where you want a folder created, and hardcode the name of the new folder. Or you could ask the user for a new folder name and a destination and then use that information to create your new folder, just like you did in Chapter 9.

First, I'll create the subroutine:

```
on createFolder(newFolder, ExistingFolder)
    tell application "Finder"
        make new folder at folder ExistingFolder ¬
          with properties {name:newFolder}
   end tell
end createFolder
```

As you can see, this subroutine takes two arguments. The first is the name of the new folder you want to create, and the second is the name of the folder in which to place the new folder. So, for example, you could create some simple variables that contain the values you want:

```
set myFolder to "test"
set myDesktop to (path to desktop as string)
createFolder(myFolder, myDesktop)
on createFolder(newFolder, ExistingFolder)
    tell application "Finder"
```

```
            make new folder at folder ExistingFolder ¬
              with properties {name:newFolder}
        end tell
    end createFolder
```

The result of this script is very simple:

```
folder "test" of folder "Desktop" of folder "myerman" of folder
   "Users" of startup disk of application "Finder"
```

However, it's not very useful at the moment. What if you let the user select the name of the new folder and the destination folder? It's time to use what you learned in Chapter 10, dealing with user input.

In the previous example, I hardcoded values for the new folder and the existing folder. In this new version of the script, I'm going to use a dialog to elicit input from the user, and then save it to the variable `myFolder`. Then I'm going to use a choose folder dialog to have the user tell me where to save the new folder.

```
set myFolder to text returned of ¬
    (display dialog ¬
      "Choose new folder name" default answer "test")
set myDesktop to choose folder with prompt ¬
"Save folder in location" ¬
default location (path to desktop folder)
createFolder(myFolder, myDesktop)
on createFolder(newFolder, ExistingFolder)
    tell application "Finder"
        make new folder at folder ExistingFolder ¬
          with properties {name:newFolder}
    end tell
end createFolder
```

The result is elegant and simple. First AppleScript prompts the user to enter a string, and then prompts for a folder in which to create the new folder; then it creates a new folder at that location with the string entered by the user.

As you can see in Figure 11.5, I've created a new folder called tommy in the Documents folder. I chose this location as an alternative to the default location (i.e., the desktop).

FIGURE 11.5

Creating a folder with user input and a subroutine

Subroutine 2: Move an item to the Trash

The second example script from the end of Chapter 7 involved working with the Trash. The script that you ended up with was a simple one that checked to see if the number of items in the Trash had reached a certain threshold, and if so, emptied it.

Now you're going to write a very simple subroutine that allows you to actually designate one or more items that go in the Trash.

Here's the subroutine, which I've called trashIt:

```
on trashIt(itemPath)
    tell application "Finder"
        delete item itemPath
    end tell
end trashIt
```

All you have to do is feed `trashIt()` a path, and it does the rest. So, true to form, you can write a little script that goes something like this:

```
set myPath to ¬
    choose file with prompt ¬
      "Trash which file?" default location (path to desktop folder)
trashIt(myPath)
on trashIt(itemPath)
    tell application "Finder"
        delete item itemPath
     end tell
end trashIt
```

When you run this script, it prompts you for a file. Its default open location is the desktop, but you're free to choose any other file. Once you've selected a file, it's moved to the Trash.

In Figure 11.6, I've deleted a file called *testing* from the desktop. You can see from the Result pane that the file now resides in the Trash folder.

FIGURE 11.6

Moving a file to the Trash folder

Subroutine 3: Count items in a folder

The final subroutine is just as simple as the other two, but it's something you'll run into a lot: working with folders and files. Specifically, you're going to learn how to create a simple reusable subroutine that counts all the items in any folder you feed to the subroutine as an argument.

This subroutine uses a `tell` block to launch the Finder. You'll pass in a `folderName` variable, which you'll use as the basis for a `count` expression, and then return that count of items.

Here's the subroutine:

```
on countFolderItems(folderName)
    tell application "Finder"
        set theFolder to (folder folderName)
        set myCount to the count of (every item of theFolder)
        return myCount
    end tell
end countFolderItems
```

I can create all kinds of scripts based on this subroutine, but I think I'll use the old standby of asking the user to choose a folder.

```
set myDesktop to ¬
    (choose folder with prompt ¬
      "Choose a folder" default location (path to desktop folder))
set counter to countFolderItems(myDesktop)
on countFolderItems(folderName)
    tell application "Finder"
        set theFolder to (folder folderName)
        set myCount to the count of (every item of theFolder)
        return myCount
    end tell
end countFolderItems
```

By now, this pattern should be pretty familiar to you: Ask for user input, save that user input in a variable, and then pass that variable to a subroutine, which returns the answer as an integer.

In Figure 11.7, for example, I've asked for a count of items in ~/Documents, and the answer is 2, which means that I have 2 folders and files directly inside that folder.

FIGURE 11.7

Getting a count of items in a folder

Summary

In this chapter, you've learned how to create reusable subroutines and learned why they're useful. You've also learned how to create recursive subroutines and how to store and load your subroutines from more centralized libraries.

CHAPTER 12

Applets and Droplets

In Chapter 11, you learned how to create reusable subroutines. Learning how to work with subroutines marks a serious step forward, as it allows you to code smarter and create reusable automation solutions.

In this chapter, you're going to take your knowledge one step further and learn how to create applets and droplets. As you'll see, applets and droplets are both very simple to comprehend, but open you up to a whole new world of possibilities in your scripting.

IN THIS CHAPTER

- What are applets?
- Three applets
- What are droplets?
- Two droplets

What Are Applets?

An applet is an application with a minimal user interface that nonetheless stands alone. It has its own icon and can be double-clicked to make its code run. It doesn't need to run inside of the Script Editor, which is what you've been doing so far in this book. The reason why it can run standalone is because an applet contains some compiled bootstrap code inside it that allows it to run in that fashion.

Just about any AppleScript you create can be turned into an applet. There's really no difference between a simple script that contains `say "hello"` and an applet with the same code, except that the applet can run by itself when you double-click its icon.

Creating an applet is extremely simple and requires only two rules (and even one of these rules is optional):

1. **First, you should encapsulate your code in an** `on run` **statement.** This isn't necessary, as the `run` command is implied, but it's good practice.
2. **Second, you need to save your script as an application (and no, this isn't optional!).**

Here's the basic format:

```
on run
   --do something useful
end run
```

Once you've created an applet, it's still editable, but you'll have to first open AppleScript Editor and choose File➪Open from the menu to edit it, as double-clicking the file merely runs the code that's inside of it.

Three Applets

In the following sections, you're going to learn about building applets by getting your hands dirty. Here are the projects you'll build:

- Open a Web page in Safari
- Run a Shell script
- Get file size

Each of these projects are very simple, but I'll try to run through some different possibilities with each to give you more of a feel for what you need to know.

Open a Web page in Safari

You've created a variety of scripts so far that open up applications, so creating an applet that opens up Safari seems like a natural transition.

First, I'll create a very simple version of this script that just opens Safari without any fuss.

```
tell application "Safari"
    activate
end tell
```

When I save this script, I'll save it as *applet open safari*, make sure that I've selected Application from the file type pop-up menu, and then save it to my desktop. What I'll end up with is an icon labeled *applet open safari* that I can double-click. When I do double-click it, Safari opens to my default home page (remember though, that in Safari 4 you might have your defaults set to Top Sites or even a blank page).

Here's the basic script shown in Figure 12.1.

FIGURE 12.1

The basic applet for opening Safari

Figure 12.2 shows you what the icon should look like on your desktop.

FIGURE 12.2

The desktop icon for the applet open safari file

Of course, there are a few problems with this approach, the first being that you haven't really added an `on run` handler to the code. Now, this code isn't required, but it is a good idea to do things properly. The other problem is also simple, but very annoying: If you double-click your script icon while Safari is still open, you don't get a new window. In fact, it appears as though your script isn't working at all.

To solve that problem, you need to do a bit of extra scripting. You not only want Safari to activate, but you also need to tell System Events to access the Safari process and click a certain menu item — specifically, the File➪New Window menu option.

Here's the new code, including the `on run` statement:

```
on run
    tell application "Safari" to activate
    tell application "System Events"
        tell process "Safari"
            click menu item ¬
                "New Window" of menu "File" of menu bar 1
        end tell
     end tell
end run
```

Figure 12.3 shows the revised applet code inside Script Editor.

FIGURE 12.3

The revised applet open safari script

There's one more thing that you should do to make this a truly useful applet: ask the user to enter a URL that Safari can open. To do that, add a display dialog component that asks for this information. Then, all you have to do is save that URL into a variable and then tell Safari to open that location.

What's the best thing about this rewrite? Telling Safari to open a new location will, by default, open either a new window or a new tab, depending on how you've set your preferences inside Safari, which means you can get rid of some of that pesky code.

```
on run
    set myUrl to text returned of ¬
      (display dialog ¬
      "URL to open?" ¬
      default answer "http://www.google.com")

    tell application "Safari"
        open location myUrl
    end tell
end run
```

The final code, along with its `display dialog` component, is pictured in Figure 12.4.

FIGURE 12.4

The final code for opening a specific location

Please note that Safari does its best to open up non-canonical addresses. Type in **google.com**, and it opens up `http://www.google.com`.

Run a shell script

Next, you're going to learn how to run a shell script in AppleScript. You'll do that by learning how to run a very basic shell command, `ping`, which you'll use frequently if you're ever in a situation where you're testing network connectivity. As you should know, you can use the `ping` command to help determine if a server is up or down.

Here's a very simple AppleScript that asks a user for a server address, pings the address with four packets of data, and then displays the results of the command in a dialog. The first part of the script is wholly lifted (and then modified) from the last script. Notice that running the shell script only requires a `do shell script` command:

```
on run
    set myDomain to text returned of ¬
      (display dialog ¬
      default answer "www.google.com")

    display dialog (do shell script "ping -c4 " & myDomain)
end run
```

Figure 12.5 shows the code and the results of the `ping` command on `www.google.com`.

FIGURE 12.5

Running the `ping` command on `www.google.com`

You can, of course, write even more complex shell scripts and run them inline like this. In Part 4, you'll have the opportunity to run various shell scripts from both Automator and AppleScript.

Get file size

Your Mac comes with a Get Info feature in Finder that you can use to get all kinds of information about a file — metadata, file size, permissions, and more. Some days, though, all you want to know is what the file's size is, nothing else, and you want to be able to do it by double-clicking an icon on the desktop.

Here's a simple AppleScript that does just that. It opens a choose file dialog, saves your choice as a variable (`myFile`), uses the `info for` command to derive the size, and then prints it in a dialog display.

```
on run
    set myFile to choose file
    set mySize to ¬
      size of (info for myFile) as string
    display dialog (myFile as string) & ¬
    return & "size: " & mySize
end run
```

Figure 12.6 shows the code in action.

FIGURE 12.6

Getting the file size with an applet

What Are Droplets?

A droplet is simply an applet that also contains an `on open` statement. This simple addition creates an icon on which you can drop files and folders, which then become the arguments for the `on open` code to run against.

For example, if you have a script that converts all filenames to lowercase letters and converts spaces in filenames to underscores, then you can encapsulate that code in an `on open` statement. That way, all you have to do is drop your files onto the droplet icon to run your code.

Can you combine both droplet and applet functionality into one bit of code? The answer is a resounding yes. It's possible to create a compiled bit of AppleScript that responds to a user's double-click, and to files and folders being dropped on it.

For example, if you go back to the example of a script that removes spaces in filenames:

- A double-click initiates the `on run` portion of the script, in which AppleScript prompts the user for one or more files with a choose file dialog.
- Dropping files on the icon triggers the `on open` portion of the script, in which each file is processed one at a time by the droplet.

Here's the format:

```
on open xyz --this can be any variable name
    -- do something useful
end open
```

Two Droplets

In the following sections, you're going to create two droplets. The first will be a rewrite of the file size applet you wrote in the previous section. The other will allow you to change the desktop background to any image dropped on the droplet.

Get file size

The first thing you're going to learn how to do is to rewrite the get file size applet you wrote in the previous section as a droplet. Along the way, you're going to make a few improvements — for example, you're going to allow multiple files to be chosen (or dropped) at once, and you're going to centralize the display code in one place so that it can be reused by either the applet or droplet code.

First of all, here's a quick refresher on the original code:

```
on run
    set myFile to choose file
    set mySize to ¬
      size of (info for myFile) as string
    display dialog (myFile as string) & ¬
      return & "size: " & mySize
end run
```

Two things you'll do right away will make the process simpler: Make it possible for the user to choose more than one file at a time, and create a subroutine for processing each file and getting back a file size. The way you're going to do that is to set up some concatenation, allowing a string to be filled up with more and more information, and then, upon exiting the `repeat` loop, allowing that concatenated string to be published.

```
on run
    set myFiles to ¬
      choose file with multiple selections allowed
    getSizes(myFiles)
end run
on getSizes(files_)
    set myString to ""
    repeat with file_ in files_
        set myFile to ((file_ as string) & return ¬
          & "size: " & size of (info for file_) ¬
          as string) & return & return
        set myString to myString & myFile
    end repeat
    display dialog myString
end getSizes
```

Please notice that I'm using the variable names `file_` and `files_` in the `getSizes()` subroutine. This is because `file` and `files` are both reserved names in AppleScript — using them as variables causes AppleScript to basically grind to a halt.

Figure 12.7 shows the new script so far, along with the results of two PDFs that I've chosen to run through the program. Notice that I've renamed the file droplet *get file size*.

FIGURE 12.7

Creating a reusable subroutine and allowing for multiple file selections

However, as lovely as the new code is, you're not done yet! This code is still just an applet. It is not a droplet yet, as it doesn't have an `on open` statement. The `on open` statement is just a simple addition at this point, because all you need to do is call the reusable subroutine that is already in place.

```
on open theFiles
    getSizes(theFiles)
end open
```

That's all you have to do to make this a droplet! Figure 12.8 shows the whole script, as well as what happens when I drag two Excel documents onto the droplet. As you can see, I get a concise report of filenames and file sizes.

FIGURE 12.8

Getting filenames and file sizes from a droplet

Keep in mind that if you drop a folder onto this new droplet, you get a combined report of all sizes for files and folders within that main folder. In Figure 12.9, you can see that I've dropped a folder with more than 26MB worth of data onto the droplet.

FIGURE 12.9

Dropping a folder onto the droplet

Set the desktop background

By now, you probably know how to set your Mac's desktop background picture by going to System Preferences. Well, in this section, I'm going to show you how to build a very simple droplet that changes your desktop background to any image you drop on it.

The first step in this process is to create an error message if anyone double-clicks your droplet instead of dragging an image on it. To do that, you use a simple dialog to give the user an error message:

```
on run
    display dialog "Error! Please drag an image onto this droplet!"¬
      buttons {"OK"} with icon 0
end run
```

Figure 12.10 shows what happens if someone simply double-clicks your new droplet.

FIGURE 12.10

An error message is displayed when someone double-clicks a droplet.

Next, you'll add an `on open` statement that processes all images that you send to the droplet. The first thing you have to do is only process the last file in the group that is sent to the droplet. This way, if the user selects multiple images, only one is set as the desktop background. Also notice that I'm using a `try` block to attempt to set the desktop picture — if the file isn't an image or if it's corrupted, then I display an error message.

Here's the `on open` statement:

```
on open files_
    set file_ to last item of files_
    tell application "Finder"
        try
            set desktop picture to file (file_ as text)
         on error
            display dialog "Invalid file!" with icon 0
         end try
    end tell
end open
```

That's really all there is to it. Here's the full script in Figure 12.11.

FIGURE 12.11

The complete droplet for changing the desktop background image

Summary

In this chapter, you've learned:

- What applets and droplets are;
- How to create applets and droplets;
- When and where to use applets and droplets; and
- How to create some useful applets and droplets.

CHAPTER 13

Folder Actions

In Chapter 12, you learned how to create applets and droplets, which brought your AppleScript abilities up to a whole new level. Instead of having to invoke and run your scripts via AppleScript Editor, you can now create scripts that can be invoked via simple actions: either by double-clicking a mouse or by dropping files onto the scripts.

In this chapter, you're going to learn about folder actions, another extremely useful tool that will make your automations a lot more sophisticated. Without further ado, let's get started!

THIS CHAPTER

- **What are Folder Actions?**
- **Enabling Folder Actions**
- **Folder Action Scripts**
- **Creating your first Folder Action Script**
- **Creating Folder Action plug-ins with Automator**

What Are Folder Actions?

Put simply, Folder Actions are a feature that lets you associate AppleScripts with individual folders. Whatever you do to that folder (move, copy, add files, remove files, rename, whatever) can trigger the Folder Action Script that's been associated with that folder.

In other words, think of Folder Actions as giving you the ability to create *hot folders* that respond to actions. Imagine being able to create a folder on your desktop that automatically sends any file placed into it to the printer. Or adding an alert feature to your Public folder that lets you know when anyone on the local network adds a file to it. Or adding a workflow to a folder that archives any files that are dropped into it.

At the end of the day, Folder Actions are just one more way of getting things done in an efficient manner.

Enabling Folder Actions

The easiest way to enable Folder Actions is to follow these steps:

1. **Create a folder on your desktop (or other location, if you prefer).**
2. **Ctrl+click the folder and choose Folder Actions Setup (see Figure 13.1).** An application called Folder Actions Setup launches (see Figure 13.2).

FIGURE 13.1

Enabling Folder Actions

FIGURE 13.2

The Folder Actions Setup application

3. **Click the + button at the bottom of the first column.**
4. **Select the folder for which you just enabled Folder Actions.** The system shows a list of available Folder Action Scripts.
5. **Choose one of the available Folder Action Scripts to associate with your folder and click Attach (see Figure 13.3).** In my case, I've chosen the script called convert – PostScript to PDF. It's one of the scripts that come prepackaged on the Mac, and it converts any postscript files to PDF. (I'll discuss these scripts more in a moment.)

FIGURE 13.3

Adding a Folder Action Script to a folder

6. **When you're done, you should have a folder in the left column and one script listed in the right column (see Figure 13.4).** You can add more scripts later if you like — they are run in the order they are listed.

To test this new Folder Action, I downloaded a test postscript file and dragged it to my new actions folder. Within seconds of dropping the file there, the Folder Action Script ran, creating two folders: an Original Files folder, into which the postscript file was copied, and a PDF Files folder, into which the converted PDF was placed. Figure 13.5 shows the completed process.

FIGURE 13.4

Folder and script associated

FIGURE 13.5

The Postscript-to-PDF conversion in action

Folder Action Scripts

After seeing Folder Actions in, well, action, you're probably wondering where that list of available scripts came from, and what constitutes a *legal* Folder Action Script.

Both of these questions are easy enough to answer. To wit, a legal Folder Action Script:

- Must reside in one of two folders: /Library/Scripts/Folder Action Scripts or ~/Library/Scripts/Folder Action Scripts.
- Must be saved as a script (with a .scpt file extension).
- Must contain a special handler.

In the next section, you are going to create a simple Folder Action Script, but for now, here's a list of the 13 Folder Action Scripts provided by Apple, with a little description of each. If you're interested in learning more about them (and about improving your AppleScript game overall), it's a good idea to examine them to figure out what makes them work.

- **Add-new item alert.scpt:** This script alerts you if a file is added to a folder. This can be very useful for monitoring folders that are shared on the network.
- **Close-close sub-folders.scpt:** This script closes all Finder windows associated with files and folders dropped into the watched folder. It's an easy way to clear your screen if you've got dozens of Finder windows open.
- **Convert-PostScript to PDF.scpt:** This script converts any dropped PostScript (EPS or PS) files to PDF. It's very useful if you're working in Adobe Illustrator, for example, and need to output PostScript files and then immediately check to make sure they look okay.
- **Image-Add Icon.scpt:** This script adds a thumbnail icon to any dropped image.
- **Image-Duplicate as JPEG.scpt:** This script duplicates any dropped image into a JPEG image.
- **Image-Duplicate as PNG.scpt:** This script duplicates any dropped image into a PNG image.
- **Image-Duplicate as TIFF.scpt:** This script duplicates any dropped image into a TIFF image.
- **Image-Flip Horizontal.scpt:** This script flips any dropped image horizontally (see Figure 13.6).
- **Image-Flip Vertical.scpt:** This script flips any dropped image vertically.
- **Image-Rotate Left.scpt:** This script rotates a dropped image to the left.
- **Image-Rotate Right.scpt:** This script rotates a dropped image to the right.
- **Image-Info to Comment.scpt:** This script prompts you to add information about the image to the Spotlight comments field.
- **Image-Show comments in dialog.scpt:** This script displays Spotlight comments for an image in a dialog.

FIGURE 13.6

A portion of the Image – Flip Horizontal Folder Action Script

Creating Your First Folder Action Script

At some point, you'll want to create your own Folder Action Scripts, so I say, "Why wait?" You can build one right now. Lucky for you, creating a Folder Action Script isn't very different from building any other kind of AppleScript.

The easiest way to create your own Folder Action Script is to take an AppleScript that you've already got working and update it to make it work as a Folder Action.

In Chapter 11, you created a subroutine that deleted any files passed to it. Here it is again, just to refresh your memory:

```
on trashIt(itemPath)
    tell application "Finder"
        delete item itemPath
    end tell
end trashIt
```

What I'd like to do is reuse this subroutine in a Folder Action. To do that, I create a new script in Script Editor and add the following handler to it:

```
on adding folder items to folder_ after receiving files_
end adding folder items to
```

This handler lets Mac OS X know your code should run whenever you add files to the folder in question. The folder itself is stored in the `folder_` variable, and the dropped files are represented by the variable `files_`.

Once that's out of the way, you can use a `repeat` loop to process each dropped file:

```
on adding folder items to folder_ after receiving files_
    repeat with file_ in files_
        trashIt(file_)
    end repeat
end adding folder items to
```

Now all you have to do is add the `trashIt()` subroutine. Here's the entire script:

```
on adding folder items to folder_ after receiving files_
    repeat with file_ in files_
        trashIt(file_)
    end repeat
end adding folder items to
on trashIt(itemPath)
    tell application "Finder"
        delete item itemPath
    end tell
end trashIt
```

Once you've got the script finished, you need to save it in the right place.

1. **Choose File➪Save from the menu.**
2. **Navigate to /Library/Scripts/Folder Action Scripts if you want it to be accessible to all accounts on your Mac. Otherwise, place it in the ~/Library/Scripts/Folder Action Scripts folder.**
3. **Give your script a descriptive name and click Save.** Figure 13.7 shows that I've named my new Folder Action Script **delete files**.
4. **Create a new folder on your desktop called DeleteFiles.**
5. **Ctrl+click that folder and choose Folder Actions Setup from the contextual menu (see Figure 13.8).**
6. **In the Folder Actions Setup window, click the + button at the bottom of the left column.**
7. **Add the newly created DeleteFiles folder by selecting it.**

FIGURE 13.7

Saving a Folder Action Script

FIGURE 13.8

Configuring Folder Actions for the DeleteFiles folder

8. When the dialog slides down, prompting you to attach a script, select delete files.scpt from the list, and click Attach (see Figure 13.9).

9. You should now see the DeleteFiles folder highlighted in the left column and the delete files script listed in the right column (see Figure 13.10).

FIGURE 13.9

Attaching the delete files script

FIGURE 13.10

The final product

To test this Folder Action Script, I opened Terminal and entered these commands:

```
Last login: Sun Jul  5 08:28:27 on ttys000
Macintosh:~ myerman$ cd desktop
Macintosh:desktop myerman$ touch testa.txt testb.txt
```

This command sequence created two text files on the desktop. I dragged those two files to the DeleteFiles folder. After a few seconds' pause, the Folder Actions script processed the files and sent them directly to the Trash (see Figure 13.11).

FIGURE 13.11

The test files are successfully deleted.

Creating Folder Action Plug-ins with Automator

Just as you can create a Folder Action Script, you can also create Folder Action plug-ins that make your workflows operate when changes occur within any folder. Being able to save your workflows as Folder Actions brings an entirely new level of functionality to Automator, and to your automation skillsgenerally.

Just about anything you can do in a workflow can be applied to a Folder Action. In other words, if you're only feeling so-so about creating a script-based Folder Action, turn to the more intuitive workflow-creation model offered by Automator and you'll probably find a solution.

In this section, I'm going to guide you through the creation of a simple workflow that will become a Folder Action. The workflow in question is extremely simple — it extracts the text from any dropped PDFs.

To create this workflow, follow these steps:

1. Start Automator or choose File ➪ New from the menu.
2. Click Folder Action and then click Choose (Figure 13.12). When the workflow appears, you'll notice a header above the workflow pane on the right that reads "Folder Action receives files and folders added to" followed by a popup menu.
3. Click the popup menu and click Other (Figure 13.13).
4. Click on Desktop and click New Folder to create a new folder.
5. Name that new folder extract and click Create (Figure 13.14).

FIGURE 13.12

Choosing a template

6. When the folder is created, click Choose to assign it to the workflow.
7. Back in Automator, click PDF in the first column; then drag the Extract PDF Text action to the workflow (Figure 13.15).

FIGURE 13.13

Choosing a folder to assign to the workflow

FIGURE 13.14

Creating a new folder

8. Click the radio button next to *Rich Text*.
9. Save the Output to the desktop.
10. Add a custom header and footer if you like.

FIGURE 13.15

Dragging the Extract Text PDF action to the workflow

11. In Automator, choose File ➪ Save as from the menu.
12. Name your plug-in something appropriate, such as extract-pdf-text (Figure 13.16).

FIGURE 13.16

Saving your folder action workflow

13. Click Save.

To test this Folder Action plug-in, I simply drag a PDF into the extract folder, and a few seconds later, an RTF file is created on the desktop (Figure 13.17).

FIGURE 13.17

Extracting text from a PDF

That's it — now you know how to create both Folder Action workflows and scripts.

Summary

In this chapter, you've learned:

- What folder actions are
- How to enable Folder Actions
- How to create different effects with Folder Actions
- How to create Folder Action Scripts and workflows

CHAPTER 14

AppleScript Studio

In the last few chapters, you've learned how to create Folder Actions, applets, and droplets, folder actions and honed your knowledge of subroutines, loops, and conditionals.

In this chapter, you're going to learn the basics of working with AppleScript Studio, a development environment that allows you to build AppleScript-based applications with graphical user interfaces. You'll learn how to create these interfaces, complete with buttons, text fields, pop-up menus, and other elements, and tie those elements to AppleScript logic on the backend.

AppleScript Studio is a combination of XCode, Interface Builder, and other tools available on your Mac. Starting with Snow Leopard, you'll be working with a specific type of AppleScript/Cocoa combination called AppleScriptObjC. Although you're technically working with some Objective-C/Cocoa in this chapter, there's enough AppleScript in here to keep you in somewhat familiar territory.

If you've ever wanted to go a bit further with AppleScript, here's your chance to create slick applications while at the same time getting a little taste of XCode.

IN THIS CHAPTER

- **What is AppleScript Studio?**
- **How do you access AppleScript Studio?**
- **What can you do with AppleScript Studio?**
- **Pros and cons**
- **Your first AppleScript Studio project**

What Is AppleScript Studio?

According to the official introductory documentation provided by Apple, "AppleScript Studio is a combination of application framework and development environment." What does that mean, exactly?

Well, put simply, you'll be using an offshoot of Xcode (a development environment used by OS X programmers and iPhone developers) to create applications with a sophisticated user interface driven by AppleScript.

Why is this such a big deal? Up to now, you've been creating scripts without much of an interface. Yes, you can use commands such as `choose file` or `display dialog` to prompt users for choices, but overall, these user input devices pop up as needed. There has been no easy way (so far, anyway) to create a single interface in which you ask the user for all the information you need in order to run a script.

For example, if you want to create an archival script, your script can prompt the user for a set of files to archive, then prompt the user for the completed archive's filename, and then actually create the archive.

For most situations, this is more than enough, but some cases call out for a more unified approach, and that's what AppleScript Studio allows you to do. For example, instead of having multiple screens pop up during different parts of the process, asking for bits of information from the user, why not just use one screen that prompts the user for all the information the script needs? That's what AppleScript Studio will allow you to do.

At the same time, you're also going to learn a bit about certain XCode and Objective-C/Cocoa terms, like outlets, actions, interface elements, and the like. If you're already familiar with XCode (for example, you've already created Cocoa or iPhone applications) then this chapter will be a breeze. For the rest of you, though, it'll take a bit to get used to it.

How Do You Access AppleScript Studio?

The easiest way to fire up AppleScript Studio is to open the Developer folder (it's located at the root of your Mac), double-click the Applications folder, and then double-click Xcode (see Figure 14.1).

FIGURE 14.1

Open AppleScript Studio via Xcode

If you don't have a Developer folder, then insert your Snow Leopard installation DVD and install the developer tools from there. You'll find them in the Optional Installs folder. Once you have that open, you can double-click the Xcode.mkpg file to install.

You'll get the current version of Xcode, which at this writing is 3.2. You can also download Xcode from developer.apple.com, but please note that you can't run Xcode 3.1 (which has version specific dependencies on Lepard) in the Snow Leopard environment.

Once you have Xcode open, do yourself a favor and Ctrl+click its icon in the Dock; then choose Options ➪ Keep in Dock from the shortcut menu. This way, you'll have it at your fingertips.

Once Xcode is open, follow these steps:

1. **Click Create a new Xcode project on the welcome screen.** The New Project window opens (see Figure 14.2).
2. **Click Cocoa-AppleScript Application and then click Choose.**

FIGURE 14.2

Choosing a new AppleScript Application project

Every time you start a new project, Xcode prompts you for a name and location in which to save the project. You'll come back to this in a moment. Right now, I need to take you on a brief detour.

What Can You Do with AppleScript Studio?

There's pretty much no limit to what you can do with AppleScript Studio. To get some idea, take a look at the sample applications included in the /Developer/Examples/AppleScript Studio folder. The folder holds over 30 applications, including:

- **Plain Text Editor:** A simple example of a document-based plain-text editor. It can read files in, allow changes, and write the changes back to the file system.
- **Mail Search:** A simple search tool that can look for text strings in To, From, Subject, and Body fields.
- **Simple Shell:** A simple interface wrapper for the `do shell script` command in AppleScript.
- **Currency Converter:** A simple application that converts U.S. dollars into another currency.
- **Command Finder:** A simple search tool that finds `shell` commands that match a user-entered string. The search can be constrained by using a pop-up menu of options (for example, **begins with** or **contains**).

Figure 14.3 shows a screenshot of some of these applications. As you can see, some of them are pretty much indistinguishable from the kinds of slick, professional applications that you can buy for your Mac.

FIGURE 14.3

Some representative AppleScript Studio sample projects

Pros and Cons

Here's a list of pros for using AppleScript Studio:

- **There is no need to learn another language.** You already know AppleScript (or at least, at this point, you better know it!), so your learning curve is minimal. You will need to wrap your AppleScript in a bit of Cocoa, but there isn't enough here to make it a problem.
- **You have more options for modularizing your code.** You could create organized libraries of code that handle specific tasks (such as what happens when you click certain buttons), making it easier for you to create other applications (and maintain the ones that are active).
- **AppleScript Studio gives you complete control over the interface.** AppleScript limits you to three buttons per dialog box, and it's hard to combine different elements into a single unified interface. AppleScript Studio, on the other hand, lets you put together a more designed interface in front of the user. For example, instead of prompting the user for different pieces of information at different times during a process, AppleScript Studio lets you prompt the user all at once.
- **You can create working applications quickly.** Once you get the hang of the drag-and-drop Interface Builder and learn how to attach AppleScript commands, you'll be building all kinds of applications. They can be full-fledged software projects, working prototypes, or anything in between.

Here's a list of cons for using AppleScript Studio:

- **AppleScript Studio applications are more complex then, and therefore bulkier then, AppleScripts.** Once you start adding interface elements, Dock icons, menu bars, and all the rest, your programs will be a bit more complex as compared to simple scripts and workflows.
- **Your AppleScript Studio creations will be more limited in scope.** You can't use them as Folder Actions, applets, or droplets.

Your First AppleScript Studio Project

Okay, enough background, let's get started! For your first project, I'd like to build a simple application that prompts the user for a URL and then opens that Web site with Safari at the click of a button.

The project includes the following UI elements:

- A main window that contains all other UI elements
- A text field that can hold a URL entered by the user
- A clickable button

On the backend, this project needs some very basic AppleScript that opens Safari to the designated URL once the button is clicked. If you've been following along in this Bible, then you know how to do everything in that short sentence:

- You can `activate` Safari.
- You can `open` a window to a specific URL.
- You can wrap both of those commands in a specific handler that the button can use (okay, to be fair, you don't know how to do this part quite yet).

Earlier in this chapter, you started to create a new project, and I took you on a quick detour just as Xcode was prompting you for a project name. Let's continue, shall we?

1. **Enter a name for your new project — call it OpenSafari (see Figure 14.4).**

FIGURE 14.4

Naming your new project

2. **Once you've selected a folder to store your project in, click Save.** The first thing you see when you open AppleScript Studio is the Xcode development environment, which, if this is the first time you've seen it, makes you think twice about going any further (see Figure 14.5). However, most of the stuff that's in Xcode isn't anything to worry about, so just keep following along and you'll be fine.

3. **Find a file called OpenSafariAppDelegate.applescript in the file listing.** You should see a bit of code filled in, starting with a script OpenSafariAppDelegate. Before you do anything else, put the following code inside that block:

```
property urlField: missing value
on openURL_(sender)

end openURL_
```

FIGURE 14.5

The Xcode development environment

The property that you've just set will become an outlet in the interface (more on what an outlet is presently). You're going to tie that outlet to a textfield. The handler called openURL_ will become an action. The action will be assigned to a button. Notice that the handler has an underscore at the end of its name and only one argument? You need both to make this work.

4. **Find a file named MainMenu.xib in the file listing.** That file contains all the instructions for laying out your application's user interface. At the moment, this file is blank.

5. **Double-click he blank file.** I've circled it in Figure 14.6. When you double-click this file, Xcode automatically launches Interface Builder. Whereas Xcode is all about keeping track of all your project files and giving you an environment in which you can code, Interface Builder is all about laying out the user interface for your project.

Note

You might be thinking to yourself that you have no formal training in UI design, that you really don't have time to learn this skill, and who really cares anyway, but I'm telling you that Interface Builder is extremely easy to use. You drag-and-drop different user interface elements onto a canvas, and the program even provides you with on-screen guides so that you line everything up nicely. You'll get the hang of it in no time.

7. **Open Interface Builder.** A series of panels and windows opens up, similar to Figure 14.7. The panel named OpenSafari is the main window for your application. The Library panel contains user interface elements such as buttons, text fields, tabbed interfaces, and more. You can drag elements from the Library panel to the OpenSafari window.

FIGURE 14.6

The MainMenu.xib file

FIGURE 14.7

Interface Builder

8. **With Interface Builder open, do a search (using the search bar at the bottom of the Library panel) for *button*.**
9. **Drag a push button from the Library panel to the OpenSafari window.**
10. **Do a search for *text field* and drag a text field over to the OpenSafari window, just above the button.** You know you've positioned it right when you see a dashed, blue line indicating that these two elements are left-aligned with each other.

    Figure 14.8 shows what your project should look like right now.

FIGURE 14.8

Adding user interface elements

11. **Next, click the button once and rename it to** Open Safari**.** It will need a bit of resizing when you do that.
12. **You can easily resize the button by dragging its right side over to the right until all the text displays.** You'll notice that, as you drag, Interface Builder adds helpful dotted lines to guide you (see Figure 14.9).

FIGURE 14.9

Resizing the push button

13. **You can also resize your window panel — after all, you don't need all that screen real estate!** The completed interface looks like Figure 14.10.

FIGURE 14.10

The completed interface

Now that you've completed the interface, it's time to connect interface elements with the properties and handlers you created on the XCode side:

1. **Ctrl+click the button and drag it to the blue box that's labeled Open Safari Application Delegate.** Don't release any mouse buttons until you drag all the way to that box. A window will open that lists outlets and received actions. An *outlet* is how you attach an interface element with a variable or property.

   Remember that we created a urlField property in XCode? Well, this is where that (and other) property shows up. Right now, though, I want you to focus on received actions. You should see a list of actions in the list, the most important one being openURL (Figure 14.11). This openURL action corresponds to the openURL_ handler you created in XCode.

2. **Select the little circle next to openURL to associate the button with your handler in XCode.**

FIGURE 14.11

Associating your button with an action

3. **Ctrl+click the blue box labeled Open Safari App Delegate and drag to the text field.**

4. **Release your mouse button on the text field.** You'll see a little window popup with the word urlField in it (Figure 14.12).

FIGURE 14.12

Associating your text field with a property

5. **Click that word, and voila, you've now made a connection between all the relevant points of your code:**
   - The urlField property is connected to the text field in your interface
   - The openURL_ handler is connected to the button in your interface

Now that you've created your interface and attached the relevant parts to your backend code, you need to go back to XCode and finish the job. Go back to the Xcode environment and click the file called OpenSafariAppDelegate.applescript. You might have to scroll down to see it.

Let's revisit the openURL_ handler, which currently has nothing in it:

```
on openURL_(sender)
    (*Add your script here.*)
end openURL_
```

This handler accepts a single argument, and I've decided to use the default **sender** here. It basically accepts the standard message from the Cocoa framework saying that it's receiving a message of some sort. Our job now is to fill in some code that will actually do something.

The code you'll fill in will be two very simple lines of code. The first line grabs whatever is in urlField, recasting that object as a text string and setting a variable called myURL with that value. You need to recast as text because, otherwise, Safari will try to open an object, and that won't happen at all. Instead, Safari will just open to its default page, and you'll be wondering why your stuff isn't working. So the second line uses a tell statement to open that URL as a new Safari document.

Here's the code:

```
on openURL_(sender)
  set myURL to urlField's stringValue() as text
  tell application "Safari" to make new document with properties
   {URL:myUrl}
end openURL_
```

One more thing that needs to be done before closing out the chapter. Notice that the text field is blank. You could manually enter a URL (such as http://www.google.com) directly into the interface, or you can set a default value using XCode.

To create a default value for a text string in XCode, you'll need to create a new handler called awakeFromNib() and set your value there. The awakeFromNib() handler is run any time the interface is displayed from startup, so it's a good function for initializing values.

Here's a quick bit of code to set the default value of the text field to Google.com:

```
on awakeFromNib()
  urlField's setStringValue_("http://www.google.com")
end awakeFromNib
```

Figure 14.13 shows the final results — notice that Safari opened the Google Web page, because that's the URL you requested.

FIGURE 14.13

The final AppleScript Studio results

Summary

In this chapter, you've learned:

- What AppleScript Studio is
- What you can build with AppleScript Studio
- What its pros and cons are
- How to create a simple GUI-driven program

Part III

Automation Projects

IN THIS PART

Chapter 15
Ten Automation Projects for Files and Folders

Chapter 16
Ten Automation Projects for Music and Audio Files

Chapter 17
Ten Automation Projects for Photos and Images

Chapter 18
Ten Automation Projects for Text Files

Chapter 19
Ten Custom Automation Projects

CHAPTER 15

Ten Automation Projects for Files and Folders

Welcome to Part III of this Bible, in which you get down to the basics of creating automation projects. In this and upcoming chapters, you're going to walk through a total of fifty automation projects. All of them will feature heavy use of Automator, most of them will also include advanced automation by using AppleScript or shell scripting, and some will feature AppleScript-only alternatives.

In this first chapter, you'll learn how to automate work that is centered on files and folders. All the projects in this section focus on making you productive in this specific area. In some cases, you'll find that what you're learning has already been covered in previous chapters, but that's okay: remember, the chapters in Part 4 are all about practical know-how.

IN THIS CHAPTER

- The projects
- Creating a basic workflow to process specific files
- Converting basic flow to accept any files
- Finding files and folders and renaming them
- Finding files and folders and trashing them
- Creating aliases for files and folders
- Filtering Finder items
- Connecting to a server
- Getting folder contents
- Opening files with the proper application
- Setting Spotlight comments for files and folders

The Projects

All of the projects in this chapter will focus on one area: files and folders. Working with files and folders is part of the standard Workflow choice on the welcome screen, as seen in Figure 15.1.

Working with files and folders is a large part of the work that you do in AppleScript, as you can tell from all the examples in the chapters devoted to AppleScript in this Bible.

FIGURE 15.1

Working with files and folders in Automator

The ten projects I'm going to cover are:

1. Creating a basic workflow to process specific files
2. Converting a basic workflow to accept any files
3. Finding files and folders and renaming them
4. Finding files and folders and trashing them
5. Creating aliases for files and folders
6. Filtering Finder items
7. Connecting to a server
8. Getting folder contents
9. Opening files with the proper application
10. Setting Spotlight comments for files and folders

Within each project, I'll use a similar framework to help you absorb what's going on — think of it as a series of recipes with the same formatting. Here's the framework:

- **Introduction:** A discussion of the problem, issue, or task.
- **Using Automator:** A response to the problem/issue/task using Automator only.
- **Using AppleScript:** A response to the problem/issue/task using AppleScript only. Some projects may not have an AppleScript section.
- **Advanced Topics:** Other possible ways of attacking the problem/issue/task. Note that not every project will contain an advanced topic section.

Creating a Basic Workflow to Process Specific Files

Probably the most common automation task that you'll encounter is one that appears suddenly. You're happily working on something, and suddenly you get an e-mail with eight attachments. The sender of the e-mail is desperately seeking your help with those files, and because that person is your spouse/boss/whatever, it's impossible to turn them away.

So you succumb to the inevitable and save those files in a folder on your desktop. It doesn't matter what you're asked to do; what really matters is that you have to do the same thing over and over to the same group of files. It could be as simple a task as renaming all the files or copying them.

Using Automator

Luckily, Automator gives you a great starting point for working with a specific set of files. Here's how you'd go about doing the work.

1. **Start Automator by clicking its icon on the Dock.** (You did put Automator in the Dock, right?) The template screen displays.
2. **Click Workflow and then click Choose.** You'll see a blank workflow (Figure 15.2).
3. **In the first column under Actions, click Files & Folders, then drag the Get Specified Finder Items action to the workflow.** Your workflow should now look like Figure 15.3.
4. **Click Add to start adding files to the action (see Figure 15.4).**

At this point, you can add any other actions you want to the workflow. It's a great starting point if you have a set of specific tasks that you want to handle for a specific set of files. Although you might not think this situation is common, it's perfect for handling the files you get on a regular basis (such as status reports).

For example, once you have a specific set of files, you might find yourself renaming files, moving or copying the files, deleting the files, or any other kinds of operations. As you work through this chapter, you'll learn more about each of these specific opportunities.

FIGURE 15.2

A blank workflow

FIGURE 15.3

The completed workflow

FIGURE 15.4

Adding files to the workflow action

Using AppleScript

The AppleScript equivalent to this extremely simple Automator workflow is to create a list of files and then do something with that list. Here's one way to do that, using a hard-coded list:

```
set files_ to {"myerman:users:myerman:desktop:todo:runes.pdf", "myerm
    an:users:myerman:desktop:ama-leadership-conference:AMA Expense
    Form.xls"}
```

Of course, you must adjust your paths as needed. This is kind of a clumsy way of doing things, but it might be necessary to do it this way if you're consistently handling the processing of specific files that are spread out in different folders. Don't despair, though; this is just the first lesson, and you have to start somewhere — in the next part, you'll learn how to handle any files.

Advanced topics

If you're familiar with shell scripting, you can do something like the following:

1. **Start the Terminal by going to /Applications/Utilities and double-clicking Terminal.**
2. **At the command line prompt, type:**

```
cat > listing.txt
```

3. **Open a Finder window and drag the files you want into the Terminal window. They'll be converted to paths.**
4. **When you're done dragging files, press Ctrl+C.**
5. **In the terminal window, type `more listing.txt` to see the list of files with paths.** You can now use that list in other contexts, such as emailing a list of files you've been working on to your boss.

Converting a Basic Workflow to Accept Any Files

Now that you know how to create a basic workflow to process a specific set of files, you start using it in different ways to do different tasks. Everyone notices how productive and efficient you are. It seems that no matter what kind of assignment you get, you're the automation hero.

Problem is, after a few months of wearing tights and a cape around the office, people are starting to talk. Oh, and you've got a bunch of specific workflows sitting on your Mac that are becoming difficult to keep organized.

So it's time to start thinking about a more generalized approach to processing files. What you need is for your workflows to prompt the user for files to process. That way, you can create workflows that are based on a specific task (such as renaming files or cropping images) instead of a specific set of files.

Using Automator

Fortunately for you, processing any files with a workflow is such a common task that it's built right in to the Files & Folders starting point. Here's how it works:

1. **Start Automator or choose File➪New from the menu.** The template screen displays.
2. **Click Workflow and then click Choose.**
3. **Drag the Ask for Finder Items action into the workflow (see Figure 15.5).**

FIGURE 15.5

The standard Ask for Finder Items action

4. **Feel free to configure this action with a default starting point, a different type, and multiple selections.** As you can see in Figure 15.6, I've changed the prompt to say **Choose a file or folder**, and I've allowed multiple selections of both files and folders.

FIGURE 15.6

Allowing multiple selections when asking for finder actions

Using AppleScript

The AppleScript equivalent to this simple Automator workflow is the `choose file` command. The simplest version of that command is:

```
set files_ to choose file
```

In this example, the variable `files_` contains any selection made by the user in the dialog. Of course, you'll probably want to add a few other features, such as a custom prompt, a default location, and allowing multiple files:

```
set files_ to ¬
   choose file with prompt ¬
      "Choose file(s):" default location (path to documents folder) ¬
      with multiple selections allowed
```

Advanced topics

It might be useful to consider a more specific case now that you're processing any files in a workflow. For example, how would you create a workflow to ask a user for specific files and then print them?

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Files & Folders under Library.**
4. **Drag Ask for Finder Items to the workflow pane (Figure 15.7).**
5. **Under Actions in the first column, click Utilities.** Drag the Print Finder Items action to the workflow.

FIGURE 15.7

Asking for Finder items and printing them

Another option is to use the AppleScript code you created directly in the workflow.

1. **Start Automator or choose File➪New from the menu.** Make sure you click Workflow.
2. **Type** applescript **in the search bar next to Actions and Variables (see Figure 15.8).**
3. **Drag the Run AppleScript action to the workflow.**
4. **Add your custom code in the space where it says** `(* Your script goes here *)`.
5. **Return whatever variable you set to contain the list of files selected by the user.** As shown in Figure 15.9, I'm returning the variable `files_`. Also notice that I pass this selection of files on to another action.

Once you have a list of files, why waste the opportunity? Why not do something useful right then and there? Again in Figure 15.9, notice that I've added a Label Finder Items action to the workflow.

Any files that I select via AppleScript will now be labeled with whatever color I've designated in the action. To make that action even more dynamic, I could click Options at the bottom of the action and select the Show this action when the workflow runs checkbox. That simple change would allow the user to select whatever color he or she wanted for those files.

FIGURE 15.8

Searching for the Run AppleScript action

FIGURE 15.9

Adding custom AppleScript code

Finding Files and Folders and Renaming Them

Now that you know how to load up a list of files (either by hardcoding the list or by having the workflow prompt you for files), it's time to actually do something useful with those files.

One of the more common tasks you'll encounter is renaming files and folders. You'll get a bunch of files from someone, and there will be all kinds of goofy things in the filenames that you probably won't want to keep, such as spaces and funny characters. Or, you'll find yourself receiving a set of files with the same names as files you already have, but you won't want to overwrite your files just yet. What you'll want to do is add some kind of text (maybe a letter designator, a date, or something else) to the new set of files to keep them distinct from your current files.

Using Automator

Automator allows you to rename files very easily (see Figure 15.10). Here's how you do it:

1. **Start Automator or choose File➪New from the menu.** The template screen displays.

FIGURE 15.10

Adding a Rename Finder Items action

2. **Click Workflow and then click Choose.**
3. **Click Files & Folders under Actions.** Drag the Ask for Finder Items action into the workflow, and then drag Rename Finder Items to the workflow.
4. **Depending on your level of comfort, you can tell Automator to add a Copy Finder Items action to the workflow.** If you choose this option, Automator makes a copy of any files you choose just in case you make a mistake.
5. **In the Rename Finder Items action, you've got a whole bunch of options for renaming your files.** For example, you can add a date or time, add text, change the case of a filename, replace text, and more (see Figure 15.11).

FIGURE 15.11

Options for renaming files

6. **As an example, choose *Add Text* from the pop-up menu, then type** myfiles- **in the text field, and then choose *before name* from the pop-up menu.** Every file processed by the workflow will have its name altered in a specific way: each filename will have myfiles- placed in front of it (see Figure 15.12).

FIGURE 15.12

Adding custom text to filenames

Using AppleScript

Here's where I get to pass along some great stuff without having to code it myself. Your installation of AppleScript comes with a bunch of ready-to-use scripts, and all of them are accessible via the menu bar, but only if you've enabled the Script menu.

To turn on the Script menu, follow these steps:

1. **Go to Utilities ➪ AppleScriptEditor.**
2. **Open the Preferences pane by pressing ⌘+ and clicking the General tab.**
3. **Check the check box next to Show Script menu in menu bar (see Figure 15.13).**

FIGURE 15.13

Enabling the Script menu

Now that the Script menu is enabled, you should see a little scroll icon on the top menu bar. On my system, it's next to the Time Machine logo in the upper-right corner, but it could be different on yours.

To access the scripts in the Script menu, simply click the Script Menu icon.

To use the file renaming script in the Script menu, follow these steps:

1. **Place all the files you want to rename in a single folder.**
2. **Open that folder in the Finder.**
3. **Click the Script Menu icon and navigate to Finder Scripts.**

4. Click Add to File Names (see Figure 15.14).
5. When prompted, type in a text string and click either Prefix or Suffix (see Figure 15.15).

The script runs, adding your text as either a prefix or suffix as specified. As you can see from Figure 15.16, my test files have been renamed with a prefix.

FIGURE 15.14

Using the Add to File Names script in the Script menu

FIGURE 15.15

Using the script to add a prefix or suffix

FIGURE 15.16

The result of the Add to File Names script

Finding Files and Folders and Trashing Them

Another common task you'll probably want to automate is trashing files and folders. Although you can easily do this manually, there will be situations in which you'll want to specify a set of filters to help you determine if a set of files or folders is ready for trashing.

For example, you might want to select files by age, label, or file size to help you make a decision.

Using Automator

To set up a workflow for trashing files based on a specific set of criteria (in this case, files of a certain age and file size), follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Select Files & Folders in Library.**
4. **Drag the Find Finder Items action to the workflow (see Figure 15.17).**

FIGURE 15.17

Using the Find Finder Items action

5. **Choose your Home folder from the Search pop-up menu.**
6. **To find files that were created more than six months ago (because you want to clear out older files), select *Date Created* from the first pop-up menu**, *is before* in the second pop-up menu, and the appropriate date from the text field.
7. **Click the + sign to add one more criterion.**
8. **Choose *size* from the first pop-up menu and *is greater than* from the second pop-up menu.**
9. **Choose an appropriately large file size, such as 20MB.**
10. **In the action search box (it's above the second column of actions to the left of the workflow), type the string trash.**
11. **Drag the Move Finder Items to the Trash action to the workflow (see Figure 15.18).**

When you run this workflow, files older than six months and over a certain size will be moved to the Trash.

FIGURE 15.18

Using Automator to trash files with certain criteria

Using AppleScript

A very simple AppleScript that you could create for deleting files might look like this:

```
tell application "Finder"
    set files_ to ¬
      choose file with multiple selections allowed
    repeat with file_ in files_
      delete file_
    end repeat
end tell
```

The pattern here is simple: use the `choose file` dialog to prompt a user for one or more files to delete, and then delete those files with a `repeat` loop. Of course, you could also make this work at the folder level, using a `choose folder` dialog:

```
tell application "Finder"
    set folder_ to choose folder
    delete every item of folder_
end tell
```

Advanced topics

Remember that AppleScript can also run shell scripts, so you could modify your scripts to include a `do shell script` command that runs `rm`, the UNIX `delete` command. Keep in mind that if you run the `rm -rf` command, you permanently delete your files without sending them to the Trash folder for later review. Also, you need to use POSIX paths to make this work, as the shell doesn't understand the standard colon delimiters that Mac OS X uses in its file paths.

So, essentially, you want to run a variant of this `shell script` command:

```
do shell script "rm -rf " & path_
```

To get that path, you need to convert from a Mac path to POSIX paths:

```
set path_ to quoted form of POSIX path of file_ as string
```

The complete script looks like this:

```
tell application "Finder"
    set files_ to ¬
      choose file with multiple selections allowed
    repeat with file_ in files_
        set path_ to quoted form of POSIX path ¬
          of file_ as string
        do shell script "rm -rf " & path_
    end repeat
end tell
```

Remember that when you run this script, you don't hear the familiar sound associated with adding files to the Trash, nor do you see the files you've selected in the Trash folder. This particular version of the `rm` command deletes the files permanently, with no ability to undo.

Creating Aliases for Files and Folders

You might have a list of files or folders that need to exist in one location, but need pointers to those files and folders from various other locations. If you're used to the Windows world, these are called *shortcuts*, but in the UNIX world of Mac OS X, they're known as *aliases*.

An alias is simply a pointer to an existing file or folder. You can use aliases to simplify how you access files and other resources. For example, instead of memorizing a lengthy path for a file that you need access to, create an alias for it.

The easiest way to create an alias is to Ctrl+click a file or folder and choose Make Alias from the contextual menu. However, if you've got more than one alias to create, doing it manually can be kind of boring.

Using Automator

Here's how you create aliases with Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Select Files & Folders under Library.**
4. **Drag the Ask for Finder Items action to the workflow and then drag the New Aliases action to the workflow (see Figure 15.19).**

FIGURE 15.19

Using the New Aliases action

5. **Make any necessary changes to your actions.** For example, you might want to change the prompt and starting point for the Ask for Finder Items action, or specify that the aliases be created in a specific folder and not on your desktop.

For example, I changed my workflow to only accept folders when asking for Finder items, and then selected my Web Receipts folder to create the alias. The result is shown in Figure 15.20. I can now access the Web Receipts folder directly from the desktop.

FIGURE 15.20

Creating an alias for the Web Receipts folder

Using AppleScript

Creating an alias of a folder is a very simple operation in AppleScript — all you have to do is use the `make alias` command:

```
tell application "Finder"
    set folder_ to choose folder
    make new alias file at desktop to folder_
end tell
```

Advanced topics

If you're familiar with Unix, then you probably know how to use the `ln -s` command to create an symbolic link to a file:

```
ln -s source_file target_file
```

In the above example, the command would create an alias (target_file) to source_file. For example:

```
ln -s myfile.pdf myfile
```

If you ran that command (and if you had a file called myfile.pdf!) you would end up with a symbolic link, an alias if you will, called myfile. Double-click myfile and you would be redirected to myfile.pdf.

To use the `ln` command in AppleScript, just use it in a `do shell script` command. For example, if the file you want to link to is on the desktop and you want the alias to appear in the same place, you can do this:

```
do shell script "ln -s ~/Desktop/file.pdf ~/Desktop/shortcut_file"
```

Of course, this isn't that useful as a script. What you need is some way to prompt the user for a file, convert that file to a path, then prompt the user for the name of the shortcut, then put that shortcut onto the user's desktop.

Here's that script:

```
set file_ to choose file
set path_ to POSIX path of file_
display dialog "Choose name of shortcut:" default answer ""
set shortcut_ to text returned of result
do shell script "ln -s " & path_ & " " & "~/Desktop/" & shortcut_
```

Filtering Finder Items

Some tasks involving files and folders might require you to filter results. For example, you may need to process all text files in one or more folders. You could easily use a series of criteria to find those files or feed those files to a workflow manually or ask the user to just select those files, but none of these choices is very smart.

Instead, Automator's ability to filter files and folders can help you easily get at the files that you want to work with and ignore all the rest.

Using Automator

Here's how you filter Finder items with Automator.

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Select Files & Folders under Library.**
4. **Drag the Ask for Finder Items action to the workflow.** Make sure that the Type pop-up menu is set to Files.
5. **If you want to allow multiple selection, check that check box.**
6. **Drag the Filter Finder Items action to the workflow.**
7. **Change the criteria such that your File Type is Text File (see Figure 15.21).**

FIGURE 15.21

Filtering Finder items

When I run this workflow, I navigate to a folder and then select all files in the folder using ⌘+A. The workflow then selects all files that match the file type (see Figure 15.22).

FIGURE 15.22

Filtering text files

Using AppleScript

If you want to filter by file type in AppleScript, then you need to learn how to use the different Uniform Type Identifiers (UTIs). You use UTIs with choose file dialogs to restrict a user's choice to certain file types.

For example, here's how you restrict choices to .txt files:

```
tell application "Finder"
    set files_ to ¬
      (choose file of type {"public.text"} ¬
      with multiple selections allowed)
end tell
```

The `public.text` designation after the `type` clause is the UTI for .txt files. Various common or widely used file types use the public domain, including the following:

- public.text
- public.plain-text
- public.html
- public.mpeg
- public.tiff
- public.gif
- public.jpeg

Other file formats, such as QuickTime, have identifiers that look like Java designators:

- com.apple.quicktime-movie

You can pass multiple UTIs to a `type` clause by adding more items to the list:

```
tell application "Finder"
    set files_ to ¬
      (choose file of type {"public.html", "public.jpeg"} ¬
        with multiple selections allowed)
end tell
```

Or you can declare the types in a separate named list to make them easier to maintain:

```
tell application "Finder"
    set types_ to {"public.html", "public.jpeg"}
    set files_ to ¬
      (choose file of type types_ ¬
        with multiple selections allowed)
end tell
```

Advanced topics

Decades before anyone had thought of such slick search tools as Spotlight on the Mac, Unix users had the venerable and hard working `find` command. You could use the find command in many different situations, provided you knew which options to use. Needless to say, some of the more advanced options can get pretty esoteric, but you only had to know a little bit to get by.

For example, let's say that you wanted to list out all the directories in your Home folder. You could use this command:

```
find ~ -type d -print
```

When I run that command from Terminal, I start getting a dump of all directories in my Home folder, including every subdirectory encountered. It looks a bit like this:

```
/Users/myerman
/Users/myerman/.adobe
/Users/myerman/.blurb
/Users/myerman/.config
/Users/myerman/.config/gtk-2.0
/Users/myerman/.cups
/Users/myerman/.dvdcss
/Users/myerman/.dvdcss/GIS_FOR_ELECTIONS_DVD#2007111512362000
/Users/myerman/.dvdcss/GIS_FOR_ELECTIONS_DVD-2007111512362000
/Users/myerman/.dvdcss/MY_GREAT_DVD-2008051611420800
/Users/myerman/.dvdcss/TUSAFITNESS-2008081812154000
/Users/myerman/.dvdcss/UNTITLED_DISC-2008093013345500
/Users/myerman/.filezilla
/Users/myerman/.fontconfig
/Users/myerman/.gnupg
/Users/myerman/.JxBrowser
/Users/myerman/.ssh
/Users/myerman/.subversion
/Users/myerman/.subversion/auth
/Users/myerman/.subversion/auth/svn.simple
/Users/myerman/.subversion/auth/svn.ssl.client-passphrase
/Users/myerman/.subversion/auth/svn.ssl.server
/Users/myerman/.subversion/auth/svn.username
/Users/myerman/.thumbnails
/Users/myerman/.thumbnails/normal
```

And so on — it goes on for quite a long time. If you wanted to match for a certain file type, you could use this variant of the command:

```
find ~ -name *.txt -print
```

Again, when I run that command, I start getting a bunch of output:

```
/Users/myerman/.subversion/README.txt
/Users/myerman/Documents/ama/foo.txt
/Users/myerman/Documents/proposals/abbott-clinical/.svn/README.txt
/Users/myerman/Documents/proposals/abbott-ifs/.svn/README.txt
/Users/myerman/Documents/proposals/abbott-jahern/.svn/README.txt
/Users/myerman/Documents/proposals/communicard/.svn/README.txt
/Users/myerman/Documents/proposals/Dexma/.svn/README.txt
```

```
/Users/myerman/Documents/proposals/Dexma/dexma-seo-sem.pages/.svn/
   README.txt
/Users/myerman/Documents/proposals/Dexma/dexma-seo-sem.pages/
   Contents/.svn/README.txt
```

What if you could harness the power of the find command and put it at AppleScript's disposal? You could prompt the user to type in a file type (such as TXT) and then run the find command via `do shell script`.

Here's the script. It's very short, but it does precisely what the previous paragraph describes:

```
display dialog "Choose file type:" default answer "txt"
set filetype_ to text returned of result
do shell script "find ~ -type f -name *." & filetype_ & " -print"
```

Please note that there's no error checking here; if the user enters a string that isn't associated with a known file extension, then the find command won't return anything useful.

Connecting to a Server

If you're at work, or working with a client, you may get the occasional (or not so occasional, depending) request to connect to a remote server. Once connected, you may need to copy files to or from that server, add folders, back up the volume onto your machine, or perform other tasks. If this happens infrequently, there's no need to fret about automation — connect manually and off you go.

However, at some point, this kind of task just begs for automation. It could be that you're connecting to the server once a week or every single day. Or there may be a set of specific tasks that you do with the files every time you copy them to your machine (such as renaming them). Regardless, Automator makes it easy to connect to the server and do the work.

Using Automator

Here's how to connect to a remote server using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Select Files & Folders under Library.**
4. **Drag the Get Specified Servers action to the workflow.**
5. **Click Add and then select a server from the list.**
6. **Drag the Connect to Servers action to the workflow (see Figure 15.23).** When the workflow runs, you're prompted to mount a volume from the server you've connected to.

FIGURE 15.23

Connecting to a server using Automator

Using AppleScript

You can create a very simple AppleScript that connects to a remote server. All you have to know is the URL for that server, for example:

```
tell Application "Finder"
  mount volume "afp://my-remote-server/folder-to-mount"
end tell
```

Make the necessary changes to this script (for example, you need to enter in a real address and folder path), and then save it as an applet and place it on your desktop. The applet opens the remote server every time you double-click it.

Getting Folder Contents

Some days you don't want to pick out individual files, even if you're going to feed them to an Automator workflow. Nor are you in the mood to filter different files from different sources, looking for the proverbial needle in a haystack. No, some days you want all the contents of a folder, no matter what's in there.

Using Automator

Thankfully, Automator has a very handy action that lets you do just that: Get Folder Contents. Here's how to get all folder contents with Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Files & Folder.**

3. **Click Workflow and then click Choose.**
4. **When the workflow opens, click Files & Folders under Library**.
5. **Drag the Ask for Finder Items action to the workflow.** Make sure that you choose Folders as the type.
6. **Drag the Get Folder Contents action to the workflow (see Figure 15.24).** When I run the action and choose a folder in my Documents area, I get a complete list of files and folders in that folder (see Figure 15.25).

FIGURE 15.24

Getting folder contents

FIGURE 15.25

The results of getting folder contents

Using AppleScript

You can use several different approaches in AppleScript to get every item from a folder. One way is to use a `choose folder` dialog to prompt the user to select a folder and then to fill up a list with `every item of` that folder.

Here's the code:

```
set folder_ to choose folder
set files_ to {}
tell application "Finder"
    set files_ to every item of folder_
end tell
```

Notice that at the beginning of the script, I set the `files_` variable to an empty list using `{}`. This empty list gets filled in with whatever files are in the folder that is selected by the user. You end up with a list like this:

```
{document file "my-file1.txt" of folder "sample" of folder "Desktop"
   of folder "myerman" of folder "Users" of startup disk of
   application "Finder", document file "my-file2.txt" of folder
   "sample" of folder "Desktop" of folder "myerman" of folder
   "Users" of startup disk of application "Finder", document file
   "my-file3.txt" of folder "sample" of folder "Desktop" of folder
   "myerman" of folder "Users" of startup disk of application
   "Finder", document file "my-file4.txt" of folder "sample" of
   folder "Desktop" of folder "myerman" of folder "Users" of startup
   disk of application "Finder"}
```

Advanced topics

If you want to see what files are inside of any folders and subfolders discovered by the Automator workflow, then make sure to check the *Repeat for each subfolder found* check box under Get Folder Contents. As you can see in Figure 15.26, this results in a lot more files being discovered.

FIGURE 15.26

Finding files in subfolders

Opening Files with the Proper Application

One day, you get a disk full of files from a coworker, client, or well-meaning soon-to-be ex-friend. Not just five or six, mind you, but dozens of files. Some are text files, and others are images; still others are Keynote files and Flash movies.

You need a quick way to open these different files with their proper applications so that you can figure out what they contain. Although you can easily select all the files with ⌘+A and open them all with ⌘+O, creating a simple workflow will make it easier for you to rerun it whenever you need to.

Using Automator

Here's how to open these files with their proper applications using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **When the workflow opens, click Files & Folders under Library.**
4. **Drag the Ask for Finder Items action to the workflow.** Make sure that you choose Folders as the type and check the check box next to *Allow Multiple Selection*.
5. **Drag the Open Finder Items action to the workflow (see Figure 15.27).**
6. **Select *Default Application* from the Open with pop-up menu.**

FIGURE 15.27

Opening files with their default applications

Using AppleScript

Fortunately for you, opening files in AppleScript is pretty simple. Use the `open` command with an application as an argument, and AppleScript will use that application to open a file. Don't pass in an application name; AppleScript will try to open the file you specify with its default application.

Here's a very simple AppleScript that prompts the user to choose one or more files, and then uses the `open` command to open those files with the proper applications:

```
tell application "Finder"
    set files_ to ¬
      choose file with multiple selections allowed
    repeat with file_ in files_
        open file_
    end repeat
end tell
```

Here again, you have an excuse to create a useful applet/droplet, one that you can either double-click or drop files on to get started. Here's the new code, which I've saved as type Application and placed on the desktop:

```
on open files_
    tell application "Finder"
        repeat with file_ in files_
            open file_
        end repeat
    end tell
end open
on run
    tell application "Finder"
        set files_ to ¬
            choose file with multiple selections allowed
        repeat with file_ in files_
            open file_
        end repeat
    end tell
end run
```

The `on open` portion of the code is what runs when you drop files on the droplet icon. Notice that I've specified `files_` on that line, so I can then simply use `repeat with` to process each file that's dropped.

The `on run` portion of the code is what runs when you double-click the droplet icon — notice that it's no different from the original code — you double-click the icon and choose files to open from a dialog.

Advanced topics

One way to simplify this Automator workflow is to combine it with the Get Folder Contents action item, placing it in between the Ask for Finder Items and Open Finder Items actions. Just make sure that you limit Ask for Finder Items to folders only and then you've got a workflow that will open every single file in a folder with its default application (see Figure 15.28).

FIGURE 15.28

Adding the Get Folder Contents action

Caution

I must warn you that if you use this workflow to open a folder with hundreds of files in it, you're going to potentially launch dozens of different applications. This may or may not slow your Mac down, depending on how much memory/RAM/resources you've got.

Setting Spotlight Comments for Files and Folders

If you've been using your Mac for any time, then you know about adding Spotlight comments and other metadata to your files. These kinds of activities are high reward/high ROI because they make it easier to find, organize, and work with files.

However, the problem is that you don't always remember to add metadata to your files, and when you do remember to do it, you're usually stuck entering your metadata manually, one file at a time.

Using Automator

Here's how to add Spotlight comments to your files using Automator:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **When the workflow opens, click Files & Folders under Library.**

4. Drag the Ask for Finder Items action to the workflow. Choose Files as the type and check the check box next to Allow Multiple Selection.
5. Drag the Set Spotlight Comments for Finder Items action to the workflow.
6. Click the Options button and check the check box next to Show this action when the workflow runs (see Figure 15.29).

FIGURE 15.29

Adding Spotlight comments

When I run the workflow, I select a number of text files on my desktop when prompted and then enter some sample Spotlight comments (see Figure 15.30).

FIGURE 15.30

A sample Spotlight comment

When I click Continue, the rest of the workflow runs, adding Spotlight comments to the files I've selected. When I Ctrl+click one of those files and choose Get Info from the pop-up menu, I can see that my Spotlight comments have been added (see Figure 15.31).

FIGURE 15.31

Ensuring that the Spotlight comments were added

Using AppleScript

Setting Spotlight comments with AppleScript is easily done with the `set comment` command. However, first you should probably add some way for the users to select files (using a choose file dialog) and an easy way to set the Spotlight comment (using a `display dialog` command).

Here's a fairly straightforward script that will do what you want:

```
tell application "Finder"
    set files_ to ¬
      choose file with multiple selections allowed
    display dialog ¬
      "Comment:" default answer "Spotlight comments"
    set comment_ to text returned of result
    repeat with file_ in files_
        set comment of file_ to comment_
    end repeat
end tell
```

When you run the script, you're prompted for one or more files, and then prompted to enter a Spotlight comment. After you enter a comment, the script adds the comment to the files you've specified.

Of course, this is a perfect excuse to create a droplet, which only requires two small changes on your part to put together: first, save it as an application, and then add the `on open` statement. I've also added an error dialog if someone double-clicks the droplet.

```
on run
    display dialog "If you want to add comments to files, then drag
   those files here!" with icon 0
end run
```

```
on open files_
    tell application "Finder"
        display dialog ¬
          "Comment:" default answer "Spotlight comments"
        set comment_ to text returned of result
        repeat with file_ in files_
          set comment of file_ to comment_
        end repeat
    end tell
end open
```

Advanced topics

An easy way to modify the original Automator script is to process a folder full of files at any one time. So instead of asking users to choose multiple files, have them choose a folder. In the Ask for Finder Items action, just make sure that the Type is set to Folders. Then use the Get Folder Contents action to grab all the contents of that folder before setting the Spotlight comments.

This new workflow is shown in Figure 15.32. However, keep in mind that this kind of workflow adds Spotlight comments to any subfolders it finds in your folder.

FIGURE 15.32

Using Get Folder Contents to process all files in a folder

Summary

In this chapter, you've learned how to put together the following automation projects with Automator and AppleScript:

- Creating a basic workflow to process specific files
- Converting a basic workflow to accept any files
- Finding files and folders and renaming them
- Finding files and folders and trashing them
- Creating aliases for files and folders
- Filtering Finder items
- Connecting to a server
- Getting folder contents
- Opening files with the proper application
- Setting Spotlight comments for files and folders

CHAPTER 16

Ten Automation Projects for Music and Audio Files

As with Chapter 15, this chapter will feature heavy use of Automator, AppleScript, and other tools to make your life and work more productive via automation.

In this second chapter of Part III, you'll learn how to automate work that is centered on music and audio files. All the projects in this section focus on making you productive in this specific area. In some cases, you'll find that what you're learning has already been covered in previous chapters, but that's okay.

Remember, the chapters in Part III are all about practical know-how.

IN THIS CHAPTER

- The projects
- Playing a specific iTunes song
- Adding songs to a playlist
- Filtering iTunes songs
- Setting iTunes volume
- Pausing and playing iTunes
- Set information on iTunes songs
- Removing empty playlists
- Changing case of song names
- Converting text to audio files
- Adding audio files to an iPod

The Projects

All of the projects in this chapter focus on one area: audio and music files. If you're like most people, then you probably have a pretty big collection of audio files. Learning how to automate these files will take you a long way toward getting some of your day back.

The ten projects I'm going to cover are as follows:

1. Playing a specific iTunes song
2. Adding songs to a playlist
3. Filtering iTunes songs
4. Setting iTunes volume

5. Pausing iTunes
6. Setting information on iTunes songs
7. Removing empty playlists
8. Changing case of song names
9. Converting text to audio files
10. Adding audio files to an iPod

Within each project, I'll use a similar framework to help you absorb what's going on — think of it as a series of recipes with the same formatting. Here's the framework:

- **Introduction:** A description of the problem, issue, or task you are trying to solve.
- **Using Automator:** A response to the problem/issue/task using Automator only.
- **Using AppleScript:** A response to the problem/issue/task using AppleScript only. Not all projects will require AppleScript.
- **Advanced topics:** Other possible ways of attacking the problem/issue/task. Not all projects will have an advanced topic section.

Playing a Specific iTunes Song

You're in the mood for a specific song. It doesn't matter what that song is, what your taste is, or what culture you live in. Sometimes you've just got to have Mellencamp/Metallica/Marvin Gaye/Dolly Parton/The Who. And specifically, you want to hear that one song that will capture the mood you're in, your immediate situation, or your aspirations.

Using Automator

Here's how to get at that specific song using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose (see Figure 16.1).**
3. **You will see an empty workflow (see Figure 16.2).**
4. **Click Music under Library and then drag the** Ask for Songs action to the workflow (see Figure 16.3).
5. **If you like, enter a new prompt and deselect the check box next to *Allow multiple selection*.**

FIGURE 16.1

Working with a new workflow

FIGURE 16.2

Selecting a specific song or playlist when the workflow runs

FIGURE 16.3

The Ask for Songs action

When you run this workflow, you're presented with a media browser (see Figure 16.4). To actually play your song, follow these steps:

1. **Click iTunes.**
2. **Click the Music library or one of your playlists, or run a search for the song you want to play.**

FIGURE 16.4

The media browser

3. **If you want to play a song while keeping the media browser open:**
   - Double-click a song; or
   - Click a song and click the Play button.
4. **If you want to just play one song, click a song and click Choose.**

Using AppleScript

The simplest way to play a song (or *track*) on iTunes with AppleScript is to use the `play track` command. Here's a simple framework:

```
tell application "iTunes"
   play track "X" of playlist "Y"
end tell
```

You don't have to list the playlist, because all tracks are also listed in the Music library. However, you have to know the name of that specific song and the playlist it's in — and by *knowing*, I mean you really have to know. No shortcuts, abbreviations, or wildcards are allowed. If you want to play a specific track, you have to feed that track's name to AppleScript.

Here's an example:

```
tell application "iTunes"
    play track "Back in Black"
end tell
```

As I've got a nice, big collection of AC/DC songs, AppleScript easily finds *Back in Black* without a problem (see Figure 16.5).

You can also play a track number from a playlist, if you know it:

```
tell application "iTunes"
    play track 12 of playlist "80's Music"
end tell
```

Note

If you're curious, this plays Bruce Hornsby's The Way It Is on my setup. I know; you're getting to know me better than you expected.

FIGURE 16.5

Playing a specific song with AppleScript

Advanced topics

If you just want to play the first song that iTunes finds, then you can do that with just a simple `play` command:

```
tell application "iTunes"
    play
end tell
```

As you can see from Figure 16.6, running that script plays *Gangland* by Iron Maiden. I detect a pattern here. Perhaps iTunes is trying to keep me motivated with great music?

FIGURE 16.6

Playing the first song iTunes finds

You can also explicitly instruct iTunes to play a random track by using `play some track`:

```
tell application "iTunes"
    play some track
end tell
```

Again, this picks a random song from the Music library. To limit the randomizer to a playlist, add a `playlist` clause, but this time use the `tell` command to play:

```
tell application "iTunes"
    tell (some track of playlist "Alternative") to play
end tell
```

Playing that AppleScript just now, I get Cake's *Never There*. A definite change of mood, but the quality is still high (see Figure 16.7).

FIGURE 16.7

Playing a random song from a playlist

One more note before moving on. You could also play a random song from a random playlist, explicitly, like this:

```
tell application "iTunes"
    tell (some track of some playlist) to play
end tell
```

Adding Songs to a Playlist

You've purchased or imported a bunch of songs, and now you want to add them to a new playlist. You could add these files to a new playlist, but you know you'll have other songs that will come along soon that also need to be categorized.

Using Automator

To add songs to a playlist using Automator, follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag an Ask for Songs action to the workflow.**
5. **Drag the Add Songs to Playlist action to the workflow (see Figure 16.8).**

FIGURE 16.8

Adding songs to a playlist

6. **If you want to add songs to a new playlist, choose *New playlist* from the pop-up menu and enter a name for it in the text field.**

When I run the workflow, I'm presented with the media browser. I click the Purchased icon to see what songs I've purchased lately, make a number of selections, and click Choose (see Figure 16.9).

Incidentally, I've hardcoded the name Added from Automator into the new playlist. I did that by entering that name directly into the text field for the Add Songs to Playlist action. As soon as I click Choose, I open iTunes to see my new playlist (see Figure 16.10).

FIGURE 16.9

Choosing which files to add to the playlist

FIGURE 16.10

My new playlist with selected music files

Using AppleScript

You can add songs to a playlist using a whole bunch of different methods, but one that makes a lot of sense is to create an applet that allows you to double-click an icon any time you want to add a currently playing song to a playlist.

For example, let's say you've got a playlist called Current Favorites that contains all your favorite songs. If you're playing a random song in iTunes that you think belongs in this playlist, then this script would add it:

```
set myFavs to "Current Favorites"
tell application "iTunes"
    set currentList to playlist myFavs
    add (get location of current track) to currentList
end tell
```

If you haven't created the playlist, you can just ask the user for a new playlist name and create it for them:

```
display dialog "Create Playlist" default answer "new"
set playlist_ to text returned of result
tell application "iTunes"
    make new user playlist with properties {name:playlist_}
    set currentList to playlist playlist_
    add (get location of current track) to currentList
end tell
```

Advanced topics

If you want to create a more generally useful Automator workflow for adding songs to a new playlist, follow these steps:

1. **In the Add Songs to Playlist action, click Options.**
2. **Check the check box next to Show this action when the workflow runs.**

The next time you run the workflow, you are prompted to enter a name for your new playlist (if that was your choice) or to choose from an existing playlist (also if that was your choice). Figure 16.11 shows the modification in the workflow.

FIGURE 16.11

Modifying the workflow to accept user input for the playlist name

Filtering iTunes Songs

Some days, you just need to filter what you listen to. It doesn't matter if you're playing random songs from your entire library, or only songs from a certain playlist. If the song's too long, too short, or doesn't have certain criteria, you probably don't want to hear it at all.

Using Automator

To filter songs being played on iTunes using Automator, follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag Filter iTunes Items to the workflow (see Figure 16.12).**

FIGURE 16.12

Filtering iTunes items

6. **Make any necessary changes to filter the list of songs. For example, to not play a certain artist, follow these steps:**
   a. Choose Songs from the Filter pop-up menu.
   b. Next to Whose, choose Artist from the first pop-up menu.
   c. Choose is not equal to from the second pop-up menu.
   d. Enter the name of the artist you don't want to hear.
7. **Drag the Start iTunes Playing action to the workflow.**

When this workflow runs, you see the media browser window again. You make your song selections, and then the Filter Sources in iTunes action removes the artist you don't want to hear before handing the list over to iTunes for playing.

For example, for whatever reason, I may not be in the mood for Van Halen at the moment, and so I put their name in as an artist to filter out. When I'm working with the media browser, I choose six songs, two of which are Van Halen songs. As you can see from Figure 16.13, the Van Halen songs appear in the Ask for Songs results, but not in the final list that is played by iTunes.

FIGURE 16.13

Filtering out songs by artist name

Advanced topics

An easy way to make this a more user-friendly workflow is to prompt the user for a filter criterion (see Figure 16.14). For example, in the following workflow, I've changed the filter option to Genre, then clicked Options, and then checked the check box next to Show this action when the workflow runs.

This time, when I run the workflow, I can select songs as before in the media browser, but now I can change the criteria for filtering songs. I can stick with Genre, or I can change to some other criterion, such as Artist, Duration, or Year (see Figure 16.15).

FIGURE 16.14

Prompting the user to filter songs

FIGURE 16.15

Prompting the user to change filtering criteria

Setting iTunes Volume

You're blasting away with your music, but then you need to turn it down because it's time for a phone conference. You could set this kind of thing manually, of course, but there are other cases where you might want to automate volume levels. Perhaps you'd like to blast your tunes really loud at lunch, and then turn it back down once you're back in business mode.

Using Automator

To get information on a song using Automator, follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag Set iTunes Volume to the workflow.**
5. **Make sure you click Options and check the check box next to *Show this action when the workflow runs* (see Figure 16.16).**

FIGURE 16.16

Setting the iTunes volume with Automator

When you run this simple workflow, you' see a simple slider that allows you to adjust the volume of your music (see Figure 16.17). Adjust the slider to the volume you want, and then click Continue.

FIGURE 16.17

The volume adjuster

Using AppleScript

To control your computer's volume with AppleScript, you' need to create a list of values (a good range might be 0-10), and then use the `choose from list` dialog to ask the user to choose a volume level from that list. Finally, you take that user input and use it as an argument to the `set volume` command.

Here's the code:

```
set volumes_ to {"0", "1", "2", "3", "4", "5", "6", "7", ¬
    "8", "9", "10"}
set myVolume to choose from list volumes_ ¬
    with prompt "Choose volume level:"
set volume myVolume
```

Figure 16.18 shows the code and the list of choices.

FIGURE 16.18

Choosing a volume setting from a list

Advanced topics

You could write a very simple AppleScript (you'll call it **quiet**) that sets a certain volume:

```
set volume 3
```

You could then create another script (call it **Loud**) and set a higher volume:

```
set volume 8
```

You can assign these scripts to iCal events to control how loud your music is. For example, you might want to have quiet music during business hours, but once 5 o'clock hits, you want to rock it.

1. Open iCal.
2. Create a new event at 9am.
3. Under repeat, make sure that it runs every day.
4. Under alarm, choose Run script.
5. Browse to the script called quiet and choose it.
6. Click Done when you're finished.

What you end up with is something that looks like Figure 16.19 — an event that runs a custom script (in this case, to turn the volume down to level 3 at 9 every morning).

FIGURE 16.19

Running a script every morning

Pausing and Playing iTunes

Sometimes, lowering the volume isn't enough — you get an interruption in the middle of a great song, and you want to pause the song. As soon as the distraction or annoyance (or person bugging you, whatever) is gone, you can go back to enjoying your music.

Using Automator

To pause or play a song in iTunes using Automator, follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag Pause iTunes to the workflow (see Figure 16.20).**

FIGURE 16.20

Pausing iTunes

Playing this workflow pauses iTunes literally in its tracks (yeah, I know, that was a bad pun), and so it's a good one to save as an application rather than as a workflow. Personally, I'd put it on the desktop to get easy access to the pause feature.

Using AppleScript

The AppleScript that pauses iTunes is pretty simple. Here it is in all its glory:

```
tell application "iTunes"
    pause
end tell
```

That's it! That's all you have to know. Of course, you can get a bit more sophisticated by turning it into a toggle: if it's paused, then tell iTunes to play (and vice versa):

```
tell application "iTunes"
    if player state is paused then
        play
     else
        pause
    end if
end tell
```

Or, you could do something even simpler:

```
tell application "iTunes"
    playpause
end tell
```

Advanced topics

You can execute AppleScripts directly from the command line if you use the `osascript` command. An easy way to learn how to do this is to write a very simple shell script that uses the `playpause` command in AppleScript. Once the script is written, you can control iTunes directly from a Terminal window.

Here's a very basic script:

```
#!/bin/sh
osascript -e 'tell application "iTunes" to playpause' 2>/dev/null
```

Notice the `2>/dev/null` part of the script? It pipes any error output that might arise from the script and drops it into a *bit bucket* — essentially, it's an easy way to take any and all error messages and keep them from showing up on the screen.

Name the script `playpause.sh` and then run this command:

```
chmod +x playpause.sh
```

Now you can play and pause iTunes directly by invoking the command:

```
./playpause.sh
```

Each time you run the script, it will either pause the music (if it's playing) or play music (if it's paused). If iTunes isn't an active application, the `playpause` command will activate iTunes and then play a track.

Setting Information on iTunes Songs

Occasionally, you'll run into a situation in which one or more of the songs in your playlists lacks certain key information. For example, you might have an old CD lying around that you burned a long time ago. All the songs on the CD might be from a certain album, by a certain artist, or from a certain year. None of that information has been saved in iTunes for whatever reason, and now it's time to get this information back into the system.

Using Automator

To set information about a song using Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**

5. **Drag the Set Info of iTunes Songs action to the workflow (see Figure 16.21).**
6. **If you have specific information about the songs being imported, make those selections now.** Or simply check the check boxes for specific information you want to enter, such as song name, artist name, album name, and so on.
7. **If you don't have specific information, be sure to click Options and check the check box next to Show this action when the workflow runs.**

FIGURE 16.21

Setting information about iTunes songs

Using AppleScript

Each track or song in iTunes is an object with various pieces of metadata or properties that can be individually set by using AppleScript. To see the whole list, you just have to open the Dictionary for iTunes.

How do you do that? Well, if you've got AppleScript Editor open, choose Window➪Library. To see the dictionary items for iTunes, double-click iTunes in the list. This opens the dictionary, which looks something like Figure 16.22.

To get an idea of what's scriptable, click iTunes Suite in the first column, and then click *track* in the second column. The third column should fill up with all kinds of properties, including, but not limited to, the following:

- artwork
- album
- album artist
- album rating
- bit rate
- category
- compilation
- composer
- grouping

FIGURE 16.22

Opening the iTunes dictionary

- kind
- lyrics
- played count
- rating
- release date
- sample rate
- size
- start

- time
- track count

Each of these is a property that you can set and get using AppleScript. If you want to know more about a certain property, then refer to the information window that's directly below the three columns along the top of the dictionary (see Figure 16.23).

For example, the `album` property consists of text, and holds the album name of a track. The `bit rate` property is an integer, and tells you what bit rate the track has, in kbps. Date added is a date, description is text that describes the track, and so on.

You could very easily write an AppleScript that allows you to add a comment to any song currently playing. First, use a dialog to capture user input. Then use the `set` command to update the `comment` property of the current track.

FIGURE 16.23

Some properties for the track object

```
display dialog "Add tags:" default answer "tag"
set notes_ to text returned of the result
tell application "iTunes"
    tell the current track
        set the comment to notes_
    end tell
end tell
```

What would be even better is to have the name of the track actually show in the dialog, in case you forget or something. All you have to do is run a `get` command to pull the `name` property out for the current track, but notice that all the code is now within the `tell` block:

```
tell application "iTunes"
    set name_ to (get the name of the current track)
    display dialog "Add tags to " & name_ & ¬
      ":" default answer "tag"
    set notes_ to text returned of the result
    tell the current track
        set the comment to notes_
    end tell
end tell
```

Figure 16.24 shows the code and the dialog that lets you add tags to a current song, complete with the name of the current track!

FIGURE 16.24

Tagging a current track

Removing Empty Playlists

If you work with playlists and songs long enough, you'll get to the point where you accidentally have empty playlists. It won't be something you intend, nor will it always be something you can easily keep track of. However, there's nothing more annoying than having a bunch of empty playlists sitting around with no songs in them.

To remove empty playlists from iTunes using Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Remove Empty Playlists action to the workflow (see Figure 16.25).**

FIGURE 16.25

Removing empty playlists with Automator

By the way, this is probably the safest way to delete playlists. Deleting can be done with AppleScript, but there's no confirmation involved, so deleting a track or playlist means losing whatever you've got in there without much of a chance at redemption.

Changing Case of Song Names

Depending on your preferences, you may want your song names to be a certain way — I don't know; maybe you prefer names to be all lowercase. Or perhaps a CD you own uses ALL CAPS for the song names, which makes you crazy when you look at your display of tracks in iTunes.

Using Automator

To change the case of iTunes songs using Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag the Change Case of Song Names action to the workflow (see Figure 16.26).**
6. **Make sure that you select the right case:**
   - **Title Case:** This Turns All Song Names To Initial Caps
   - **Lower Case:** this turns all song names to lowercase.
   - **Upper Case:** THIS TURNS ALL SONG NAMES TO UPPERCASE.

FIGURE 16.26

Changing the case of song names

When you run this workflow, you should see a list of playlists and songs come up in the media browser. Select your songs and click Choose. Whatever case you've chosen in the workflow converts the song names for you.

Advanced topics

An easy way to make this a more flexible workflow is to click Options under Change Case of Song Names and check the check box next to *Show this action when the workflow runs* (see Figure 16.27).

FIGURE 16.27

The updated workflow

You could also make this kind of Automator workflow work very well by processing a playlist at a time. For example:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag the Change Case of Song Names action to the workflow.**

6. **Make sure that the Find Sources in iTunes action (the first one in the workflow) is set to the following:**
   - Set the Find criterion to Songs.
   - Set the following criteria to Playlist is Library.
   - Click Options and check the check box next to Show this action when the workflow runs (see Figure 16.28).

FIGURE 16.28

Applying case changes to entire playlists

Converting Text to Audio Files

If you're like most Mac users, you know that your Mac can talk. Using the `say` command in AppleScript is probably the oldest trick in the book, and I even mentioned it in one of the early chapters of this book. Well, now you've got a much bigger task — you need to convert an entire text file (such as a readme document or e-mail) to an audio file.

Using Automator

To convert text to audio using Automator, follow these steps:

1. **Start Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Text under Library.**
4. **Drag the Get Contents of TextEdit Document to the workflow.**

5. **Drag the Text to Audio File action to the workflow.**
6. **Choose a voice and a filename for the new audio file.**

The completed workflow should look like Figure 16.29. Keep in mind that it works only if you've got a TextEdit document open on the Mac.

FIGURE 16.29

Saving a text file as audio

Using AppleScript

You already know how to say something with AppleScript, but did you know that you could also save what you say into a file? Here's a basic script to save whatever input a user enters into a sound file:

```
display dialog ¬
    "Say something:" default answer "Something!"
set saying_ to text returned of result
say saying_ using "Alex" saving to ¬
    ":Users:myerman:Desktop:something.aiff"
```

Of course, you need to change the path of the saved file, but you get the general idea. Now, how do you load up an entire text file? The easiest way is to create a subroutine called `readFile()`. This subroutine uses various commands to help you execute what you need to do:

- The subroutine takes as its only argument a file path.
- Use `open for access` to open that file for reading.
- Use the `read` command to read in the entire file, from start to eof (end of file).
- Use the `close access` command to close the file.
- Return whatever text is captured by the `read` command, and pass that along to the `say` command.

Here's the code:

```
set path_ to (choose file)
set saying_ to readFile(path_)
```

```
say saying_ using "Alex" saving to ":Users:myerman:Desktop:something.
   aiff"
on readFile(path_)
    set file_ to (open for access (path_))
    set txt_ to (read file_ for (get eof file_))
    close access file_
    return txt_
end readFile
```

Keep in mind that you need to change the path of the output file to suit your own needs.

Advanced topics

An easy way to make this a more flexible workflow is to click Options under Text to Audio File and check the check box next to Show this action when the workflow runs (see Figure 16.30). Doing this allows you to change filenames, choose different system voices, and choose different locations for saving your file.

FIGURE 16.30

A more flexible Text to Audio File workflow

Adding Audio Files to an iPod

You may want to add a selection of songs to your iPod. You can normally synchronize entire playlists and libraries if you're running the iPod in its default configuration. However, sometimes you need to configure the iPod to manually manage the music and videos on the device. For example, last summer, my wife was learning Spanish via audio files and wanted to manually update her iPod with certain files, each of which represented a different lesson. They had to be copied in the right order.

To make this workflow function properly, you must have an iPod connected to your Mac, it must be mounted in iTunes, and it must be set to manually manage music and videos under iPod settings.

Using Automator

To add specific tracks to an iPod or iPhone using Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag the Add Songs to iPod action to the workflow.**
6. **Drag the Eject iPod action to the workflow.**

The entire workflow looks like Figure 16.31.

FIGURE 16.31

Updating an iPod manually

Using AppleScript

A fairly easy way to update your iPod with AppleScript is by using the `update` command:

```
tell application "iTunes"
    set ipod_ to some source whose name is "My iPod"
    update ipod_
end tell
```

Of course, for this to work, you've got to have an iPod named My iPod attached to your Mac, mounted in iTunes, and all the other things you have to do to transfer music from iTunes to an iPod. This script completely updates the entire iPod without any distinctions.

Advanced topics

Let's say that you want to update your iPod but keep a record of the songs that you've added to the device. You can easily do that by adding a Set Contents of TextEdit action to your workflow.

For example, in the following example, I ask for songs, start playing them, and then set the contents of the list to a TextEdit document.

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag the Start iTunes Playing action to the workflow.**
6. **Drag the Set Contents of TextEdit Document action to the workflow.**

When I run this workflow, whatever songs I select will be played in iTunes, and recorded in a new TextEdit document.

To make something similar happen with an iPod, just add the Set Contents of TextEdit action to the end of the iPod update workflow:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Music under Library.**
4. **Drag the Ask for Songs action to the workflow.**
5. **Drag the Add Songs to iPod action to the workflow.**
6. **Drag the Eject iPod action to the workflow.**
7. **Drag the Set Contents of TextEdit Document action to the workflow.**

Summary

In this chapter, you've learned how to put together the following basic and advanced automation projects with Automator and AppleScript:

- Playing a specific iTunes song
- Adding songs to a playlist
- Filtering iTunes songs
- Setting iTunes volume
- Pausing iTunes
- Setting information on iTunes songs
- Removing empty playlists
- Changing the case of song names
- Converting text to audio files
- Adding audio files to an iPod

CHAPTER 17

Ten Automation Projects for Photos and Images

Now that you've had some time to work with files, folders, music files, and audio files, you're becoming familiar with these mini-projects. It's time, though, to turn your attention to the world of photos and images.

Typically, unless you're a Web designer, illustrator, or other visual creative, you don't really work with photos unless you take a lot of pictures of gatherings, trips, vacations, and loved ones. In any case, you end up having to slog through a lot of files and do tedious, repetitive work with those files: matching colors, cropping, resizing, flipping, removing red-eye, and more. With any luck, the techniques you'll learn in this chapter will help to reduce the tedium.

IN THIS CHAPTER

- The projects
- Applying color changes to groups of images
- Cropping and resizing images
- Creating thumbnails
- Converting images
- Flipping and rotating images
- Finding specific images
- Importing images to iPhoto
- Exporting images from iPhoto
- Reviewing photos in a PDF contact sheet
- Automate taking pictures with a digital camera

The Projects

All of the projects in this chapter focus on one area: images and photos. If you're like most people, you'll end up with a lot of personal photos in your iPhoto collection — and if you're a professional designer, photographer, or illustrator, you'll have even more than the average person, as images are your stock in trade.

The ten projects I'm going to cover are:

1. Applying color changes to groups of images
2. Cropping and resizing images
3. Creating thumbnails
4. Converting images
5. Flipping and rotating images

6. Finding specific images
7. Importing images to iPhoto
8. Exporting images from iPhoto
9. Reviewing photos in PDF Contact Sheet
10. Automating taking pictures with a digital camera

Within each project, I'll use a similar framework to help you absorb what's going on — think of it as a series of recipes with the same framework. Here's the format:

- **Introduction:** A description of the problem, issue, or task you're trying to solve.
- **Using Automator:** A response to the problem/issue/task using Automator only.
- **Using AppleScript:** A response to the problem/issue/task using AppleScript only.
- **Advanced topics:** Other possible ways to attack the problem/issue/task. Not every project will have an advanced topics section.

Applying Color Changes to Groups of Images

You are handed a CD or thumb drive full of images and you're told, "Make all of these black and white, okay?" or, "We need all of these images lightened up a bit." You open the CD and silently thank the gods that you know something about automation, because otherwise, it would take you several days to manually make those kinds of changes to more than 100 images.

Even if you had access to Photoshop, you'd still have to load those images into that environment, which might prevent you from scaling your approach. You might also be in another type of situation, where you're working remotely without all of your design and illustration tools, but you'll still have ready access to Automator and AppleScript. In this chapter, you'll learn how to apply automation to any group of images you might come across.

Using Automator

In Automator, there's a workflow action called Apply ColorSync Profile to Images that allows you to make all kinds of changes to batches of images at the same time.

To apply color changes to groups of images, follow these steps:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose (Figure 17.1).**
3. **You'll see an empty workflow appear (Figure 17.2).**
4. **Click Photos under Library and drag theAsk for Photos action to the workflow**.

FIGURE 17.1

Working with photos and images

FIGURE 17.2

Working with an empty workflow

5. **Drag the Apply ColorSync Profile to Images action to the workflow.**
6. **Copy images when you are prompted to.** Automator automatically adds the Copy Finder Items action to the workflow. It's a good idea to make copies of images in case you make a mistake.
7. **Select an appropriate folder to copy images to.** For example, you can designate the desktop or create a new folder (see Figure 17.3).

FIGURE 17.3

Choosing a folder to copy images into

8. **Select an action from the Profile pop-up menu.** For example, you may want to convert all images to grayscale. You can do that by choosing Profile ➪ Gray Tone from the pop-up menu (see Figure 17.4).

FIGURE 17.4

Applying color changes with a workflow

When this workflow runs, you are prompted to select one or more images for processing. Those images are copied to the folder you designated, and then the workflow applies the changes you specified.

Using AppleScript

Depending on your preferences and budget, you've probably already got a photo editing suite loaded onto your Mac: Adobe Fireworks, Adobe Illustrator, Adobe Photoshop, or any one of the other excellent programs that are available. You've also got three others that you can work with, and this is true of every Mac on the planet:

- **iPhoto,** which is where most of your images are stored and organized, is somewhat scriptable.
- **Preview,** which you've used before to view PDFs and images, is also scriptable. It also contains within it a number of useful image editing tools, such as cropping, resizing, and more.
- **Image Events,** which is only accessible via AppleScript, contains a pretty wide array of scriptable functions for image manipulation (and has the added benefit of not being a memory hog).

Most of what you'll be learning in this chapter will be centered on Image Events. This is mostly because it's available on all Macs and it provides a great deal of functionality even to those (like myself) who are not visual designers with a full range of expensive tools, but who still have to deal with images in their day-to-day work.

To find out more about Image Events, you need to choose Window➪Library from AppleScript Editor's menu selections, and then double-click Image Events. This opens a dictionary screen for Image Events — in other words, every scriptable portion you can work with in AppleScript (see Figure 17.5).

The script suites included in the Image Events dictionary are:

- **Standard Suite**. This suite includes the standard objects and commands used by most scriptable applications. However, unlike other applications, Image Events runs as an invisible process with no user interface, and so only a few commands from this suite, such as close and quit, are relevant.
- **Text Suite**. This default suite is included in most scriptable applications and is used for manipulating text objects. Because Image Events does not currently support any text classes, this suite is not used.
- **Type Definitions Suite**. This suite gives you a bit of scripting control over printing.
- **Disk-Folder-File Suite**. This suite is used for locating files and folders on a disk. The classes and commands from this suite make it possible to locate, open, or delete files and folders using the Image Events application instead of the Finder.
- **Image Suite**. This suite contains the properties and commands for manipulating image files.

FIGURE 17.5

The Image Events dictionary

- **Image Events Suite**. This suite contains the properties of the Image Events application. During the course of this chapter, I'm going to focus mostly on the Image Suite. Presently, however, I'd like to focus on one particular part of that suite, the `match` command. The dictionary definition for this command is:

  ```
  match v : Match an image
  match specifier : the object for the command
  to destination profile : the destination profile for the match
  ```

You're going to be using this command to match an image to a profile, one of which has a color space of Gray. All you have to do is create a droplet that accepts images, processes them one by one, matching each to the preferred profile, and then saves each file to another location, and you'll end up with a powerful script for automating your image manipulation.

First, though, you're going to write a simple script that tells you what color space an image has:

```
set file_ to choose file
tell application "Image Events"
    launch
    set image_ to open file_
    copy the color space of image_ to space_
    close image_
end tell
display dialog "Space: " & (space_ as string)
```

This is a fairly straightforward script that allows the user to choose a file via a selection method, then opens that file, copies the `color space` property to a variable, and then displays that variable in a dialog.

However, changing an image's color space takes a little more doing. Why? Because many properties of an image are read only, which means you need to use the Preview tool to help you automate some processes. Changing an image to grayscale is one of those situations.

Here's a droplet that uses Preview and System Events to process any images you drop on it. The System Events application is how you manipulate menus, buttons, and other user interface elements. Most of this should be pretty straightforward for you, but let's run through the process:

1. **First, use an `on open` statement to accept the images on the droplet.**
2. **Activate Preview, which opens it for viewing.**
3. **Use a `repeat` loop to process each image, using the System Events application to press Option+⌘+C to open the Adjust Color dialog in Preview (see Figure 17.6).**

FIGURE 17.6

Adjusting the color saturation with AppleScript

4. **Set the value of the Saturation slider to 0, removing all color.**

5. **Save your image, and quit Preview when you've processed all files.**

Here's the droplet code:

```
on open images_
    tell application "Preview"
        activate
        repeat with image_ in images_
            open image_
            tell application "System Events"
              keystroke "c" using {option down, command down}
            tell process "Preview"
              set value of value indicator 1 of slider 2 of window 1
   to 0
            end tell
        end tell
        end repeat
        tell application "System Events" to tell process ¬
            "Preview" to click button 1 of window 1
        tell application "System Events"
            keystroke "s" using command down
         end tell
         quit
     end tell
end open
```

Advanced topics

You can easily give the user more options by clicking Options under the Apply ColorSync Profile action and then selecting the check box next to *Show this action when the workflow runs*. That way, the user can select which profile to apply to the selected images.

Cropping and Resizing Images

Another common scenario is having someone walk up to you and say, "Hey, I've got these 300 images that I need resized to 400x400 pixels, can you do that? By tonight?" or, "Hey, here's 500 images, can you crop them so that none of them are wider than 300 pixels? Okay, thanks!"

In the world of image manipulation, there are few tasks more mundane, boring, or repetitive than resizing images. Luckily, you've got a whole bunch of great automation tools at your disposal.

Using Automator

In Automator, there's a workflow action called Crop Images that allows you to resize images to certain dimensions. You can crop by dimension or by percentage, and although in my mind *crop*

means to crop away a part of an image, not resize it, it effectively does both functions in a very efficient manner.

To crop a group of images, follow these steps:

1. **Open Automator or choose File ➪ New from the menu.**
2. **Click Workflow, then click Choose.**
3. **Click Photos under Library, then drag the Ask for Photos action to the workflow.**
4. **Drag the Crop Images action to the workflow.**
5. **When prompted, be sure to add the Copy Finder Items action to the workflow.** I find that when you're working with images, you can't be safe enough.
6. **Choose a target folder for your cropped images.**
7. **Select how to crop your images: by dimension or percentage.**
   - A dimensional crop resizes an image to certain widths and/or heights, and works well if most of your images are the same size to begin with.
   - A percentage crop resizes images by a certain percentage across the board. You might have bigger images mixed in with smaller ones dimension-wise, but they might all get a 10 percent crop vertically and a 20 percent crop horizontally, resulting in different sizes.
8. **If you're choosing a dimensional crop, set your dimensions in pixels for height and width.** If you're selecting a percentage crop, choose width and height percentages from the pop-up menu (see Figure 17.7).

FIGURE 17.7

Cropping images with Automator

Using AppleScript

AppleScript's `scale` command lets you resize an image. You can scale by factor (or percentage) or to a specific size. The `factor` clause requires a single number that represents a percentage — with 1 standing for 100 percent, .5 standing for 50 percent, and so on. The `size` clause requires a single integer that represents how much to resize the longest factor of the image (either height or width, depending on its orientation).

Here's a simple script that scales an image by percentage:

```
set file_ to choose file
tell application "Image Events"
    launch
    set image_ to open file_
    scale image_ by factor 0.5
    save image_
    close image_
end tell
```

The user chooses a file, then Image Events opens that file, scales it to 50 percent, saves it, and closes the file. Figure 17.8 shows what the script did to an amusing holiday photo my wife took of me. Don't ask about the hat, okay?

FIGURE 17.8

Scaling an image 50 percent with AppleScript

Of course, you could also add a dialog that asks a user for an amount to scale the image, like so:

```
set file_ to choose file
display dialog ¬
    "How much to scale? (0.1 to 1.0):" ¬
    default answer "0.5"
set scale_ to text returned of result
tell application "Image Events"
    launch
    set image_ to open file_
    scale image_ by factor scale_
    save image_
    close image_
end tell
```

Here's a script to resize an image by a certain size. It prompts the user to enter a numeric value:

```
set file_ to choose file
display dialog ¬
"How much to resize?" default answer "400"
set size_ to text returned of result
tell application "Image Events"
    launch
    set image_ to open file_
    scale image_ to size size_
    save image_
    close image_
end tell
```

This resizes an image based on the longest side of the image. If the image is taller than it is wide (portrait), then the longest side (the vertical) is resized, and vice versa for wider (landscape) images. So this script is pretty useful, except that it may not work the way you want it to in all cases. You might be surprised to see your images being resized in different ways.

The way to alleviate that problem is to change your script in an important way: calculate which is greater, width or height. Once you know the answer to that question, you can multiply by the size you want and then divide by the other dimension to get your new, scaled height or width.

```
set file_ to choose file
display dialog ¬
    "How much to resize?" default answer "400"
set size_ to text returned of result
tell application "Image Events"
    launch
    set image_ to open file_
    --figure out which is greater, H or W
```

```
        copy dimensions of image_ to {W, H}
        if W < H then
            set scalesize_ to (H * size_) / W
            set scalesize_ to round scalesize_ ¬
              rounding as taught in school
        else
            set scalesize_ to (W * size_) / H
            set scalesize_ to round scalesize_ ¬
              rounding as taught in school
        end if
        scale image_ to size scalesize_
        save image_
        close image_
    end tell
```

Figure 17.9 shows what happens when you run this script twice against the same image.

FIGURE 17.9

Resizing images

The first image is 1600x1200 and is reduced to 533x399 (the height was shorter, and so it was used to calculate the resize) by entering a value of 400. Running this smaller second image through the script with a value of 250 results in an image that is 333x249.

Advanced topics

When you run the basic workflow, you end up with resized images copied to your target folder. Sometimes, though, you need to do a bit of scaling if you're working with different sizes and orientations with your images.

For example, running a dimensional crop workflow against a series of images (all of which start off being different sizes and orientations) results in some interesting crops (see Figure 17.10). That's because some images are bigger than others, while others are portrait instead of landscape.

For example, in Figure 17.10, in the upper left, you can see that the man's head has been completely cropped out except for his face.

FIGURE 17.10

Sometimes, you end up with weird crops

I took that photo at the Austin Pecan Street Festival a few years ago, and I can tell you that not only is the man's entire head (and hat) in that image, so is a sign above his head. In the lower right corner is an image of a dried creek bed, but the most interesting part of the image (a tree) has been left out by the cropping. In the lower left corner, you get only a very small part of a plaque that features prominently at the Texas state capitol. In the upper right, you see the horse, but not the larger context of the street scene.

These are the kinds of things you might run into when you run a cropping algorithm.

One way to minimize this is to process pictures that have similar sizes and orientations, of course, but Automator also gives you the ability to scale your images before the crop (see Figure 17.11).

FIGURE 17.11

Scale before the crop

For example, if you do a dimensional crop, you can choose to:

- **Scale to Width.** This resizes your images to your width setting and then resizes height accordingly.
- **Scale to Height.** This resizes your images to your height setting and then resizes width accordingly.
- **Scale to Long Side.** This resizes images to their long side (that is, they scale to height if they're portrait or to width if they're landscape) before cropping the short side.
- **Scale to Short Side.** This resizes images to their short side (that is, they scale to width if portrait or to height if landscape) before cropping the long side.

When I rerun the workflow on the same images using a long side scale, I get a different result, and a much better set of crops (see Figure 17.12). You can now see the man making balloon animals in the upper left, the building behind the horse, the entire plaque at the Texas state capitol, and the tree leaning over the dry creek bed that's become a trail.

FIGURE 17.12

Using a long side scale

Creating Thumbnails

Another common task you'll often need to perform is creating thumbnails. For example, you' have 300 images that need to go up on the ecommerce site by tonight, and each of them needs a little thumbnail. Fortunately, you can use Automator to speed through this task.

Although this task can easily be accomplished by tools such as Photoshop or Fireworks, there will come a time when you either don't have access to those tools, or need to combine the thumbnail creation process with some other process (such as cropping or resizing), or you'll need to create thumbnails of non-image files (such as PDFs) and for these kinds of situations, Automator offers a compelling substitute.

Using Automator

Here's how to use Automator to create thumbnails:

1. **Open Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**

3. **Click Photos under Library and drag the Ask for Photos action to the workflow.**
4. **Drag the Create Thumbnail Images action to the workflow.**
5. **Choose a size (in pixels) for your thumbnails.**
6. **Choose a suffix for your newly created images (such as** _thumb**), as shown in Figure 17.13.**

FIGURE 17.13

Creating thumbnails with Automator

Advanced topics

The previous workflow is a perfect one to rework as an application. An Automator application is a standalone, self-running workflow. Any files or folders that are dropped onto an application will be used as input for the workflow.

To create an application that creates thumbnails, follow these steps:

1. **Open Automator or choose File ➪ New from the menu.**
2. **Click Application and then click Choose (see Figure 17.14).**
3. **Click Photos under Library and drag the Create Thumbnail Images action to the workflow.** There's no need to use the Ask for Photos action because any received files or folders automatically become input.
4. **Choose a size (in pixels) for your thumbnails.**
5. **Choose a suffix for your newly created images (such as** _thumb **as in Figure 17.15).**
6. **When you save your application, save it to your desktop or other convenient location.**

FIGURE 17.14

Choosing a template for your type of workflow

Whenever you want to create thumbnails, simply drag image files and folders onto your application. Your thumbnail images are created in the same folder as the original images. For example, if your original folder called MyPhotos is on the desktop, and you drag that folder onto the application icon, then your thumbnail images will appear inside that folder.

FIGURE 17.15

Your application for creating thumbnails is complete!

Converting Images

Sometimes you need to convert large numbers of JPEG files to GIFs or GIFs into PNGs. Ordinarily, this kind of work requires that you organize your images, open them one at a time in an image editor, convert them, save them, and move on to the next image.

Well, if that's the way you've handled things in the past, welcome to the world of automation!

Using Automator

To convert images to another file type with Automator, follow these steps:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Photos under Library and drag the Ask for Photos action to the workflow..**
4. **Drag the Change Type of Images action to the workflow. Be sure to use the Copy Finder Items action as well.** (Automator inserts it automatically if you say you want it.)
5. **Choose a type to convert your images to: BMP, JPEG, JPEG 2000, PICT, PNG, or TIFF (see Figure 17.16).**
6. **Choose a suffix for your newly created images (such as** _thumb**).**

FIGURE 17.16

Converting images with Automator

Using AppleScript

The Image Events scripting dictionary gives you access to the `save` command, which allows you to save an image as any of the following file types:

- BMP
- JPEG

- JPEG2
- PICT
- PNG
- PSD
- QuickTime Image
- TIFF

When saving to a new file type, always save to a new file — otherwise, you risk corrupting the original image file.

Here's a simple AppleScript that uses the `save as` command to convert files to GIFs:

```
set file_ to choose file
set the target_ to (choose file name default name "newimage.gif")
set the path_ to the target_ as Unicode text
tell application "Image Events"
    launch
    set image_ to open file_
    save image_ as GIF in path_ with icon
    close image_
end tell
```

Advanced topics

You can easily give the user more options by clicking Options under the Change Type of Images action and then selecting the check box next to Show this action when the workflow runs. That way, the user can select which type to convert images to (see Figure 17.17).

FIGURE 17.17

Giving the user some options

Flipping and Rotating Images

Sometimes you have to flip or rotate a whole bunch of images all at once. Apart from creating thumbnails or resizing images, this is just about the most tedious work since the creation of desktop publishing software. Thankfully, you don't have to put up with the tedium.

Using Automator

To flip images with Automator, follow these steps:

1. **Open Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Photos under Library and drag the Ask for Photos action to the workflow..**
4. **Drag the Flip Images action to the workflow.** Be sure to use the Copy Finder Items action as well. (Automator inserts it automatically if you say you want it.)
5. **Choose how you want your images flipped:**
   - Horizontally
   - Vertically
   - Both

Figure 17.18 shows the flip image workflow.

FIGURE 17.18

Flipping an image with Automator

To rotate an image using Automator, follow these steps:

1. **Open Automator or choose File ➪ New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Photos under Library and drag the Ask for Photos action to the workflow..**
4. **Drag the Rotate Images action to the workflow.** Be sure to use the Copy Finder Items action as well. (Automator inserts it automatically if you say you want it.)
5. **Choose how you want your images rotated:**
   - Left
   - Right
   - 180 degrees

Figure 17.19 shows the rotate image workflow.

FIGURE 17.19

Rotating images with Automator

Using AppleScript

You can easily flip or rotate images using AppleScript commands. To flip an image, use the `flip` command — you can pass in a horizontal or vertical argument to that command to control how the image is flipped. To rotate an image, use the `rotate` command, passing in an angle argument to control the rotation.

Here's a simple AppleScript for flipping an image:

```
set file_ to choose file
tell application "Image Events"
    launch
    set image_ to open file_
    flip image_ with horizontal
    save image_
    close image_
end tell
```

In Figure 17.20, you can see that there are two photos of Kafka, our lovely Husky-something mutt. On the left is the original image, and on the right is the horizontal flipped version of the shot, thanks to this AppleScript.

Rotating an image requires the use of the `rotate` command and a `to angle` clause with an `integer argument`. The integer you pass to the command should be between 1 and 359, and should correspond to the 360 degrees in a circle.

Here's a simple AppleScript that takes user input and then rotates an image by the specified amount:

```
set file_ to choose file
display dialog "Angle of rotation?" default answer "90"
set angle_ to the text returned of result as integer
tell application "Image Events"
    launch
    set image_ to open file_
    rotate image_ to angle angle_
    save image_
    close image_
end tell
```

FIGURE 17.20

Flipping an image is easy with AppleScript

Figure 17.21 shows what happens to the image of Kafka when I rotate it 33 degrees. Notice that AppleScript pads the image with a black background to achieve the rotation if the image isn't rotated at exactly 90, 180, 270, or 360 degrees.

FIGURE 17.21

Rotating an image with AppleScript

Advanced topics

Both flipping and rotating images are good candidates for an Automator application. An Automator application is a standalone, self-running workflow. Any files or folders that are dropped onto an application will be used as input for the workflow.

To create an application that rotates images, follow these steps:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Application and then click Choose.**
3. **Click Photos under Library and drag the Rotate Images action to the workflow.** There's no need to use the Ask for Photos action because any received files or folders automatically become input.
4. **When prompted by Automator, click Add to copy images before rotating them.** While you're at it, choose a destination folder for the copied images.
5. **Choose which direction to rotate the images as a default mode.**
6. **Click Options under the Rotate Images action and select Show this action when the workflow runs (see Figure 17.22).** That way, users can choose their own rotation direction.
7. **When you save your application, save it to a convenient location, like the desktop.**

FIGURE 17.22

Your application for rotating images is completed!

To create an application that flips images, follow these steps:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Application and then click Choose.**
3. **Click Photos under Library and drag the Flip Images action to the workflow.** There's no need to use the Ask for Photos action because any received files or folders automatically become input.
4. **When prompted by Automator, click Add to copy images before rotating them.** While you're at it, choose a destination folder for the copied images.
5. **Choose which direction to flip the images as a default mode.**
6. **Click Options under the Rotate Images action and select Show this action when the workflow runs.** That way, users can choose their own rotation direction.
7. **When you save your application, save it to a convenient location, like the desktop (see Figure 17.23).**

FIGURE 17.23

Your application for flipping images is complete!

Finding Specific Images

If you're anything like me, you probably have as many photos in your collection as you have music files — hundreds if not thousands of images squirreled away inside iPhoto and other places. Trying to find a specific image sounds like a daunting task, but you have some pretty amazing Automator tools at your disposal to help you.

Using Automator

To find specific images with Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose (see Figure 17.24).**
3. **Click Photos under Library and drag the Find Photos in iPhoto action to the workflow.**
4. **Add the criteria you want to run your search.** For example, you could choose Date from the first pop-up menu and choose within last 2 months from the second pop-up menu as criteria (see Figure 17.25).

FIGURE 17.24

Searching photos with Automator

FIGURE 17.25

Finding photos with Automator

Advanced topics

Don't forget that you're not limited to working with iPhoto. By changing your search to the Mac, you end up with a Find Finder Items action in the workflow (see Figure 17.26). As such, this search isn't that much different than a file search.

FIGURE 17.26

Don't forget that you can search other places besides iPhoto

Importing Images to iPhoto

Sometimes you get a whole bunch of images on a CD that need to be imported to iPhoto. Normally, you would open iPhoto and use the menu to import images to an existing photo album (or create a new photo album for the images before importing). Using Automator, you can reduce the number of steps required.

Using Automator

To import photos into iPhoto using Automator, follow these steps:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Files and Folders under Library and drag the Ask for Finder Items action to the workflow.**
4. **Select Allow Multiple Selection if you want to import more than one file at a time.**

5. **Drag the Import Files into iPhoto action to the workflow.**
6. **You can choose to add the photos to an existing album or create a new album altogether.**

In Figure 17.27, I've opted to create a new album called **imported stuff** to hold my imported images.

FIGURE 17.27

Importing images using Automator

Advanced topics

Did you know that you could also grab images from any URL and import them into iPhoto using Automator? Well, you do now! Here's how to do it:

1. **Open Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose (Figure 17.28).**
3. **Click Files and Folders under actions, then drag the** Ask for Servers action to the workflow.
4. **Select the check box next to Allow the user to enter a URL in dialog.**
5. **Click Internet under Library.**
6. **Drag the Get Image URLs from Webpage action into the workflow.**
7. **Drag the Download URLs action into the workflow.**

8. Click Photos under Library.
9. Drag the Import Files into iPhoto action to the workflow.
10. Choose to add the photos to an existing album or create a new album altogether.
11. I would strongly suggest that you select the check box next to Delete the Source Images after Importing Them to keep things neat and tidy in your Downloads folder.

FIGURE 17.28

Starting a new workflow

In Figure 17.29, I'm importing Web photos into a brand new library, and I'm using a variable name that has captured the URL I've entered. I end up with an album name that corresponds to the URL, keeping my work well organized.

FIGURE 17.29

Using a variable-named library to import Web photos

Exporting Images from iPhoto

Now that you've got all your photos organized in iPhoto, you can export your photos from iPhoto so that you can put them on your Web site, burn them to a CD, or add them to Flickr. Instead of manually exporting images, learn how to use Automator to alleviate some of the tedious back-and-forth work you have to do, such as scaling the images and creating thumbnails to go along with them.

To export images using Automator, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Photos under Library.**
4. **Drag the Ask for Photos action to the workflow.**
5. **Drag the Crop Images action to the workflow.** Be sure to accept the Copy Finder Items action to make a copy of anything you work with, and select a working folder for those images.

6. In Crop Images, choose By Percentages in the first pop-up menu, and then create a small crop of 10 percent on both width and height.
7. Drag the Create Thumbnail Images action to the workflow.
8. Select a size for the thumbnails and a file suffix.

In Figure 17.30, you can see my completed workflow.

FIGURE 17.30

Creating an image export workflow with Automator

Reviewing Photos in a PDF Contact Sheet

If you've just come back from a big vacation trip, or you're a professional photographer doing a photo shoot, you sometimes need to look at your images in a contact sheet layout. There's just something about seeing four, eight, or ten images on one sheet to give you some clarity about the kinds of images you've got to work with.

Fortunately for you, Automator has a contact sheet action that does exactly that: it builds a contact sheet with any images you specify.

Using Automator

Here's how to build a contact sheet using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow and then click Choose.**
3. **Click Images under Library, then drag the Find Photos in iPhoto action to the workflow.**
4. **Click PDFs under Library.**
5. **Drag the New PDF Contact Sheet action to the workflow.**
6. **Type a default name for your contact sheet and select a destination to save the PDF.**
7. **Select a paper size.**
8. **Select how many columns to use on your contact sheet.**

Figure 17.31 shows the workflow I've just built.

FIGURE 17.31

Creating a PDF contact sheet with Automator

When I run this workflow on a group of images that I took on a trip about ten years ago, I get the PDF contact sheet shown in Figure 17.32.

FIGURE 17.32

A sample PDF contact sheet

Advanced topics

One way to make this particular workflow more versatile is by asking the user for the name they want for their contact sheet, storing that input in a variable, then using that variable to name the contact sheet in the final step.

As you can see from Figure 17.33, I've done exactly that. I've added the Ask for Text action, allowing me to ask the user for a filename (and it's required, they must answer!). Then I save that filename into a variable called `filename`. Then, I use that variable in the Save As field of the New PDF Contact Sheet action.

FIGURE 17.33

Asking for user input to get a filename for the contact sheet

Automating Taking Pictures with a Digital Camera

The final automation project will be a fun one. Did you know that you could connect a camera to your Mac and then control it with Automator? Knowing this allows you to create all kinds of fun projects, including self-portraits, time-lapse photography, and other applications (security cameras, anyone?).

Using Automator

To take a photo with a camera using Automator, follow these steps:

1. **Attach a digital camera to your Mac. Make sure that the camera is set up properly to take photos.**
2. **Start Automator or choose File➪New from the menu.**

3. **Click Workflow and then click Choose.**
4. **Click Photos under Library.**
5. **Drag the Take Picture action to the workflow.**

Remember, not all digital cameras support this feature. The key seems to be in configuring the USB port — if you can do that, set it to PTP to give the Mac control of the camera. When you run the workflow, the camera takes an image.

Using AppleScript

Unfortunately, the iSight camera that comes with your iMac or MacBook isn't very scriptable, as it doesn't have a standard dictionary. However, it, like most other Mac applications, does respond to the System Events application, which is what you use to manipulate buttons and menu items.

The iSight camera is totally controllable with the Photo Booth application. Inside that application, you can take a snapshot by choosing File➪Take Photo from the menu, or by pressing ⌘+T on the keyboard. Therefore, all you really have to do is activate Photo Booth, and then tell the System Events application to press ⌘+T.

Here's the script:

```
tell application "Photo Booth"
    activate
    tell application "System Events"
        keystroke "t" using {command down}
    end tell
end tell
```

From here, it's not that hard to imagine adding this script to all kinds of applications, or using iCal to run the script at different times of the day. I wish I'd had a Mac five years ago when my wife and I took a long road trip without our dogs and we had pet sitters at home. The way we have our Macs set up now, we'd have had no problem taking photos with this AppleScript at regular intervals and then e-mailing the images to ourselves using Automator.

Summary

In this chapter, you've learned how to put together the following basic and advanced automation projects with Automator and AppleScript:

- Applying color changes to groups of images
- Cropping and resizing images
- Creating thumbnails

- Converting images
- Flipping and rotating images
- Finding specific images
- Importing images to iPhoto
- Exporting images from iPhoto
- Reviewing photos in PDF Contact Sheet
- Automating taking pictures with a digital camera

CHAPTER 18

Ten Automation Projects for Text Files

Working with text files is something that you're probably very used to if you've come from the world of Perl, shell scripting, Ruby, or any of the dozens of other scripting languages with natural text-processing facilities.

You'll find that both Automator and AppleScript give you quite a degree of control over text files. In this chapter, you'll learn how to work with different aspects of text files.

IN THIS CHAPTER

- The projects
- Opening text files
- Asking for text from user
- Getting a specific word
- Getting a specific character
- Getting a specific paragraph
- Combining text files
- Getting the definition of a word
- Using BBEdit: Working with quotes
- Using BBEdit: Convert spaces to tabs
- Using BBEdit: Zapping gremlins

The Projects

All of the projects in this chapter focus on one area: text files. If you're a programmer, business owner, writer, or anyone else who thrives on words, then you'll likely have lots and lots of text files lying around — or a need to convert from some other format to text.

From a practical standpoint, it's a good thing to be able to manipulate not only text files, but also words, paragraphs, and text selections — and believe me, you'll get plenty of that by creating the workflows (see Figure 18.1) in this chapter.

FIGURE 18.1

Choosing a template

The ten projects I'm going to cover are:

1. Opening text files
2. Asking for text from the user
3. Getting a specific word
4. Getting a specific character of text
5. Getting a specific paragraph of text
6. Combining text files
7. Getting the definition of a word
8. Using BBEdit: Working with quotes
9. Using BBEdit: Converting spaces to tabs
10. Using BBEdit: Zapping gremlins

Within each project, I'll use a similar framework to help you absorb what's going on — think of it as a series of recipes with the same formatting. Here's the framework:

- **Introduction:** A description of the problem, issue, or task you're trying to solve.
- **Using Automator:** A response to the problem/issue/task using Automator only. In just a few projects in this chapter, you'll notice that I jump right to AppleScript, skipping Automator entirely. I'm not trying to confuse you, just show you some options that involve different approaches.
- **Using AppleScript:** A response to the problem/issue/task using AppleScript only. Not every project will have an AppleScript section.
- **Advanced topics:** Other possible ways to attack the problem/issue/task. Not every project will have an advanced topics section.

Opening Text Files

You're given a bunch of text files to process — they might be README files, server logs, or just simple e-mail messages saved in the venerable TXT format. It's the kind of job that's just begging for automation. After all, who wants to sit there and manually open text files all day?

In the Leopard version of Automator, you could create a Finder Plug-in that you could access by control-clicking a file or folder. There have been a few changes in Snow Leopard in this area — they're called Services. In this first project, you're going to create a service that allows you to open text files.

Using Automator

To open a text file in Automator:

1. **Start Automator or choose File➪New from the menu if it's already started.**
2. **Click Service and then click Choose (see Figure 18.2).**
3. **Click Files & Folders in the first column under Library.**
4. **Drag the Open Finder Items action to the workflow.** Make sure you choose TextEdit from the drop-down menu (see Figure 18.3). You can also select whether to process just files or folders or both.

FIGURE 18.2

Choosing a Service

FIGURE 18.3

Opening Text Files

5. **Save your work with a descriptive name, such as** open-with-textedit.

You can now run this Automator workflow by Ctrl-clicking any file and choosing open-with-textedit from the menu (or whatever you called your script), as shown in Figure 18.4.

FIGURE 18.4

Your new Finder plug-in in action

Using AppleScript

To open a text file using AppleScript, all you need to do is invoke the `open` command inside of a block that invokes TextEdit. You can also add a bit of code that allows the user to choose a file from a choose dialog:

```
set file_ to choose file
tell application "TextEdit"
    open file_
end tell
```

Advanced topics

If you wanted to strictly limit the AppleScript from the previous section to a specific type of file (such as TXT files), then you would need to add a `type` clause to the `choose file` portion of the script:

```
set file_ to choose file of type {"public.text"}
tell application "TextEdit"
    open file_
end tell
```

Of course, if you regularly use BBEdit (or some other text editor), you can also create a similar script to open a text file:

```
set file_ to choose file
tell application "BBEdit"
    activate
    open file_
end tell
```

Asking for Text from the User

If you're a programmer, you're probably used to asking for user input. Well, Automator and AppleScript are really no different in this regard, as you can ask for text from a user and then do something useful with that input, such as save it to a variable or save it to a file.

Using Automator

To ask for text from a user in Automator:

1. **Start Automator or choose File➪New from the menu if it's already started.**
2. **Click Workflow, then click Choose.**
3. **Click Text in the first column under Library.**
4. **Drag the Ask for Text action to the workflow (see Figure 18.5).**

FIGURE 18.5

Asking for user input

5. **Change the question and answer to something more appropriate to your needs.** For example, in Figure 18.6, I've changed the question to "What is your favorite color?" and the default answer to "Blue." Also, please notice that I've made the answer mandatory.

FIGURE 18.6

Changing the default question and answer

6. **In the first column under Library, click Utilities, then drag the Set Value of Variable action to the workflow.**
7. **Name the variable `user_input` (see Figure 18.7).**

What you now have is a basic workflow that allows you to ask a question, receive an answer, and then store the text of that answer in an Automator variable. You can then reuse that variable elsewhere in your workflow.

You could, for example, use the response to create an event in iCal, run a search of PDFs, or you could use the response to save Spotlight comments for a file. There are lots of possibilities.

Using AppleScript

You should already know how to do this, but I'll remind you anyway (to save you from having to turn so many pages to jog your memory!). Here's a simple script that asks a user for his first name, and then stores that value in a variable:

```
set name_ to the text returned of ¬
   (display dialog ¬
      "What is your first name?" default answer "")
```

Figure 18.8 shows what your dialog would look like.

FIGURE 18.7

Saving `user_input` as a variable

FIGURE 18.8

Asking for user input with AppleScript

Advanced topics

You could make the Automator workflow slightly more useful by actually dumping whatever comes from the user into a new TextEdit document. Here's how:

1. **Start Automator or choose File ⇨ New from the menu if it's already started.**
2. **Click Workflow and then click Choose.**
3. **Click Text under Library and drag the Ask for Text action to the workflow.**
4. **Enter a custom question and answer.**
5. **If the answer is required, select the check box next to Require an answer.**
6. **In the first column under Actions, click Text, and then drag the New TextEdit Document action into the workflow.**

Now, when the workflow runs, you end up with a new, open TextEdit document that contains the user input. Figure 18.9 shows the revamped workflow along with the new open TextEdit instance.

FIGURE 18.9

Saving user input to TextEdit

Getting a Specific Word

Most programming languages allow you to find a specific word using various tools to literally trawl for it in a sea of other words. In this section, I'm going to skip the Automator discussion and focus directly on AppleScript, as there is a lot you can learn here that will make your automation projects go faster.

Using AppleScript

Before you can work with a specific word, let's create a simple snippet of code that opens a text file:

```
set file_ to choose file
read file_
```

This is pretty simple, and should result in any text in the file being captured by the process. In fact, in Figure 18.10, you can see that I've run this script against a file that contains a shell script. The output appears in the bottom pane.

FIGURE 18.10

A simple script for reading text files

Now that you know how to grab text from a file, the next step is to capture a specific word. To do that, it's a good idea to use another variable to store whatever text is captured by the `read` command. Then you can use the `word` command with an integer argument (such as 3, 5, or 10) to get a specific word:

```
set file_ to choose file
set content_ to read file_
get word 10 of content_
```

Here I'm getting word number 10 from the text file's content. As you can see in Figure 18.11, when I run it against the very same shell script file as before, I get the word `echo` returned to me.

This may not throw you at first, but please take a minute and look at the text of that shell script. Here it is again, in plain text:

```
#!/bin/sh
number_of_words=`wc -w *.txt`
echo $number_of_words
```

If the word `echo` is word number 10, then it appears that AppleScript is using a different counting method than you may use! If you thought that, then you'd be absolutely right. AppleScript considers any string of letters or numbers delimited by white space (tab, space, line return) a word. However, the discussion doesn't end there.

FIGURE 18.11

Getting a specific word

Many of the symbols in my sample shell script, such as the equal sign (and other mathematical symbols) and various forms of punctuation (the accent grave right before `wc -w`) are considered words themselves, but other characters, such as the underscore (_) are not.

Advanced topics

Now that you know how to get a specific word from a file, you might want to know how to get a range of words (say, words 1 through 5). You can use the `words` command and pass in a range of numbers, like this:

```
set file_ to choose file
set content_ to read file_
get words 1 thru 5 of content_
```

When I run this script against the same shell script, AppleScript returns a list of individual words, as shown in Figure 18.12. One thing to note in passing: This is a pretty good way of figuring out, interactively, what AppleScript considers a word.

One last bit of fun before moving on: How about grabbing a random word from a file? You've already run into the `some` modifier before, as in `play some track` in iTunes, right? The same thing works here, too:

```
set file_ to choose file
set content_ to read file_
get some word of content_
```

FIGURE 18.12

Getting a list of words

Getting a Specific Character

There will come a day when you get an assignment that's even more specific than grabbing a single word from a file — you'll need to grab a single character (or list of characters) from a text file. Fortunately, everything you've learned so far about getting a word can be applied to the problem of getting a character.

In the last project, I didn't use Automator at all, and the same will be true here. It's just a lot easier to use AppleScript when dealing with specific characters. Of course, you can always create an AppleScript that parses out characters and is inserted into an Automator workflow.

Using AppleScript

You'll be able to reuse a lot of the code that you've already created. For example, you can reuse the code that lets you open and read a text file. Now all you have to do is use the `character` command to actually get a specific character:

```
set file_ to choose file
set content_ to read file_
get character 3 of content_
```

Advanced topics

You can also retrieve multiple characters at once, using the same techniques you used with `words`, this time substituting the `characters` command:

```
set file_ to choose file
set content_ to read file_
get characters 11 thru 19 of content_
```

As expected, this returns a list of characters, which may or may not be what you want:

```
{"n", "u", "m", "b", "e", "r", "_", "o", "f"}
```

You may want a string instead, and AppleScript provides a simple way of giving it to you without making it too difficult:

```
set file_ to choose file
set content_ to read file_
text 11 thru 19 of content_
```

As a comparison, here's what you would expect from this command:

```
"number_of"
```

It's your choice!

Getting a Specific Paragraph

Another common task is to find specific paragraphs within a text file. For example, you might have a log file with blank lines in it — wouldn't it be nice if you could just skip the blank lines? This works because AppleScript and Automator consider a paragraph to be any block of text that's followed by a newline.

Or you might need to find a paragraph that begins with or contains a specific word — that's not a problem either, as you can specify either of these and other combinations as well.

Using Automator

Here's a simple Automator workflow that filters out any blank lines from a text file:

1. **Start Automator or choose File➪New from the menu if it's already started.**
2. **Click Workflow and then click Choose.**
3. **Click Text under Library and drag the Get Contents of TextEdit Document action to the workflow.**
4. **Drag the Filter Paragraphs action to the workflow.** Make sure that you set the pop-up menu to `"are not empty"` to pick out non-empty lines.

5. **Drag the New Text File action to the workflow.**
6. **Type** noblanks.txt **as the name of the file to save as.**

Figure 18.13 shows the workflow you've been building.

FIGURE 18.13

Filtering out empty lines

When you run this workflow, make sure that you've got the file you want to process open in TextEdit. The workflow processes the file, filters out blank lines, and then creates a new text file for you that has only non-blank lines. I've taken a screenshot of the two files side-by-side for comparison (see Figure 18.14).

FIGURE 18.14

The TextEdit file, before and after

Using AppleScript

AppleScript allows you to grab paragraphs of text with the `paragraph` command. Using the same sample text file from the previous Automator workflow, you can use a simple command to grab the first paragraph:

```
set file_ to choose file
set content_ to read file_
get paragraph 1 of content_
```

The result is displayed in Figure 18.15.

FIGURE 18.15

Getting the first paragraph of text with AppleScript

From there, it's fairly easy to grab more than one paragraph at a time:

```
set file_ to choose file
set content_ to read file_
get paragraphs 1 thru 5 of content_
```

Predictably, this returns a list, as you can see in Figure 18.16.

What you really want, though, is an AppleScript that does the same thing as the Automator workflow: Strip out the blank lines, leaving you with just the non-empty lines.

To do that, you need to set up a variable, initially set as "", that contains any future matches. Then you can use a `repeat` loop to go through each paragraph, checking each for a length. If the length of any particular paragraph is greater than 0, then you can add the line to the end of the initial variable you set up.

FIGURE 18.16

Getting more than one paragraph at a time

Here's the code:

```
set file_ to choose file
set content_ to read file_
set paras_ to every paragraph in content_
set nonblank_ to ""
repeat with para_ in paras_
    if length of para_ > 0 then
        set nonblank_ to nonblank_ & return & para_
    end if
end repeat
```

Figure 18.17 shows the code and the result of the run.

FIGURE 18.17

Removing blank lines with AppleScript

Combining Text Files

Sometimes, you find yourself working with a number of text files that need to be combined. They might represent notes, meeting minutes, server logs, or any other kind of textual data — but they do you no good being in separate files.

Instead of manually combining the files together, use either Automator or AppleScript.

Using Automator

To combine text files using Automator:

1. **Start Automator or choose File ➪ New from the menu if it's already started.**
2. **Click Workflow and then click Choose.**
3. **Click Files & Folders under Library and drag the Ask for Finder Items action to the workflow.**
4. **Select the check box next to Allow Multiple Selection.**
5. **In the first column under Library, click Text.**
6. **Drag the Combine Text Files action to the workflow.**

7. **Drag the New Text File action to the workflow.**
8. **Enter an appropriate name in the Save As field, such as** combined.txt **or something similar.**

Your new workflow looks something like Figure 18.18.

FIGURE 18.18

Combining text files with Automator

Using AppleScript

One way to handle this task in AppleScript is to write a script that prompts the user to choose a number of files, then read those files in and combine them. You can use a `repeat` loop to grab the text from each file and concatenate to another variable.

Once you exit the loop, you can prompt the user for a file to save to, and then write the variable you just saved all the text into to that file.

```
set files_ to ¬
     choose file with multiple selections allowed
set combine_ to ""
repeat with file_ in files_
    set content_ to read file_
    set combine_ to combine_ & return & content_
end repeat
```

```
set output_ to choose file name default name "combined.txt"
set reference_ to open for access output_ with write permission
write combine_ to reference_
close access reference_
```

Advanced topics

A better way to handle the AppleScript for combining files is to create a droplet for it. To do that, simply encase the script in an `on open` block, which obviates the need for an initial prompt for text files to process.

```
on open files_
    set combine_ to ""
    repeat with file_ in files_
        set content_ to read file_
        set combine_ to combine_ & return & content_
    end repeat
    set output_ to choose file name default name "combined.txt"
    set reference_ to open for access output_ with write permission
    write combine_ to reference_
    close access reference_
end open
```

Another way to handle the same type of problem is to use a shell script. You can gather each of the names of the files, concatenate them together with the UNIX `cat` command (the shell command that prints out a file), and then pipe the entire thing out to a file on the desktop.

Here's the code:

```
on open files_
    set command_ to "cat "
    repeat with file_ in files_
        set pfile_ to POSIX path of file_
        set command_ to command_ & " " & pfile_
    end repeat
    do shell script command_ & " > ~/Desktop/combined.txt"
end open
```

Whichever option you choose, it's also a good idea to add an `on run` block that gives instructions for how to use the droplet. This block will run if anyone double-clicks the droplet instead of drags files and folders to it:

```
on run
    display dialog "Error! Please drop text files on this droplet to
   combine them!" buttons {"OK"} with icon 0
end run
```

Getting the Definition of a Word

You may have a sudden need to find out what a word means. What better way to do that than to fire up the Dictionary application or Dashboard widget built right into Mac OS X? Well, by letting Automator or AppleScript do it for you!

Using Automator

To look up a word in Automator:

1. **Start Automator or choose File ➪ New from the menu if it's already started.**
2. **Click Workflow and then click Choose.**
3. **Click Text under Library and drag the Ask for Text action to the workflow.**
4. **Enter a custom question and answer, something like** Word to look up? **and make it required.**
5. **Click Utilities under Library, then drag the Get Definition of Word action to the workflow.**

Figure 18.19 shows what your workflow looks like.

FIGURE 18.19

Looking up a word in the Dictionary

Figure 18.20 shows what happens when I look up the word *automation*.

FIGURE 18.20

Looking up a word

Using AppleScript

Unfortunately, Dictionary doesn't have an AppleScript Dictionary (ironic, right?), and so it isn't scriptable at all. However, you can still use the System Events application to do lookups! (See, there's a workaround for everything.)

First, set up a simple dialog to ask the user for a word to look up. Then activate Dictionary, and tell System Events to enter the text from the dialog into the search function of Dictionary and return a result.

```
set query_ to text returned of ¬
    (display dialog ¬
      "Look up in Dictionary" default answer "" buttons {"cancel",
   "look up word"} ¬
      default button 2)
tell application "Dictionary" to activate
tell application "System Events"
    tell process "Dictionary"
        set tf1_ to text field 1 of group 1 of tool bar 1 ¬
          of window "Dictionary"
        set value of tf1_ to ""
        tell tf1_
            keystroke query_
            keystroke return
        end tell
    end tell
end tell
```

When I run this script on *Automation*, Dictionary returns the definition (see Figure 18.21).

FIGURE 18.21

AppleScript finds the entry for *Automation* in Dictionary

Advanced topics

The possibilities for fun here are endless. How about another script that does a lookup in the Thesaurus, Dictionary, and Wikipedia all at once? All you have to do is make sure that you use the `keystroke` command to simulate a user pressing ⌘+0 to search in all services at once:

```
set query_ to text returned of ¬
    (display dialog ¬
      "Look up in Dictionary" default answer "" buttons {"cancel",
   "look up word"} ¬
      default button 2)
tell application "Dictionary" to activate
tell application "System Events"
    tell process "Dictionary"
        set tf1_ to text field 1 of group 1 of tool bar 1 ¬
          of window "Dictionary"
        set value of tf1_ to ""
        keystroke "0" using command down
        tell tf1_
            keystroke query_
            keystroke return
        end tell
    end tell
end tell
```

As you can see in Figure 18.22, Dictionary does a lookup on all the available services. I can't show you the entire screenshot, of course, but if you run this code and scroll through the results, you'll see entries from the Thesaurus, from Wikipedia, and so on.

FIGURE 18.22

Doing a lookup in Dictionary, the Thesaurus, Wikipedia, and other sources

Using BBEdit: Working with Quotes

The next three projects all involve BBEdit. BBEdit is a wonderful, full-featured text editor made by the fine folks over at Bare Bones Software. It's relatively inexpensive, and comes packed with a lot of user features — and bonus, you can access a lot of its services via both Automator and AppleScript.

I immediately became a fan of the tool when I first switched over to Mac from PC. (I needed a cheaper alternative to Dreamweaver that would also give me all the powerful scripting tools that a developer needed beyond just handling HTML.) I hope that these three projects will give you some insight into why it's such a wonderful tool.

If you're a Web developer or designer, then you've probably run into the quotes problem a lot. What's the quotes problem, you might ask? Well, when unwitting content developers and marketers use Microsoft Word to create their Web content, and then save that file as plain text for the developer, they often don't realize what problems they're causing.

You know what I'm talking about: the so-called *smart* quote or *curly* quote that looks so pretty on paper just seems to play havoc with Web browsers. Well, you won't have to worry about that anymore, because BBEdit has a neat set of functions called Educate Quotes and Straighten Quotes.

Note

You must have BBEdit installed on your Mac in order for these projects to work.

Using Automator

Here's how you straighten *smart* quotes in Automator (go from this "" to this ""):

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow in the Template dialog and then click Choose.**
3. **In the first column, click Text.**
4. **Drag the Get Contents of BBEdit Document action to the workflow.**
5. **Make sure that the pop-up menu says Front document.**
6. **Drag the Straighten Quotes action into the workflow.**

Figure 18.23 shows the workflow.

FIGURE 18.23

Straightening quotes with BBEdit

Now, all you have to do is open a file in BBEdit and then run this workflow. What happens is very simple: every smart quote (like you get from a Microsoft Word document: "") is straightened, and therefore ready for use in HTML or a TXT file.

Using AppleScript

Here's an extremely simple AppleScript that prompts the user to open a file, then opens the file in BBEdit and applies the straighten quotes fix:

```
set file_ to choose file
tell application "BBEdit"
   activate
    open file_
    set text_ to read file_
    straighten quotes text_
end tell
```

The `straigthen quotes` command is a simple example of how a third-party software publisher (in this case, Bare Bones) can make available valuable tools for those of you wanting to automate a process.

Using BBEdit: Convert Spaces to Tabs

Another big problem that you might encounter is having someone thoughtfully add a bunch of spaces in a text file in lieu of a simple tab. You'd think that, well into the first decade of the twenty-first century (heck, almost in the second decade of the twenty-first century!), people would just stop doing that, but you'd be wrong.

The following sections show you how to fix that problem with BBEdit.

Using Automator

To convert spaces to tabs in Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow in the Template dialog and then click Choose.**
3. **In the first column, click Text.**
4. **Drag the Get Contents of BBEdit Document action to the workflow.**
5. **Make sure that the pop-up menu says Front document.**
6. **Drag the Convert Spaces to Tabs action into the workflow.**
7. **Set a threshold for how many spaces you want to be converted to a tab.** For example, you might want four consecutive spaces converted to a tab.

Figure 18.24 shows the workflow.

FIGURE 18.24

Converting spaces to tabs

Using AppleScript

Here's a simple AppleScript for converting spaces to tabs. It uses the `entab` command, followed by a `width` clause that stipulates how many spaces get turned into a tab:

```
set file_ to choose file
tell application "BBEdit"
    activate
    open file_
    set text_ to read file_
    entab text_ tab width 4
end tell
```

Using BBEdit: Zapping Gremlins

Another common occurrence? Getting a text file that's full of what BBEdit calls *gremlins*. In other words, any of the host of non-ASCII characters that might confuse a text editor (or Web browser) completely: em dashes, ellipses, funny control characters from word processing applications, and more.

Removing these types of crazy characters is extremely tedious and error-prone, especially if you're dealing with a large file with a lot of content in it — your eyes just won't see some of them, no matter how hard you try.

Using Automator

To facilitate your copy editing efforts, follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **Click Workflow in the Template dialog and then click Choose.**
3. **In the first column, click Text.**

4. **Drag the Get Contents of BBEdit Document action to the workflow.** Make sure that the pop-up menu says Front document.
5. **Drag the Zap Gremlins action into the workflow.**
6. **Be sure to check which kinds of characters you want to process:**
   - Non-ASCII characters
   - Control characters
   - Null (ASCII 0) characters
7. **Be sure to choose what action you want to take from the pop-up menu:**
   - Delete zapped characters (this might not be useful in all cases!)
   - Replace zapped characters with code
   - Replace zapped characters with character

Figure 18.25 shows the workflow.

FIGURE 18.25

Zapping gremlins with Automator

Using AppleScript

Here's a really simple AppleScript for zapping gremlins. Notice that it just deletes the gremlins:

```
set file_ to choose file
tell application "BBEdit"
    activate
    open file_
    set text_ to read file_
    zap gremlins text_ zap action delete_gremlin with nulls, controls
   and non ASCII characters
end tell
```

Summary

In this chapter, you've learned how to put together the following basic and advanced automation projects with Automator and AppleScript:

- Opening text files
- Asking for text from the user
- Getting a specific word
- Getting a specific character of text
- Getting a specific paragraph of text
- Combining text files
- Getting a definition of a word
- Using BBEdit to work with quotes
- Using BBEdit to convert spaces to tabs
- Using BBEdit to zap gremlins

CHAPTER 19

Ten Custom Automation Projects

Now that you've automated files, folders, music files, images, and text files, it's time to spread your wings a bit and take on a few projects that are a bit more challenging. In this chapter, you'll take everything you've learned and start applying it to e-mail messages, PDFs, iCal events, Address Book contacts, and RSS.

IN THIS CHAPTER

- The projects
- Finding specific contacts in Address Book
- Finding people with birthdays
- Creating a group mailer
- Finding specific Calendar items
- Combining Mail messages
- Adding attachments to messages
- Extracting text from PDFs
- Downloading specific URLs using Automator

The Projects

This chapter will be a bit different. Instead of focusing on a specific type of automation task, it's going to have a range of tasks. One minute you'll be working with contacts in Address Book, and the next you'll be extracting text from PDFs. I'll also cover e-mail messages and RSS feeds.

Think of it as a culmination of everything you've learned so far and an opportunity to see the wider vistas of what's possible with automation.

The ten projects I'm going to cover are:

1. Finding specific contacts in Address Book
2. Finding people with birthdays
3. Creating a group mailer
4. Finding specific calendar items
5. Getting new Mail messages
6. Combining Mail messages
7. Adding attachments to messages
8. Extracting text from PDFs

9. Extracting PDF pages
10. Downloading specific URLs

Within each project, I'll use a similar framework to help you absorb what's going on — think of it as a series of recipes with the same formatting. Here's the framework:

- **Introduction:** A description of the problem, issue, or task that you're trying to solve.
- **Using Automator:** A response to the problem/issue/task using Automator only.
- **Using AppleScript:** A response to the problem/issue/task using AppleScript only. In some cases, I don't provide any coverage of AppleScript for a project.
- **Advanced topics:** Other possible ways to attack the problem/issue/task. Some projects don't include an advanced topics section.

Finding Specific Contacts in Address Book

If you're anything like me, then you collect a lot of contacts in Address Book. Trying to find those contacts again can sometimes be difficult to do manually, even if you make heavy use of groups and search functionality.

In any case, what you need to do is find a set of specific contacts in Address Book using automation, and then use that information to do something else, such as populate a text file, start a series of e-mails, or some other task.

Using Automator

To find specific contacts in Address Book using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose (see Figure 19.1).**
3. **In the first column under Library, click Contacts.**
4. **Drag the Find Address Book Items action to the workflow.**
5. **If you're looking for specific people, make sure that you select People in the Find pop-up menu.**
6. **Choose a specific criterion to search on.** You can search on a lot of different fields, including (but not limited to):
   - First Name
   - Last Name
   - Title (as in Mr., Mrs., Ms., Dr.)
   - Job Title
   - Company

FIGURE 19.1

Custom automation projects

7. **In the second pop-up menu, choose a constraint function.** For example, you can choose **contains** or **begins with** — these two give you different limits to your search.
8. **In the text field, enter a search phrase.**
9. **Click the + button to add more criteria to your search.** For example, you might want to match on a specific first name and a specific job title.

In Figure 19.2, you can see that I've run a specific search for contacts that have **manager** somewhere in their job title. The search returned 12 names, which I've obfuscated.

Using AppleScript

The Address Book is completely accessible to AppleScript. To find a person, you construct a search query that will probably remind you of SQL, in a way. Here's a quick sample script that mirrors what I just did with Automator:

```
tell application "Address Book"
    set people_ to every person ¬
      whose job title contains "manager"
end tell
```

FIGURE 19.2

Searching by job title in Automator

As you can see from Figure 19.3, the script returns a list that is identical to what Automator just returned.

FIGURE 19.3

Finding persons in Address Book with AppleScript

Advanced topics

Once you've retrieved a person from Address Book with AppleScript, it's fairly easy to retrieve their properties, such as their e-mail address, phone number, and other information. To do that, use the `properties` command to list out all the properties associated with a specific person, for example:

```
properties of (item 3 of people_)
```

This returns a fairly long result, and includes a VCARD, birthday, image (in hexadecimal notation), modification date, first name, last name, and more. I'll spare you the details — in any case, my good friend Diana (who happens to be item 3 in this list) wouldn't appreciate me divulging her contact information to the world.

By the way, here's a simple little AppleScript that fetches every e-mail address for every person in Address Book:

```
tell application "Address Book"
    get every email of every person
end tell
```

Finding People with Birthdays

I'm really bad about remembering people's birthdays. So bad, in fact, that I usually find myself rushing around the day before (or day of!) trying to buy flowers, cards, gifts, or some other tokens of my esteem, which I really do have, despite my horrible memory.

If you're anything like me, then using Automator and AppleScript to help remember whose birthdays are coming up won't seem trivial — it'll be a valuable tool for enhancing all your relationships, especially if you're in a people-oriented career field, such as sales or real estate, or just enjoy keeping in contact with friends and family

Using Automator

To find people with birthdays using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose.**
3. **Drag the Find People with Birthdays action to the workflow.**
4. **Choose an appropriate reminder threshold from the pop-up menu (see Figure 19.4).** You might be able to get away with a daily reminder, but I need more like a week or month to get my act together.

FIGURE 19.4

Finding people with birthdays

Advanced topics

The Automator workflow that you built at the start of this section is good (and very simple!), but it probably won't do you any good just being a regular workflow. What would be better is to create a Calendar plug-in. That way, you can get a reminder every week of upcoming birthdays for the week.

Here's how to do that:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click iCal Alarm, and then click Choose (Figure 19.5).**

FIGURE 19.5

Saving your workflow as a Calendar plug-in

3. **Drag the Find People with Birthdays action to the workflow.**
4. **Choose an appropriate reminder threshold from the pop-up menu.**
5. **Save your alarm with an appropriate name.** When iCal opens, you see a new event added to the calendar. Notice that it's been added to the Automator calendar for easy reference (see Figure 19.6).
6. **Click Edit to adjust the day when it runs.**
7. **Change the alarm to an appropriate day and time (such as first thing Monday morning).**
8. **Set it to repeat once a week so that you keep getting those reminders.**
9. **Click Done.**

FIGURE 19.6

Adding an Automator alarm to iCal

Figure 19.7 shows the edited event and the repeated alarms every Monday. Now, all that I need to do is faithfully record everyone's birthday in Address Book, and I should be set!

FIGURE 19.7

The completed birthday reminder alarm

Creating a Group Mailer

If you've ever needed to create a group e-mail, then you know how difficult it is. First, you have to gather the e-mail addresses, figure out what you want to say, decide on an e-mail template (if any), and then send out the messages in a timely manner. If you're only sending out 10 or 15 notices, then you're fine.

Once you start getting into the hundreds of e-mails, or having to do frequent mail drops (such as every Friday), you'll soon regret any manual approach you may have taken to get things done. Here's a simple bit of automation goodness that will help both small business owners and extremely busy, active people.

Using Automator

To create a group mailer using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose.**
3. **In the first column under Actions, click Mail.**
4. **Drag the new Mail Message action to the workflow.** Provide a subject line and mail message.
5. **In the first column under Actions, click Contacts.**
6. **Drag the Find Address Book Items action to the workflow.**
7. **Make sure that you've set up proper criteria for finding people.** For example, I'm reusing the search for people with a certain domain name in their email address.
8. **Drag the Group Mailer action to the workflow.**
9. **If you want to add a greeting, select the check box next to *Add Greeting* and then choose what you want the greeting to be.** For example, you can have each greeting start with "Dear" followed by the first name only.

Figure 19.8 shows what this workflow would look like. Please note that the Group Mailer action does not send the outgoing messages, it just queues them up in Mail.

Advanced topics

There are a lot of different ways to improve on this very basic workflow. For example, instead of looking for people using a simple set of criteria, you could set up a smart group in Address Book that indicates who hasn't given you a phone number (I'll call that smart group **NoPhone**).

Then you could create an e-mail that asks them to please respond with their full contact details so you can update them in Address Book.

FIGURE 19.8

Setting up a group mailer in Automator

In the Automator workflow, you could:

1. **Drag the New Mail Message action to the workflow and create an e-mail subject and body.**
2. **Drag the Find Address Book Items action to the workflow.**
3. **Choose People from the Find pop-up menu.**
4. **Under Whose, set the first pop-up menu to Group, the second pop-up menu to** is, **and the third pop-up menu to** NoPhone.
5. **Drag the Group Mailer action to the workflow.**
6. **Make sure that you have an e-mail message ready to go.**

Figure 19.9 shows what this new workflow might look like. What you may end up with, by the way, is a screen full of messages, as shown in Figure 19.10.

If you want to avoid having to manually send all of these messages, simply add the Send Outgoing Messages action to the bottom of your workflow. Be warned, though, that doing this sends your messages without any final approval from you!

FIGURE 19.9

Using smart groups with the Group Mailer function

FIGURE 19.10

A screen full of messages

Finding Specific Calendar Items

It doesn't take very long to get so busy that your iCal starts looking like an impenetrable mess. Although you have various search functions at your disposal, sometimes what you need is a simple bit of automation to help you find what you're looking for.

Using Automator

To find specific calendar items using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose.**
3. **In the first column under Library, click Calendar.**
4. **Drag the Find iCal Items action to the workflow.** (For some reason, it gets renamed to Find Calendars in iCal, just so you know).
5. **Use the pop-up menus to specify what you're looking for.** For example, to find all events by Calendar, choose Events in the Find drop-down menu, and then make sure you're searching for a particular name by choosing Calendar from the next row of pop-up menus, followed by **is** and then the name of your calendar.
6. **Add criteria by clicking the + button and adding a new row.** For example, you may want to establish a Start Date of next week.

In Figure 19.11, you can see that I'm searching for all events in the Invites calendar (this is a special calendar that holds any events to which I've been invited by someone else) that have a start date of **next week**.

FIGURE 19.11

Using Automator to find events in iCal

Using AppleScript

Here's a simple AppleScript that shows all events in a particular calendar (again, I'm using the Invites calendar in iCal) whose start date is greater than today's date:

```
tell application "iCal"
    tell calendar "Invites"
        show (events where start date > (current date))
    end tell
end tell
```

Advanced topics

A very simple way to make the Automator workflow more advanced is to click Options in the Find Calendars in iCal action and then select the Show this action when the workflow runs check box. That way, users can select their own criteria for searching iCal (see Figure 19.12).

FIGURE 19.12

Allowing users to set their own run-time options

Getting New Mail Messages

Although you might think that getting new e-mail is quite easy, think about a situation in which you have multiple accounts. You might have your regular account, another account for mail that goes to your Web site contact form, and a few personal accounts besides.

Wouldn't it be great to automate how you get your mail, especially if you could tie it to a day and time? For example, you may only want to get mail for a certain account at certain times of the day. For those of you who are familiar with Getting Things Done (GTD) you know that one of the gateways to productivity is only checking your email a few times a day, for example, at 11am and 4pm. If you're a GTD person, then Automator can help you!

Using Automator

To filter mail messages using Automator:

1. **Start Automator or choose File ➪ New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose.**
3. **In the first column, under Actions, click Mail.**
4. **Drag the Get New Mail action to the workflow.**
5. **In the For pop-up menu, choose which mail account you want to check.** You can check all accounts at the same time, or specify an individual account for this particular workflow.
6. **If you want to display those messages right away, drag the Display Mail Messages action to the workflow.**

Figure 19.13 shows the new workflow for getting new mail.

FIGURE 19.13

Checking for new mail with Automator

Using AppleScript

Here's a very simple AppleScript that checks for new mail in a specific account (in my case, the account named Gmail — you'll probably need to update this script with your own account):

```
tell application "Mail"
    check for new mail for account "Gmail"
end tell
```

You can change this script to check for new messages on all accounts very easily:

```
tell application "Mail"
    check for new mail
end tell
```

Unlike with Automator, you can also find a "middle ground" by checking many different accounts at the same time:

```
tell application "Mail"
    check for new mail for account "Gmail"
    check for new mail for account "Personal"
    check for new mail for account "Freelance"
end tell
```

Advanced topics

Once you've got a basic AppleScript that checks for new mail, it's very easy to run the script via iCal. Why would you want to do this? Well, the default maximum time increment by which Mail automatically fetches e-mail is one hour.

If you're like me and you don't like to be disturbed with incoming mail all the time, the only other choice you have is to pull mail down manually. If you want to check your e-mail at 9 a.m., noon, and 3 p.m., the easiest way to do it is to set up an iCal-driven system.

Here's how you do it:

1. **Open iCal.**
2. **Double-click tomorrow's date in the calendar view.**
3. **Change the name to** check e-mail **or something similar.**
4. **Click Edit.**
5. **Change the time to an appropriate setting, such as 8:00 a.m.**
6. **For alarm, choose Run Script.** Another pop-up menu opens below the Run Script option.
7. **Click Other, and then choose the script that checks for new mail.**
8. **Under repeat, choose Every Day.**
9. **Click Done when you're finished.**

If you want to have another e-mail check at noon, then you would repeat these instructions for that additional time, creating another series of recurring events. Do yourself a favor and assign all of these events to a calendar called AppleScript so that you can remove them from view at any time, thus keeping your calendar from getting too cluttered.

Figure 19.14 shows the new alarm in action.

If you wanted to create an Automator workflow that is run as an iCal alarm, you'd follow these steps:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click iCal Alarm, and then click Choose.**
3. **In the first column, under Actions, click Mail.**
4. **Drag the Get New Mail action to the workflow.**
5. **In the For pop-up menu, choose which mail account you want to check.** You can check all accounts at the same time, or specify an individual account for this particular workflow.
6. **If you want to display those messages right away, drag the Display Mail Messages action to the workflow.**

7. **When you save your workflow, give it an appropriate name, like check-email.**
8. **When iCal opens, you'll notice that your workflow has been scheduled as an event.** Make any edits you want to it (new start time, new day, recurrence, and so on) to run the workflow when you need to.

FIGURE 19.14

Setting up a recurring check for new mail messages

Combining Mail Messages

If you've ever received five different e-mails from different people (or even the same person!) that describe the same event (such as an event summary, meeting minutes, and so on), then you know how difficult it is to combine those e-mail messages into a single, cogent stream.

Well, Automator can help you combine different e-mail messages into a single message. The section after that provides a bit of advanced discussion.

Using Automator

To combine mail messages using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Workflow, and then click Choose.**
3. **In the first column under Actions, click Mail.**

4. **Drag the Get Selected Mail Items action to the workflow.**
5. **Drag the Combine Mail Messages action to the workflow (see Figure 19.15).**

To make this workflow happen, I need to select a number of messages in Mail, and then run the workflow. For example, I subscribe to a service called Booklert that tells me how sales of my books are doing on Amazon (see, I am an egomaniac!). I get, on average, one e-mail per week per book. Because I want to see all these messages combined in one place, I run the workflow against that group.

FIGURE 19.15

Combining mail messages

Figure 19.16 shows what this result set might look like.

FIGURE 19.16

The results of combining certain mail messages

Advanced topics

If you want to do something useful with all those e-mail messages you just combined, you could simply add a New Mail Message action to the end of the workflow (see Figure 19.17).

Or you could add them to a new text file by adding a New Text File action to the workflow, as I did in Figure 19.18. Doing so would be an easy way to create a simple archival system.

FIGURE 19.17

Appending all those combined messages into a new e-mail

FIGURE 19.18

Combining e-mail messages into a text file

One final trick before I move on. How about sending the text file you just created to the printer? You can do that by adding a Print Finder Items action to the workflow, and then specifying a printer in the Print To pop-up menu (see Figure 19.19).

FIGURE 19.19

Printing the combined e-mail messages

Adding Attachments to Messages

If you've got to send out one e-mail with one attachment, then you can usually get away with doing it manually. If you've got to send hundreds of e-mails with multiple attachments, or if you want to also archive your attachments into a ZIP file before mailing, then you're better off turning to Automator.

Note

Only Automator is used in this project.

To add attachments to mail messages using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Service, and then click Choose (Figure 19.20).**

3. **From the Service receives selected pop-up menu, choose files or folders and in the in pop-up menu, choose Finder.**
4. **Drag the Create Archive action to the workflow.**
5. **Specify the name of the archive as** archive.zip.
6. **In the first column under Library, click Mail.**
7. **Drag the New Mail Message action to the workflow.**
8. **Add a subject line and message body.**
9. **Drag the Add Attachments to Front Message action to the workflow (see Figure 19.21).**
10. **Save your new Service with an appropriate name like attach-files.**

To use this new workflow, open any Finder window, select a group of files you want to attach, Ctrl-click, and select attach-files from the menu (see Figure 19.22).

FIGURE 19.20

Creating a Service

FIGURE 19.21

Attaching files to a mail message

FIGURE 19.22

Your new Finder plug-in in action

When the process is complete, you should have a new e-mail message in front of you with an attached archive.zip file (see Figure 19.23).

FIGURE 19.23

The results of your new Finder plug-in

Extracting Text from PDFs

Here's an unpleasant Monday morning scenario: You get 50 PDFs from your boss, each of which contains different articles that need to be posted to the corporate Web site. Your boss wants to post these PDFs, but she also wants to offer up text-only versions of the article for those who need to run a screen reader.

In the old days, this kind of thing would mean laboriously extracting all the text from each PDF manually. However, because you're an Automator wizard, you can use Automator instead.

Using Automator

To extract text from PDFs using Automator:

1. **Start Automator or choose File ➪ New from the menu.**
2. **In the Template dialog, click Service, and then click Choose.**
3. **From the Service receives selected pop-up menu, choose files or folders and in the in pop-up menu, choose Finder.**

4. **In the first column under Library, click PDFs.**
5. **Drag the Extract PDF Text action to the workflow.**
6. **Choose either plain text or rich text as an output.** If the PDF contains a lot of formatting (including columns, fonts, and other styles), then it might be best to use the rich text option.
7. **Choose a destination for your new text. Don't forget to choose a new name for your creation (see Figure 19.24).**
8. **Choose File➪Save.**
9. **Give your new Service a name.**

FIGURE 19.24

Extracting text from PDF files

To use your new workflow, select one or more PDFs in Finder, and then choose Finder➪Services➪extract-pdf-text (if that's what you named your plug-in. Depending on the size of the PDF (and how many you've selected), the process works, and then creates a file in your desired location.

Figure 19.25 shows a PDF called *The Art of the White Paper* and the rich text extract from the Automator workflow for comparison. As you can see, it's done a pretty good job of grabbing the text from the PDF.

FIGURE 19.25

A sample of what the workflow can do

Advanced topics

How about a quick workflow that extracts PDF annotations? If you've ever had to gather review comments from different PDFs, you know how difficult this is. I recently ran across this problem when I was making comments on a novel that my wife was writing. I needed an easy way to allow her to quickly extract all comments from a PDF so that she could review them in a text file.

Here's how to do that:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Service, and then click Choose.**
3. **From the Service receives selected pop-up menu, choose files or folders and in the in pop-up menu, choose Finder.**
4. **In the first column under Library, click PDFs.**
5. **Drag the Extract PDF Comments action to the workflow.**
6. **Make sure that you select each check box in the Annotation Kinds area.** However, in the Fields to Extract area, you probably only want the Page Label and the Contents, as the contents field usually has the username and modification date.

7. **In the first column under Actions, click New Text File.**
8. **Drag the New Text File action to the workflow.**
9. **Give your new text file a name, such as** comments.txt **(see Figure 19.26).**
10. **Choose File➪Save and give your Service a name.**

FIGURE 19.26

A workflow for extracting PDF annotations

Extracting PDF Pages

Another Monday-morning nightmare scenario: Your boss calls you to let you know that he's sending over 50 PDFs full of screenshots and line art. He'd like each page from these PDFs extracted so that they can be posted in a gallery on the corporate Web site.

Again, Automator comes to the rescue, as it contains an action that lets you easily extract each page in a PDF. Only Automator is used in this project.

To convert PDFs to images using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click Service, and then click Choose.**

3. **From the Service receives selected pop-up menu, choose files or folders and in the in pop-up menu, choose Finder.**
4. **In the first column under Library, click PDFs.**
5. **Drag the PDF to Images action to the workflow.**
6. **Pick a target destination for your images.** I suggest creating a new folder on your desktop (see Figure 19.27).
7. **Choose File➪Save.**
8. **Give your new Service a name.**

FIGURE 19.27

A simple workflow for extracting PDF pages

Downloading Specific URLs

Let's say that every morning, you go to the same five Web sites to get your news. You click around to figure out what's happening in the world, in your town, and in your industry, and after a little while, you toddle off to get the day's work done.

What if you could automate this task, downloading the material that you read, every morning at a specific time? How much more productive would you be if this information was just waiting for you every morning?

To download specific URLs using Automator:

1. **Start Automator or choose File➪New from the menu.**
2. **In the Template dialog, click iCal Alarm, and then click Choose.**

3. **In the first column under Library, click Internet.**
4. **Drag the Get Specified URLs action into the workflow.**
5. **Add specific sites that you follow.**
6. **Drag the Download URLs action to the workflow.**
7. **Don't forget to specify a download folder.** I'd suggest a folder called news on the desktop (see Figure 19.28).

FIGURE 19.28

Grabbing Web content with Automator

8. **Choose File ➪ Save.**
9. **Give your new alarm an appropriate name (such as** download-news**),.**
10. **When iCal pops up with the new event, set the time to an appropriate value (such as first thing in the morning).**
11. **Click Repeat and choose Every day from the pop-up menu.**
12. **Click Done.**

Figure 19.29 shows what your new iCal event looks like when it's all done. You now have a morning script that pulls down the news you want to read, but makes it accessible as an offline copy.

FIGURE 19.29

Adding the workflow as an iCal alarm

Summary

In this chapter, you've learned how to put together the following basic and advanced automation projects with Automator and AppleScript:

- Finding specific contacts in Address Book
- Finding people with birthdays
- Creating a group mailer
- Finding specific calendar items
- Getting new Mail messages
- Combining Mail messages
- Adding attachments to messages
- Extracting text from PDFs
- Extracting PDF pages
- Downloading specific URLs

Part IV

Appendixes

IN THIS PART

Appendix A
Automator Resources

Appendix B
AppleScript Resources

Appendix C
AppleScript Reference

Appendix D
Automator Actions and Variables

APPENDIX

A

Automator Resources

Because you can't possibly learn everything about a subject from just one book (even a book as marvelous as this one!), I decided to put together a few appendixes that offer links and brief reviews of other Automator resources.

In this appendix, you'll find links to tutorials; online communities; tools and downloads; and training. Keep learning, keep searching; it's a big world out there. Also, I've bookmarked many of these resources on my Delicious account. Just go to `www.delicious.com/myerman/automator`.

IN THIS APPENDIX

- **Web-based tutorials**
- **Online community**
- **Tools and downloads**
- **Training**

Web-Based Tutorials

There are a number of excellent Web-based tutorials out there (both free and for a fee) that cover Automator. Here are just a few of them.

VTC Mac OS X Automator tutorials

These tutorials are available at:

`www.vtc.com/products/Mac-OS-X-Automator-tutorials.htm`

If you don't want to type all that, here's a BudURL for it:

`http://budurl.com/lgnd`

The tutorials here are all video-based. The first dozen or so tutorials are free (comprising over 30 minutes of instruction), and the rest of them (over 30) are available for around $100. From what I can tell perusing the free resources, Jesse Feller has created a very nice product. He teaches in a step-by-step way, providing you with a lot of hands-on instruction. The audio and video quality is very good for this format, and you can choose from either QuickTime or Flash media types.

The last time I checked, these tutorials were up to date as of the Leopard release. Figure A.1 shows the tutorial selection page.

FIGURE A.1

Mac OS X Automator tutorials available on VTC.com

Automator.us

Automator.us is an attractive, slick, and extremely useful site. It offers tons of video tutorials, links to other resources, and a valuable podcast RSS so that you can download video tutorials on a regular basis. Figure A.2 shows the home page.

Two valuable sections on this site are Examples (shown in Figure A.3), which contains various learn-by-example tutorials (including taking a video snapshot, using iWork to automate database publishing, and others), and Downloads (shown in Figure A.4), which contains various downloadable workflows and scripts.

FIGURE A.2

The Automator.us home page

FIGURE A.3

The Examples section on Automator.us

FIGURE A.4

Downloadable examples on Automator.us

Online Community

This section details some mailing lists, blogs, and forums devoted to Automator.

Automator World

Automator World (`www.automatorworld.com`) is a must-see destination. Their tag line is "better living through Macintosh scripting," and they really believe it! The site contains workflows, custom actions, news, downloads, interviews, hints, tips, opinion pieces, and more. The Automator World home page is shown in Figure A.5.

If you have a question about Automator, a good place to find an answer is on the Automator World forum, which contains discussions ranging from AppleScript to working with URLs and PDFs to mounting network drives (see Figure A.6).

FIGURE A.5

Automator World

FIGURE A.6

The Automator World discussion forums

Automator-dev mailing list

If you like participating in mailing lists, then join the Automator-dev group at:

`http://lists.apple.com/mailman/listinfo/automator-dev`

You can post questions to the group, receive e-mail, and interact with other Automator developers (see Figure A.7).

FIGURE A.7

The Automator-dev mailing list

Tools and Downloads

Here are a few useful downloads for Automator.

Photoshop Action Pack

If you do a lot of work in Photoshop, then you should download the Photoshop Action Pack at `www.completedigitalphotography.com/index.php?p=339#more-339`.

Or, if you don't want to type all that, try this BudURL:

```
http://budurl.com/yatf
```

The Photoshop Action Pack consists of a series of workflows that help you automate your image processing. It includes filters that operate of EXIF tags, color mode, size, orientation, aspect ratio, and more. It also includes batch operations and branching logic for more complex functionality.

The best part is that it's donationware. If you like it, then you can send the author some money via PayPal.

OttoMate

OttoMate is a set of Automator actions designed to allow automated testing of Web sites. It's currently an open source project available on Google Code:

```
http://code.google.com/p/ottomate/
```

Put simply, OttoMate uses Safari to visit sites, click links and buttons, fill out and submit Web forms, and check for various attributes (titles, text, HTML source, and even JavaScript). Check out the site's interface in Figure A.8.

FIGURE A.8

The OttoMate Project page

Automated Workflows

Automated Workflows is a company run by Ben Waldie. They offer an Automator action pack over at:

`www.automatedworkflows.com/software/automator_actions.html`

Here's the BudURL:

`www.budurl.com/9spr`

The $79 action pack contains extended actions for:

- FileMaker
- Finder
- Illustrator
- InDesign
- Photoshop
- QuarkXPress
- Safari
- Twitter
- TextEdit

There's also a separate action pack available for Twitter for $7.99. Figure A.9 displays some of Workflow's action packs.

FIGURE A.9

The Automated Workflows action packs

Training

Here are some folks who provide Automator training.

Automated Workflows

You're probably thinking — these guys again? Well, I can't help it that Ben Waldie's outfit gives so much to the Automator community. Along with putting together great workflow action packs, doing podcasts, and all the rest, his company also provides Automator training courses. They offer customized classes on AppleScript and Automator, as well as basic introductory classes. You can learn more at:

`www.automatedworkflows.com/training/training.html`

or

`http://budurl.com/b9he`

TECSoft

TECSoft (`www.tecsoft.com`) provides training and workshops for both Automator and AppleScript. They run workshops around the country and offer training software you can purchase.

Summary

In this appendix, you've learned about some tutorials; online communities; tools and downloads; and training. Don't forget to visit my Delicious bookmarks at `www.delicious.com/myerman/automator`.

APPENDIX

B

AppleScript Resources

In Appendix A, I introduced you to a series of Automator resources. I'll do the same thing for AppleScript in this appendix. Here, you'll find links to tutorials, online communities, tools and downloads, and training. I've bookmarked many of these resources on my Delicious account. Just go here:

```
www.delicious.com/myerman/applescript
```

IN THIS APPENDIX

- **Web-based tutorials**
- **Online community**
- **Tools and downloads**
- **Training**

Web-Based Tutorials

There are a number of excellent Web-based tutorials out there (both free and for a fee) that cover AppleScript. Here are just a few of them.

MacResearch AppleScript tutorials

The MacResearch AppleScript tutorials are billed as tutorials for scientists, and so you'll see plenty of AppleScripts that are focused on automating scientific research. Don't be alarmed, though, because scientists tend to do a lot of repetitive stuff that you'll find very familiar.

These tutorials are available at:

```
www.macresearch.org/applescript_tutorials
```

If you don't want to type all that, here's a BudURL for it:

```
www.budurl.com/urvp
```

The tutorials on this site are all text-based, with plenty of screen shots and instructional text to keep you from getting lost. Figure B.1 is a screenshot of the tutorial selection page.

FIGURE B.1

AppleScript tutorials at MacResearch.org

Figure B.2 shows part of a tutorial.

FIGURE B.2

A tutorial on MacResearch.org

MacScripter

The folks at MacScripter have a pretty good AppleScript tutorial section at:

`http://macscripter.net/?sourcebooktips/AS4AS.html`

Figure B.3 shows the home page.

An extremely valuable part of this site is ScriptBuilders (which you can actually get to via `www.scriptbuilders.net`). This is MacScripter's growing archive of AppleScripts (see Figure B.4). Incidentally, you'll also find plenty of Automator workflows and actions here.

FIGURE B.3

The MacScripter AppleScript area

Each script comes with an information page, which tells you who submitted the code, what it does, what you need to run it, and more. Figure B.5 shows the information page for Get_iTunes_Lyrics.

MacTech

Another great web resource is MacTech.com. Run a search on **Applescript Essentials** to get access to a great deal of articles on AppleScript, including:

- Introduction to AppleScript Studio
- Using Repeat Loops in AppleScript
- Storing and Accessing Data with AppleScript
- Performing Basic Image Manipulation
- Introduction to Scripting Address Book
- User Interaction Basics
- An Introduction to Handlers
- Becoming More Efficient through Folder Watching
- An Introduction to Scripting Fetch

FIGURE B.4

ScriptBuilders.net

FIGURE B.5

The information page for Get_iTunes_Lyrics

Video tutorials on the Web

If you run a video search on Google for AppleScript tutorials, you'll end up here (see Figure B.6):

```
http://video.google.com/videosearch?client=safari&rls=en-us&q=applescript+tutorials&oe=UTF-8&um=1&ie=UTF-8&ei=IAJiSqvGOeG3tweOg9T0Dw&sa=X&oi=video_result_group&ct=title&resnum=10#
```

Let's BudURL that:

```
http://budurl.com/xyjy
```

Using Google to search for AppleScript tutorials is a pretty smart way of doing it — you'll always have access to the latest and greatest material from YouTube, Google Video, and other sites.

FIGURE B.6

Video tutorials on Google

Online Community

Here are some mailing lists, blogs, and forums devoted to AppleScript.

MacScripter

I've already mentioned it in the previous section, but MacScripter.com is pretty much the best online community out there for folks who are into AppleScript and Automator. It has hundreds of forums, thousands of topics, scripts, links, resources, and so on, as you can see from Figure B.7. If you're working on an AppleScript project, then you should go here.

FIGURE B.7

MacScripter AppleScript Forums

Don't forget about the AppleScript for Absolute Beginners forum (see Figure B.8) — it contains a lot of valuable information in tutorial format.

FIGURE B.8

The AppleScript for Absolute Beginners forum

Tools and Downloads

Here are a few useful downloads for AppleScript.

Doug's AppleScripts for iTunes

You can't really mention AppleScript tools and downloads without mentioning Doug's AppleScripts over at `www.dougscripts.com/`. Doug has over 450 (and counting!) free, user-contributed AppleScripts for iTunes that will "make maintaining your digital music collection easier and more fun!"

Figure B.9 shows the types of iTunes scripts that are available, by category.

FIGURE B.9

Doug's AppleScripts for iTunes

AppleScript downloads on CNET

If you go over to CNET, you'll find some interesting AppleScript downloads (see Figure B.10):

```
www.cnet.com/topic-software/applescript.html
```

Here's a BudURL:

```
http://budurl.com/u598
```

You'll find over 180 different AppleScripts that automate a variety of tasks. Feel free to download them and figure out how they work.

FIGURE B.10

The CNET AppleScript Downloads page

Training

Well, here we go again with the Automated Workflows guys! That's right, Ben Waldie's group does AppleScript training — both customized and basic introductory stuff (see Figure B.11). You can learn more at:

`www.automatedworkflows.com/training/training.html`

or

`http://budurl.com/b9he`

FIGURE B.11

AppleScript Training by Automated Workflows

Summary

In this appendix, you've learned about some tutorials, online communities, books, tools and downloads, and training. Don't forget to visit my Delicious bookmarks at `www.delicious.com/myerman/applescript`.

APPENDIX

C

AppleScript Reference

In this appendix, I'll try to provide as much reference material as I can on the different working aspects of AppleScript. Before going into too much detail here, please note that Apple has published a very detailed AppleScript Language Guide, available at:

```
http://developer.apple.com/documentation/
   applescript/Conceptual/AppleScriptLangGuide/
   introduction/ASLR_intro.html#//apple_ref/doc/uid/
   TP40000983-CH208-SW1
```

Here's the BudURL:

```
http://budurl.com/stpn
```

This guide (see Figure C.1) is constantly being updated by the good folks over at the Apple Developer Connection, and can answer a lot of your questions about the deeper, more esoteric aspects of the AppleScript language.

IN THIS APPENDIX

- **Class reference**
- **Commands reference**
- **Operators reference**
- **Control statements reference**
- **Handler reference**
- **AppleScript reserved keywords**
- **AppleScript error numbers**

FIGURE C.1

The AppleScript Language Guide

Class Reference

A class is simply a category of objects that share certain characteristics. Each object in a script is an instance of a specific class — it has the same properties, shares the same kinds of elements, and supports the same kinds of operations.

This section contains an alphabetical list of common objects that are true of every class.

Alias

This is a persistent reference to an existing file, folder, or volume. The target of an alias must already exist.

Properties

- **class:** The class identifier for the object. This is read-only and always returns a value of alias.
- **POSIX path:** The POSIX-style path to the object. This is read-only.

Coercions

Aliases can be coerced to a text object or single-item list.

Example

```
set alias_ to choose application as alias
- choose iTunes
--result might be "Macintosh HD:Applications:iTunes.app"
class of alias_ --alias
POSIX path of alias_ -- "/Applications/iTunes.app/"
```

Application

This is an application on a local or remotely available server (see Figure C.2).

FIGURE C.2

Calling an application

Properties

- **class:** The class identifier of an object. This is read only, and always returns a value of application.
- **frontmost:** A read-only Boolean that answers the question: is the application frontmost?
- **id:** The application's read-only bundle identifier for its four-character signature code.

- **name:** The application's read-only name.
- **running:** A read-only Boolean that answers the question: is the application running?
- **version:** A read-only string that contains the application's version number.

Coercions

Applications can be coerced to a single-item list.

Example

```
tell application "iTunes"
-- does NOT automatically launch the application!
    version -- I have version 8.2.1 installed
end tell
```

Boolean

The Boolean object evaluates to either true or false, both of which are constants. A Boolean expression contains one or more Boolean objects and evaluates to true or false.

Properties

- **class:** A read-only class identifier. It always returns a value of Boolean.

Operators

The following operators support Boolean objects:

- and
- or
- not
- &
- =
- ≠
- is equal to
- is not equal to
- equals

Coercions

Booleans can be coerced to a single-item list, a text object, or an integer.

Example

```
if "tom" = "thomas" then
    return true
else
    return false
end if
```

The above code returns false every time.

Class

The class object specifies the class of an object or value. You can use it to figure out how to identify an object (see Figure C.3).

FIGURE C.3

Using `class` to determine the class of an object

Properties

- **class:** A read-only identifier for the object. This always returns a value of class.

Operators

Operators that support class actions include:

- &
- =
- ≠
- as

Coercions

Classes can be coerced to a single-item list or text object.

Example

```
class of true -- Boolean!
100 as text -- converts integer into text string
```

Constant

A constant is an object with a predefined value. Unlike in other programming languages, in AppleScript, a constant is usually something you don't set in a script — it usually comes from something supported in AppleScript itself.

Properties

- **class:** A read-only identifier for the object. This always returns a value of constant.

Operators

Operators that support constant actions include:

- &
- =
- ≠
- as

Coercions

Constants can be coerced to a single-item list or a text object.

Example

```
considering white space
    "tom myer" = "tommyer"
end considering
class of white space -- constant
```

Date

A date is an object that contains the day of the week, the date (month/day, year), and the time (hours, minutes, seconds).

Properties

- **class:** A read-only identifier for the object. This always returns a value of date.
- **day:** A read/write integer property. This specifies the day of the month in a date object.
- **weekday:** A read-only constant, this can be Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday.
- **month:** A read/write constant, this can be January, February, March, April, May, June, July, August, September, October, November, or December.
- **year:** A read/write integer, this specifies the year in the date object (for example, 2009).
- **time:** A read/write integer, this specifies the number of seconds since midnight of a date object. For example, 1500 is equal to 12:25 AM (1500 / 60 = 25 minutes).
- **date string:** A read-only text object that specifies just the date portion of a date object (for example, "July 4, 2009").
- **time string:** A read-only text object that specifies just the time portion of a date object (for example, "3:45:33 PM").

Operators

Operators that support date actions include:

- &
- +
- -
- =
- ≠
- >
- ≥
- <
- ≤
- comes before
- comes after
- as

Coercions

Date objects can be coerced to a single-item list or a text object.

Examples

```
date "7/4/2009 12:01"
--result: date "Saturday, July 4, 2009 12:01:00 PM"
set date_ to the current date
--result: date "Sunday, July 19, 2009 1:06:59 PM"
day of date_
--result: 19
time string of date_
--result: "1:08:05 PM"
```

File

A file is a reference to a file, folder, or volume (see Figure C.4). This is the same as the alias object, except that it can refer to something that doesn't yet exist (such as when you use the Choose File Name dialog).

Coercions

Files can be coerced to a text object or single-item list.

Example

```
set file_ to open for access "Macintosh
   HD:Users:myerman:Desktop:test.txt"
```

FIGURE C.4

Opening a file on the desktop

Integer

An integer is any number (positive or negative) without a fractional part.

Properties

- **class:** A read-only identifier for the object. This always returns a value of integer.

Operators

Operators that support integer actions include:

- +
- -
- *
- /
- div
- mod
- ^
- =
- ≠
- >
- ≥
- <
- ≤

Coercions

Integers can be coerced to a text object, single-item list, or real number.

Examples

```
set result_ to 300 - 1
(1000 - 10) / 30 * (8 + 100)
```

List

A list is an ordered collection of values (see Figure C.5). Each value is known as an item, and an item can belong to any class.

Properties

- **class:** A read-only identifier for the object. This always returns a value of class.
- **length:** A read-only integer that specifies the number of items in a list.

- **rest:** A read-only list that contains all items in a list except for the first item.
- **reverse:** A read-only list that contains all items in a list, but in opposite order.

Coercions

Single-item lists can be coerced to just about any class supported by that item if it were not in a list. Multiple-item lists can be coerced to text objects if all objects in the list can be coerced as such.

Examples

```
{"a", 123345, "pony", "moonwalk"}
set list_ to {"a", 123345, "pony", "moonwalk"}
count of list_ -- result is 4
reverse of list -- result is {"moonwalk", "pony", 123345, "a"}
item 3 of list_ -- result is "pony"
```

FIGURE C.5

Reversing a list

Number

A number is an abstract class that represents an integer or a real number. You can't really have an object whose class is number — it'll always either be an integer or a real number.

Properties

- **class:** A read-only identifier for the object. This always returns a value of integer or real.

Operators

The operators supported are the same as for integers and real numbers.

Coercions

Coercing an object to number results in either an integer (if the result is an integer), or a real number (if it isn't an integer).

Examples

```
--all of these are valid integers or real numbers
--hence they are valid numbers as well
1
-5
1000000
10.238399101
4.0
0.1
```

POSIX file

This is a pseudo-class equivalent to a file class (see Figure C.6). You use it to convert HFS paths (which contain colons) into the more familiar POSIX paths (which contain slashes).

Properties

These are the same as for the file class.

Coercions

These are the same as for the file class.

Example

```
-- the following will open a dialog box in the Application s folder
set file_ to choose file name default location (POSIX file "/
   Applications")
```

FIGURE C.6

Using a POSIX file to choose a file

Real

A real number is a number that contains a fractional part, such as 1.1 or 1000000.28930303.

Properties

- **class:** A read-only identifier for the object. This always returns a value of real.

Operators

Operators that support real actions include:

- +
- -
- *
- /
- div
- mod
- ^
- =
- ≠
- >
- ≥
- <
- ≤

Coercions

Real numbers can be coerced into integers (thus losing their fractional parts) or as single-item lists or text objects.

Examples

```
10.255829019
1.0
0.1
```

Record

A record is an unordered collection of properties (see Figure C.7). This is very similar to hashes and associative arrays in other languages.

FIGURE C.7

Image properties are stored as a record

Properties

- **class:** A read-only identifier for the object. This always returns a value of record.
- **length:** A read-only integer that specifies the number of properties in a record.

Operators

Operators that support record actions include:

- &
- =
- ≠
- contains
- is contained by

Coercions

You can coerce records into lists — but you lose all labels in the process.

Example

```
set dog_ to {name:"marlowe", breed:"Yorkie", weight:"6 lbs"}
breed of dog_
--result: "Yorkie"
```

Script

A script is a collection of AppleScript declarations and statements.

Properties

- **class:** A read-only identifier for the object. This always returns a value of script.

Coercions

You can coerce a script into a single-item list.

Example

```
script myDog
    display dialog "My dog barks!"
end script
run myDog
```

Text

A text object is an ordered series of Unicode characters.

Properties

- **class:** A read-only identifier for the object. This always returns a value of text.
- **id:** A read-only integer that represents the Unicode code point(s) for the character(s) in the text object.
- **length:** A read-only integer that specifies how many characters are in the object.
- **quoted form:** A read-only version of the text string that is safe to pass to the shell.

Elements

A text object can contain characters, words, paragraphs, or text (that is, everything in the object).

Special characters include tab and return.

Coercions

Text objects can be coerced to single-item lists. If it represents a number, a text object can be coerced to an integer or real number.

Examples

```
set text_ to "hello there"
character 3 of text_ -- "I"
text_ as list -- {"hello there"}
get word 1 of text -- "hello"
```

Commands Reference

Tables C.1 through C.10 summarize the most common AppleScript commands.

TABLE C.1

AppleScript Suite

Command	Description
activate	Brings an application to the front and opens it if it is on the local computer and not already running.
log	In AppleScript Editor, this displays a value in the Event Log History window or in the Event Log pane of a script window.

TABLE C.2

Clipboard Commands

Commands	Description
clipboard info	Returns information about the clipboard.
set the clipboard to	Places data on the clipboard.
the clipboard	Returns the contents of the clipboard.

TABLE C.3

File Commands

Command	Description
info for	Returns information for a file or folder
list disks	Returns a list of the currently mounted volumes. This command is deprecated.
list folder	Returns the contents of a specified folder. This command is deprecated.
mount volume	Mounts the specified AppleShare volume.
path to (application)	Returns the full path to the specified application.
path to (folder)	Returns the full path to the specified folder.
path to (resource)	Returns the full path to the specified resource.

TABLE C.4

File Read/Write Operations

Command	Description
close access	Closes a file that was opened for access.
get eof	Returns the length, in bytes, of a file.
open for access	Opens a disk file for the `read` and `write` commands.
read	Reads data from a file that has been opened for access.
set eof	Sets the length, in bytes, of a file.
write	Writes data to a file that was opened for access with write permission.

TABLE C.5

Internet Commands

Command	Description
open location	Opens a URL with the appropriate program.

TABLE C.6

Miscellaneous Commands

Command	Description
current date	Returns the current date and time.
do shell script	Executes a shell script using the `sh` shell.
get volume settings	Returns the sound output and input volume settings.
random number	Generates a random number.
round	Rounds a number to an integer.
set volume	Sets the sound output and/or input volume.
system attribute	Gets environment variables or attributes of this computer.
system info	Returns information about the system.
time to GMT	Returns the difference between local time and GMT (Universal Time).

TABLE C.7

Scripting Commands

Command	Description
load script	Returns a `script` object loaded from a file.
run script	Runs a script or script file.
scripting components	Returns a list of all scripting components.
store script	Stores a `script` object into a file.

TABLE C.8

Standard Suite

Comm.and	Description
copy	Copies one more values into variables.
count	Counts the number of elements in an object.
get	Returns the value of a script expression or an application object.
launch	Launches the specified application without sending it a `run` command.
run	Launches an application, script, or script object.
set	Assigns one more values to one or more script variables or application objects.

TABLE C.9

String Commands

Command	Description
ASCII character	Converts a number to a character. This command is deprecated starting in AppleScript 2.0. Use the `id` property of the `text` class instead.
ASCII number	Converts a character to its numeric value. This command is deprecated starting in AppleScript 2.0. Use the `id` property of the `text` class instead.
localized string	Returns the localized string for the specified key.
offset	Finds one piece of text inside another.
summarize	Summarizes the specified text or text file.

TABLE C.10

User Interaction Commands

Command	Description
beep	Beeps one or more times.
choose application	Allows the user to choose an application.
choose color	Allows the user to choose a color.
choose file	Allows the user to choose a file.
choose file name	Allows the user to specify a new file reference.
choose folder	Allows the user to choose a folder.
choose from list	Allows the user to choose one or more items from a list.
choose remote application	Allows the user to choose a running application on a remote machine.
choose URL	Allows the user to specify a URL.
delay	Pauses for a fixed amount of time.
display alert	Displays an alert.
display dialog	Display a dialog, optionally requesting user input.
say	Speaks the specified text.

Operators Reference

The following sections summarize the different kinds of operators.

Logical operators

Logical conjunction (and)

You can use the `and` operator to combine two Boolean values. The result is true only if both operands evaluate to true. AppleScript checks the left-hand operand first, and if it is false, it ignores the right-hand operand. The result is always a Boolean.

For example:

```
true and true -- evaluates to true
true and false -- evaluates to false
false and false -- evaluates to false
false and true -- evaluates to false
```

Logical disjunction (or)

You can also use the `or` operator to combine two Boolean values. The result is true if either operand evaluates to true. AppleScript checks the left-hand operand first, and if it is true, it ignores the right-hand operand. The result is always a Boolean.

```
true or false -- evaluates to true
false or true -- evaluates to true
true or true  -- evaluates to true
false or false -- evaluates to false
```

Negation (not)

Negation is a unary logical operator, and as such, it only requires one operand. If the operand to its right is false, it returns true. If the operand to its right is true, it returns false. For example:

```
not true -- returns false
set myvalue to true -- returns true
not myvalue -- returns false
```

Equality

The most common way to compare equality is with the = operator, but `is equal`, `equals`, and `is equal to` also work the same. You'll use these operators to determine if two values equal each other. For example:

```
"beaumont, tx" is equal to "beaumont, tx" -- true
"beaumont, tx" = "beaumont, tx" - true
9 equals 8 -- false
```

Inequality

As with equality, you have a lot of options for testing inequality. Basically, if your two operands are different, the inequality result is true (which is what you'd expect). You can use the ≠ operator

(press Option+= on the keyboard), is not, isn't, isn't equal to, is not equal, doesn't equal, or does not equal. For example:

```
"beaumont, ca" is not "beaumont, tx" -- true
"beaumont, ca" does not equal "beaumont, tx" -- true
"beaumont, ca" doesn't equal "beaumont, tx" -- true
9 isn't 4 -- true
```

Greater than

The greater than operator results in a true return value if the left-hand operand is greater than the right-hand operand. You can use the > symbol, as well as is greater than, comes after, is not less than or equal to, or isn't less than or equal to, depending on your needs. For example:

```
9 > 4 -- true
11 > 34 -- false
12/1/2009 > 11/30/2009 -- true
"Jethro Gibbs" > "Zeva David" -- false, but only in AppleScript!
```

Less than

The less than operator results in a true return value only if the left-hand operand is smaller than the right-hand operand. You can use the < symbol, as well as is less than, comes before, is not greater than or equal to, or isn't greater than or equal to, depending on your needs. For example:

```
3 < 4 -- true
110 < 54 -- false
12/1/2007 > 11/30/2009 -- true
"Kafka" < "Marlowe" -- true
```

Greater than or equal to

The greater than or equal to operator results in a true return value only if the left-hand operand is greater than or equal to the right-hand operand. You can use the ≥ symbol (Option+. on the keyboard), as well as is greater than or equal to, is not less than, isn't less than, or does not come before, depending on your needs. For example:

```
5 isn't less than 4 -- true
333 ≥ 321 -- true
```

Less than or equal to

The less than or equal to operator results in a true return value only if the left-hand operand is less than or equal to the right-hand operand. You can use the ≤ symbol (Option+, on the keyboard), as well as is less than or equal to, is not more than, isn't more than, or does not come after, depending on your needs. For example:

```
3 isn't more than 4 -- true
11 ≤ 500 -- true
```

Mathematical operators

Multiplication (*)

The multiplication operator multiplies the number to its left and the number to its right. For example:

```
9 * 8 -- returns 72
```

Addition (+)

The addition operator adds the number or date to its left and the number or date to its right. Only integers can be added to dates — AppleScript considers that integer as a number of seconds. For example:

```
3 + 4 -- returns 7
3 + -4 -- returns -1
```

Subtraction (-)

The subtraction operator subtracts the number or date to its right from the number or date to its left. As a unary operator, it makes the number to its right negative (as shown in the example). As with addition, only integers can be subtracted from a date, and AppleScript interprets it as a number of seconds. For example:

```
100 - 1 -- results in 99
1 - 100 -- results in -99
```

Division (/)

The division operator divides the number to its left by the number to its right. You can also use the ÷ symbol (Option+/ on the keyboard). For example:

```
88 / 3 -- results in 29.333333333333
3 / 8 -- results in 0.034090909091
122 / 0 -- error! can't divide by zero
```

Integral division (div)

The integral division operator is exactly like the division operator, except that it only returns the integral part of the result — so you only see an integer.

```
88 div 3 -- results in 29
3 div 8 -- results in 0
```

Remainder (mod)

The remainder operator divides the number to its left by the number to its right and returns the remainder. For example:

```
10 mod 3 -- results in 1
```

Exponentiation (^)

The exponentiation operator raises the number to its left to the power of the number to its right. For example:

```
10 ^ 3 -- results in 1000.0 (10*10*10)
6 ^ 67 – results in 1.3681501912022E+52
```

Other operators

Concatenation

You've already seen the concatenation operator in action. Using the & symbol, you can concatenate strings, lists, and records. For example:

```
"6" & "6" -- results in "66"
6 & 6 -- results in {"6", "6"}
{yorkie:"Marlowe"} & {mutt:"Kafka"} – results in {yorkie:"Marlowe",
   mutt:"Kafka"}
```

Notice how there are different results for the different data types? This has something to do with coercion. For now, be aware that AppleScript takes whatever is on the left-hand side of an operator and tries to coerce the right-hand operand into the same data type if it can.

Containment

There are a whole bunch of operators in AppleScript that can help you determine if what you're looking for `starts with` a value, `ends with` a value, `contains` a value, `does not contain` a value, `is in` a container, or `is not in` a container. For example:

```
set mylist to {4, 5, 6}
mylist starts with 4 -- true
mylist contains 3 -- false
6 is in mylist -- true
mylist ends with 8 -- false
23 isn't in mylist -- true
```

By the way, containment operators are great ways to figure out if text strings contain substrings. For example:

```
"Kafka" contains "fk" -- true
"arlo" is contained by "Marlowe" -- true
```

Control Statements Reference

Considering and ignoring

You can force AppleScript to consider or ignore certain characteristics, such as white space or case. You can also use these clauses to consider or ignore responses from applications. When dealing with text, you can consider or ignore:

- case
- diacriticals
- hyphens
- numeric strings
- punctuation
- white space

For example:

```
"hello tom" = "hellotom" -- false
ignoring white space
    "hello tom" = "hellotom" -- true
end ignoring
```

You can also ignore or consider application responses. For example, you may want to empty the trash but not wait around for a response:

```
tell application "Finder"
    ignoring application responses
        empty the trash
    end ignoring
end tell
```

If

You usually have two different types of `if` statements: a simple one and a complex one. The simple `if` statement usually only has one Boolean expression:

```
if 1 > 0 then
    return true
end if
```

The compound `if` statement can be one that contains just an `else` branch or multiple `else if` branches on top of the main branch. For example:

```
set today_ to current date
if weekday of today_ = Sunday then
    display dialog "sleep in!"
else if weekday of today_ = Saturday then
    display dialog "work in the yard!"
else
    display dialog "go to work!"
end if
```

Repeat

The basic `repeat` loop that goes on forever (or until something interrupts the loop) looks like this:

```
repeat
    display dialog ¬
      "Please enter your first name:" default answer ""
    set firstName to text returned of result
    if firstName is not equal to "" then exit repeat
end repeat
```

The following code repeats a loop a set number of times:

```
repeat 3 times
    display dialog ¬
      "Please enter your first name:" default answer ""
    set firstName to text returned of result
    if firstName is not equal to "" then exit repeat
end repeat
if firstName = "" then display dialog ¬
    "You never gave us your first name!"
```

The following code repeats with an incremental value:

```
tell application "Finder"
    repeat with incrementValue from 1 to 5
      make new folder at desktop with properties ¬
      {name:incrementValue as string}
end repeat
end tell
```

In this case, each time through the loop, the value of `incrementValue` changes. The first time through the loop, its value is 1, hence the folder that is created on the desktop is named 1. The next time through the loop, the variable's set to 2, and so on. The result is five folders on the desktop.

There's no need to start at 1, of course. You can write scripts that use other start values:

```
tell application "Finder"
    repeat with incrementValue from 10 to 50
        make new folder at desktop with properties ¬
          {name:incrementValue as string}
    end repeat
end tell
```

This script would create 40 folders on the desktop instead of just 5, with the first folder named 10 and the last one named 50.

To increase the increment value, simply add a `by` clause:

```
tell application "Finder"
    repeat with incrementValue from 1 to 50 by 5
        make new folder at desktop with properties ¬
          {name:incrementValue as string}
    end repeat
end tell
```

This creates folders named 6, 11, 16, 21, 26, 31, 36, 41, and 46 on the desktop. Each time through the `repeat` loop, the value of `incrementValue` is incremented by 5, not 1.

To decrement a value, simply add a `by` clause with a negative number, but make sure that your starting number is higher than your ending number:

```
tell application "Finder"
    repeat with incrementValue from 10 to 1 by -1
        make new folder at desktop with properties ¬
          {name:incrementValue as string}
    end repeat
end tell
```

What you end up with when you run this script are ten folders, with the 10 folder created first, all the way down to the 1 folder.

The `repeat until` statement is used to repeat a process or action until something is true. For example, you might set a variable to a certain value, and then repeat a loop until that value has reached a certain value, incrementing each time through the loop:

```
set loopCounter to 1
repeat until loopCounter = 30
    display dialog loopCounter
    set loopCounter to loopCounter + 1
end repeat
```

In this code, I set a variable named `loopCounter` to a value of 1. Then I use a `repeat until` loop to keep working until `loopCounter` is set to 30. Then, each time through the loop, I display the value of the variable in a dialog and then increment the `loopCounter` variable by one.

By itself, this code isn't very useful at all, but you can imagine how useful it could be. For example, you could rewrite the infinite `repeat` loop at the beginning of this section to something like this:

```
set firstName to ""
repeat until firstName does not equal ""
    display dialog ¬
      "Please enter your first name:" default answer ""
    set firstName to text returned of result
end repeat
```

This does the exact same thing as the infinite `repeat` loop, but it's a bit more elegant.

You can also set a `repeat` loop to operate while something is true. This is known as the `repeat while` loop and is used to test for a particular scenario to appear.

You would use the following code to wait until a file called test.log is created on the desktop before displaying a dialog:

```
set myLogFile to (path to desktop as string) & "test.log"
tell application "Finder"
    repeat while (file myLogFile exists) = false
    end repeat
end tell
display dialog "The log file is on the desktop!"
```

You can use `repeat with` to iterate over a list, like this:

```
set myList to {"dogs", "cats", "cars", "planes", "food"}
tell application "Finder"
    repeat with noun in myList
        make new folder at desktop with properties ¬
        {name:noun as string}
    end repeat
end tell
```

Tell

A `tell` statement specifies a target for a code block. For example, you might need iTunes to do something specific (such as play a random track). You can't ask just any application to play that track — you have to tell iTunes to do it:

```
tell application "iTunes"
    play some track
end tell
```

Telling another application (such as the Finder or Mail) to play a random track would result in an error.

You can also shorten the traditional `block` statement into a single line, like this:

```
tell application "iTunes" to play some track
```

Try

A `try` statement gives your scripts a chance to catch errors and recover from them. For example, let's say you're trying to divide by zero:

```
try
    100 / 0
on error
    display dialog "Can't divide by zero!"
end try
```

Sure enough — you'll see the dialog.

Handler Reference

The most important thing to remember about handlers is this: if you use a `return` statement, that statement exits the handler and may or may not return a specified value. For example, here's a handler that returns nothing:

```
on countItems()
    2 + 2
    return
end countItems
```

Just because you've got some math going on in a handler doesn't mean anything — if you return nothing, then you return, well, nothing. Here's how to return the value:

```
on countItems()
    sum = 2 + 2
    return sum
end countItems
```

Or more simply:

```
on countItems()
    return 2 + 2
end countItems
```

If you put anything after the `return` statement, then you don't get it back — once you return, you return:

```
on countItems()
    return 2 + 2
    sum = 5 + 2
    return sum
end countItems
```

The code above returns 4 and ignores the sum variable set to 7.

Of course, if you set up a conditional, then you can set up different return values:

```
on isSunday()
    set weekday_ to weekday of (current date)
    if weekday_ = Sunday then
        return true
    else
        return false
    end if
end isSunday
```

Running this code on Sunday returns true; otherwise, it returns false (see Figure C.8).

FIGURE C.8

Running the `isSunday()` handler

AppleScript Reserved Keywords

The following are keywords that are reserved by AppleScript. Don't try to use them to name variables or for any other purpose, as you'll get an error.

- about
- above
- after
- against
- and
- apart from
- around
- as
- aside from
- at
- back
- before
- beginning
- behind
- below
- beneath
- beside
- between
- but
- by
- considering
- contain, contains
- continue
- copy
- div
- does
- eighth
- else
- end

- equal, equals
- error
- every
- exit
- false
- fifth
- first
- for
- fourth
- from
- front
- get
- given
- global
- if
- ignoring
- in
- instead of
- into
- is
- it
- its
- last
- local
- me
- middle
- mod
- my
- ninth
- not
- of
- on
- onto
- or

- out of
- over
- prop, property
- put
- ref, reference
- repeat
- return
- returning
- script
- second
- set
- seventh
- since
- sixth
- some
- tell
- tenth
- that
- the
- then
- third
- through, thru
- timeout
- times
- to
- transaction
- true
- try
- until
- where
- while
- whose
- with
- without

AppleScript Error Numbers

Table C.11 summarizes some of the error numbers you might see when working with your scripts.

TABLE C.11

Potential Error Numbers

Error number	Error message
-2700	Unknown error.
-2701	Can't divide <number> by zero.
-2702	The result of a numeric operation was too large.
-2703	<reference> can't be launched because it is not an application.
-2704	<reference> isn't scriptable.
-2705	The application has a corrupt dictionary.
-2706	Stack overflow.
-2707	Internal table overflow.
-2708	Attempt to create a value larger than the allowable size.
-2709	Can't get the event dictionary.
-2720	Can't both consider and ignore <attribute>.
-2721	Can't perform operation on text longer than 32KB.
-2729	Message size too large for the 7.0 Finder.
-2740	A <language element> can't go after this <language element>.
-2741	Expected <language element> but found <language element>.
-2750	The <name> parameter is specified more than once.
-2751	The <name> property is specified more than once.
-2752	The <name> handler is specified more than once.
-2753	The variable <name> is not defined.
-2754	Can't declare <name> as both a local and global variable.
-2755	`Exit` statement was not in a `repeat` loop.
-2760	`Tell` statements are nested too deeply.
-2761	<name> is illegal as a formal parameter.
-2762	<name> is not a parameter name for the event <event>.
-2763	No result was returned for some argument of this expression.

Summary

In this appendix, you got more in-depth exposure to classes, commands, handlers, and other AppleScript details.

APPENDIX

D

Automator Actions and Variables

In this appendix, I'll provide a summary of the basic actions and variables available in the Snow Leopard version of Automator. This is not meant to be a comprehensive reference featuring everything you need to know, just simply a summary.

IN THIS APPENDIX

Actions

Variables

Actions

The standard Automator actions include these different categories or groupings:

- Calendar
- Contacts
- Files & Folders
- Fonts
- Internet
- Mail
- Movies
- Music
- PDFs
- Photos
- Text
- Utilities

In addition, Automator provides two smart groups: Most Used and Recently Added. You can create your own smart groups, of course, to help you to stay organized.

One final note before you venture any further. Each action will normally have some kind of input and some kind of result. For example, if you're filtering calendar events, you start by giving the action a set of iCal events as input. Once the action processes those events, you end up with some kind of result — in this case, a set of filtered iCal events.

Calendar

Calendar actions are designed to help you with workflows that deal with events, calendars, and other calendar-like activities. Figure D.1 provides a visual summary of the standard actions in this group.

FIGURE D.1

The Calendar actions

The following is a list of standard Calendar actions:

- **Delete iCal Events:** This action permanently deletes iCal events, without any chance for undoing the delete. Input is one or more iCal events. Results in deleted events.
- **Event Summary:** Creates a summary of any iCal calendars and events passed into the action. Input is events and calendars. Results in text.
- **Filter iCal Items:** Use this action to set different criteria (for example, matching on calendar name, event start date, and so on) for filtering items. Input is always one or more iCal items. Results in a list of iCal items that match the filter criteria.
- **Find iCal Items:** Use this action to search for calendar items that meet a specified criteria (for example, calendar name, event start date, and so on). Input is always one or more iCal items. Results in iCal events that match the search criteria.

- **Get Specified iCal Items:** Use this action to pass in a specific set of iCal events. Input is always one or more iCal items. Results in iCal items.
- **New Calendar:** Use this action to create a new calendar in iCal. If the specified calendar already exists, then your new calendar is not created. Input is any name the user enters. Results in a new iCal calendar.
- **New iCal Events:** Use this action to create a new event in iCal. Input can include Address Book groups and people that can be included as attendees. Results in a new iCal event.
- **New To Do Item:** Use this action to create a new to do item. Input is usually text. Results in a new to do created in iCal.

Contacts

The Contacts actions allow you to work with your contacts and groups in Address Book. Figure D.2 provides a visual summary of the standard actions in this group.

FIGURE D.2

The Contact actions

The following is a list of standard Contacts actions:

- **Export vCards**. Use this action to export contacts from your Address Book in vCard format. Input is contacts from your Address Book. Results in a vCard file.
- **Filter Address Book Items.** Use this action to filter contacts by a certain criteria. Input is a group of contacts. Results in a set of contacts that match the filter criteria.
- **Find Address Book Items.** Use this action to find specific contacts in Address Book. Input is a group of contacts. Results in a set of contacts that match the search criteria.

- **Find People with Birthdays.** Use this action to find contacts in Address Book who have an upcoming birthday. Input is a set of contacts. Results in a list of contacts with upcoming birthdays. Can be set to different tolerances — birthdays that happen today, this week, this month, and so on.
- **Get Contact Information.** Use this action to extract specific information about contacts. Input is a set of contacts from Address Book. Results in text that matches extraction criteria. Different formats supported include text, spreadsheet format, and AppleScript lists.
- **Get Selected Address Book Items.** Use this action to pass selected items in Address Book to another action in the workflow. Input is a set of contacts. Results in those contacts being passed to the next action.
- **Get Specified Address Book Items.** Use this action to pass specific actions (as opposed to selected ones) to the next action in the workflow. Input is contacts from Address Book. Results in those contacts being passed on to the next action.
- **Group Mailer.** Use this action to send a prepared Mail message to a group of people. This action requires an open outgoing message in Mail prepared with subject, content, and attachments. Input is a set of Address Book contacts. Results in message sent to each contact.

Files & Folders

The Files & Folders actions allow you to work directly with files and folders on your Mac's file system. Figure D.3 provides a visual summary of the actions in this group.

FIGURE D.3

The Files & Folders actions

The following is a list of standard Files & Folders actions:

- **Ask for Finder Items.** Use this action to let users choose Finder items in a dialog. Input is files/folders. Results in selected items passed to the next action.
- **Ask for Servers.** Use this action to choose a server. Input is a list of URLs. Results in those URLs being passed to the next action.
- **Connect to Servers.** Use this action to connect your computer to a server on the network. Input is a list of URLs. Results in files/folders from that server being passed to the next action.
- **Copy Finder Items.** Use this action to copy specified Finder items to a location. Input is files/folders. Results in those files/folders being passed to the next action.
- **Create Archive.** Use this action to create a ZIP archive of specified files. Input is any files/folders you want archived. Results in an archived ZIP file.
- **Duplicate Finder Items.** Use this action to duplicate files or folders. Input is any file or folder you want duplicated. Results in those files and folders duplicated in the same location.
- **Eject Disk.** Use this action to eject a mounted disk or volume. Input is a mounted disk or volume. Results in that disk or volume being ejected.
- **Filter Finder Items.** Use this action to filter files/folders by criteria. Input is files/folders. Results in a set of files/folders that meet the filter criteria.
- **Find Finder Items.** Use this action to find files/folders by using different criteria. Input is files/folders. Results in a set of files/folders that match the search criteria.
- **Get Folder Contents.** Use this action to get all items inside a specified folder. Input is a set of folders. Results in all files/folders inside specified targets.
- **Get Selected Finder Items.** Use this action to get a selected Finder item. Input is any Finder item that is selected prior to the workflow running. Results in Finder item being passed to the next action.
- **Get Specified Finder Items.** Use this action to get a specified Finder item. Input is any specific Finder item. Results in the Finder item being passed to the next action.
- **Get Specified Servers.** Use this action to pass specified servers to the next action. Input is a list of URLs. Results in a list of URLs being passed to the next action.
- **Label Finder Items.** Use this action to apply a label to specified files. Input is any set of Finder items. Results in those Finder items being labeled.
- **Mount Disk Image.** Use this action to mount a disk image. Input is a set of files or folders. Results in a mounted disk image.
- **Move Finder Items.** Use this action to move Finder items from one location to another. Input is a set of files or folders. Results in those items moved to specified location.

- **Move Finder Items to Trash.** Use this action to move files and folders to the Trash. Input is a set of files or folders. Results in those items being moved to the Trash.
- **New Aliases.** Use this action to create aliases of files and folders. Input is a set of files and folders. Results in aliases created for files and folders.
- **New Disk Image.** Use this action to create a disk image (.dmg) file. Input is any set of files or folders. Results in a new disk image with a volume name of *untitled*.
- **New Folder.** Use this action to create a new folder with a specified name. You can provide any list of files or folders as an input — they will be copied to the new folder. Results in a new folder.
- **Open Finder Items.** Use this action to open Finder items with a specified application. Input is any files or folders. Results in those items opened with an application.
- **Rename Finder Items.** Use this action to rename files and folders. Input is any set of files or folders. Results in those items being renamed. You have different options for renaming, like adding various prefixes or suffixes (for example, a date or time, a string, a number, and so on).
- **Reveal Finder Items.** Use this action to open a Finder window with the specified items revealed. Input is any set of files or folders. Results in open Finder windows.
- **Set Application for Files.** Use this action to assign an application to open specified files. Input is any set of files or folders. Results in those files and folders assigned a specific application.
- **Set Folder Views.** Use this action to set a folder view (icon, list, column, cover flow). Input is any set of files or folders. Results in Finder windows opening and setting to the specified folder view. If you have a lot of files or folders in different windows, this action might take several minutes to run.
- **Set Spotlight Comments for Finder Items.** Use this action to add specified comments to files and folders. Input is any set of files or folders. Results in comments added to those files and folders.
- **Set the Desktop Picture.** Use this action to replace the current desktop picture with a specified image. Input is any image file. Results in desktop image being changed.
- **Sort Finder Items.** Use this action to sort a list of files or folders. Input is any set of files or folders. Results in a list of files or folders that sorted to your criteria.

Fonts

The Fonts actions allow you to work directly with fonts. Figure D.4 provides a visual summary of the actions in this group.

FIGURE D.4

The Fonts actions

The following is a list of Fonts actions:

- **Activate Fonts.** Use this action to activate fonts passed from the previous action. Input is any Font Book typeface. Results in a Font Book typeface.
- **Add to Font Library.** Use this action to add objects passed from a previous action to the Font Book library. Input can be anything. Results in a font added to the library.
- **Deactivate Fonts.** Use this action to deactivate the fonts passed from the previous action. Input is any Font Book typeface. Results in a Font Book typeface.
- **Export Font Files.** Use this action to collect the files associated with fonts passed from a previous action and copy them to a folder. Input is any set of Font book typefaces. Results in files copied to a folder.
- **Filter Font Book Items.** Use this action to determine if fonts passed in from a previous action meet the specified criteria. Input is any set of fonts. Results in a set of Font Book typefaces.
- **Filter Fonts by Font Type.** Use this action to determine if fonts are of a selected type. Input is any Font Book typeface. Results in a set of Font Book typefaces that match the filter criteria.
- **Find Font Book Items.** Use this action to search for typefaces with certain criteria. Input is any Font Book typeface. Results in a set of Font Book typefaces that match search criteria.
- **Get Files for Fonts.** Use this action to return the files associated with the fonts passed in from the previous action. Input is any Font Book typeface. Results in a set of files or folders.
- **Get Font Info.** Use this action to extract and format the given font information. Input is any Font Book typeface. Results in text.

- **Get Fonts from Font Files.** Use this action to return the fonts associated with files passed from the previous action. Input is any set of files or folders. Results in a set of Font Book typefaces.
- **Get Fonts from TextEdit Document.** Use this action to return the fonts used in the frontmost TextEdit document. Requires an open TextEdit document. Results in a set of Font Book typefaces.
- **Get Postscript name of Font.** Use this action to return the PostScript name of the fonts passed from the previous action. Input is any Font Book typeface. Results in text.
- **Get Selected Font Book Items.** Use this action to get the selected Font Book typefaces and passes them to the next action. Input is any set of selected items. Results in a set of Font Book typefaces.
- **Remove Font Files.** Use this action to remove the fonts associated with files passed from the previous action. Input is any set of files or folders. Results in files with stripped fonts.
- **Select Fonts in Font Book.** Use this action to pass fonts that are selected to the next action. Input is any set of Font Book typefaces. Results in Font Book typefaces.
- **Validate Font Files.** Use this action to validate the fonts associated with the files passed from the previous action. Input is any set of files or folders. Results in validated files.

Internet

The Internet actions allow you to work directly with RSS feeds, Web pages, and other Internet-related tasks and tools. Figure D.5 provides a visual summary of the actions in this group.

FIGURE D.5

The Internet actions

The following is a list of Internet actions:

- **Display Web Pages.** Use this action to open Web pages in your default browser, using a separate tab or window for each URL submitted. Input is a list of URLs. Results in open browser windows or tabs.
- **Download URLs.** Use this action to download specific files from the Internet. Input is a list of URLs. Results in files or folders created on your Mac.
- **Filter Articles.** Use this action to filter articles in an RSS feed based on a set of criteria. Input is any set of RSS articles. Results in a set of RSS articles that match the filter criteria.
- **Filter URLs.** Use this action to filter a list of URLs by certain criteria, like matching a name, host, or query. Input is any list of URLs. Results in a set of URLs that match the criteria.
- **Get Current Webpage from Safari.** Use this action to get the URL of the displayed Web page in the front Safari window. Requires that a Web page be open in Safari. Results in the URL.
- **Get Enclosure URLs from Articles.** Use this action to locate enclosures in published RSS feed articles. Input is any set of RSS articles. Results in a list of URLs.
- **Get Feeds from Mail.** Use this action to retrieve RSS feeds currently subscribed to in Mail. No direct input needed, you can optionally add other RSS feeds. Results in a set of RSS feeds.
- **Get Feeds from Safari.** Use this action to retrieve RSS feeds currently subscribed to in Safari. No direct input needed, you can optionally add other RSS feeds. Results in a set of RSS feeds.
- **Get Feeds from URLs.** Use this action to retrieve RSS feeds associated with URLs. Input is a list of URLs. Results in a set of RSS feeds from those URLs.
- **Get Image URLs from Articles.** Use this action to extract image URLs from RSS feed articles. Input is any set of RSS articles. Results in a list of URLs.
- **Get Image URLs from Webpage.** Use this action to extract image URLs from a Web page. Input is any set of URLs. Results in a list of URLs.
- **Get Link URLs from Articles.** Use this action to identify links in specified RSS articles. Input is any set of RSS articles. Results in a list of URLs.
- **Get Link URLs from Webpages.** Use this action to retrieve URLs from links in a Web page. Input is any set of URLs. Results in a list of URLs.
- **Get Permalinks of Articles.** Use this action to retrieve the source URL for each RSS article. Input is any set of RSS articles. Results in a list of URLs.
- **Get Specified URLs.** Use this action to pass specified URLs to the next action. Input is any list of URLs. Results in those URLs being passed to the next action.
- **Get Text from Articles.** Use this action to extract text from RSS feed articles. Input is any set of RSS articles. Results in text from those articles.

- **Get Text from Webpage.** Use this action to extract text from a Web page. Input is any set of URLs. Results in text from that Web page.
- **Mark Articles.** Use this action to mark RSS feed articles as unread, read, unflagged, or flagged. Input is any set of RSS articles. Results in a set of marked RSS articles.
- **New Safari Documents.** Use this action to open new tabs or windows in Safari. Input is any list of URL addresses passed from a previous action. Results in new Safari tabs or windows.
- **Run Web Service.** Use this action to run a simple interface tool to run a specific Web service with given parameters. Input can be anything. Results in output of Web service returned as a list.

Mail

The Mail actions allow you to work directly with emails, attachments, and other Mail-related tasks. Figure D.6 provides a visual summary of the actions in this group.

The following is a list of Mail actions:

- **Add Attachments to Front Message.** Use this action to attach files to a message. You must have Mail running and an outgoing message ready. Input is any set of files to attach to the message. Results in messages with attachments.

FIGURE D.6

The Mail actions

- **Combine Mail Messages.** Use this action to combine the text from selected messages. Input is any set of Mail messages. Results in text of combined messages.
- **Display Mail Messages.** Use this action to focus the Mail viewer window to the messages passed to the action. Input can be messages or mailboxes. Results in Mail messages.
- **Filter Mail Items.** Use this action to filter Mail items by certain criteria. Input is any set of messages, mailboxes, or accounts. Results in a set of items filtered by the criteria.
- **Find Mail Items.** Use this action to search for items with specified criteria. Input is any set of messages, mailboxes, or accounts. Results in a set of items that meet the search criteria.
- **Get Attachments from Mail Messages.** Use this action to extract files attached to Mail messages. Input is any set of Mail messages. Results in files passed to the next action.
- **Get New Mail.** Use this action to check for new messages on Mail. Can be used to check specific accounts or all accounts.
- **Get Selected Mail Items.** Use this action to get selected items and pass them to the next action. Input can be any messages, mailboxes, or accounts. Results in passing those items to the next action.
- **Get Specified Mail Items.** Use this action to pass specified Mail items to the next action. Input can be any Mail item. Results in Mail items passed to the next action.
- **Group Mailer.** Use this action to send a prepared Mail message to a group of people. This action requires an open outgoing message in Mail prepared with subject, content, and attachments. Input is a set of Address Book contacts. Results in message sent to each contact.
- **New Mail Message.** Use this action to create a new Mail message. Input can be text or files passed into the action. If files are passed in, they are attached. If text is passed in, it becomes the body of the message. Results in Mail messages.
- **New To Do Item.** Use this action to create a new to do item. Input is usually text. Results in a new to do created in iCal.
- **Send Birthday Greetings.** Use this action to send an email with a Birthday greeting. Input is any set of Address Book contacts or groups. Results in Mail messages.
- **Send Outgoing Messages.** Use this action to send outgoing Mail messages. Input is any set of Mail messages. Results in those messages being sent.

Movies

The Movies actions allow you to work directly with multimedia movie files. Figure D.7 provides a visual summary of the actions in this group.

FIGURE D.7

The Movies actions

The following is a list of Movies actions:

- **Add Color Profile.** Use this action to add a color profile to a QuickTime movie. Input is any set of QuickTime files. Results in a set of QuickTime files.
- **Ask for Movies.** Use this action to let the user choose movies from a dialog. Input is any set of files or folders. Results in a set of files or folders.
- **Browse Movies.** Use this action to display the given movie files in Safari. Movie files are passed in from a previous action. Results in movie files opening in Safari.
- **Convert Quartz QuickTime Movies.** Use this action to convert Quartz Composition files to QuickTime movies. Input is any set of files or folders. Results in Movie files.
- **Enable or Disable Tracks.** Use this action to enable or disable QuickTime movie tracks. Input is any set of files or folders. Results in a set of files or folders.
- **Export Movies.** Use this action to export movie files viewable in different formats (iPhone, iPod, or AppleTV). Input is any set of files or folders. Results in formatted movie files.
- **Get Specified Movies.** Use this action to pass specified movies to the next action. Input is any set of movie files. Results in a set of movie files.
- **Hint Movies.** Use this action to add hints to the given QuickTime movie files. Input is any set of files or folders. Results in a set of files or folders.

- **New Audio Capture.** Use this action to open a new QuickTime audio recording window. Input can be anything. Results in anything.
- **New Screen Capture.** Use this action to open a new QuickTime screen recording window. Input can be anything. Results in anything.
- **New Video Capture.** Use this action to open a new QuickTime video recording window. Input can be anything. Results in anything.
- **Pause Capture.** Use this action to pause any current QuickTime capture. Input can be anything. Results in anything.
- **Pause DVD Playback.** Use this action to pause a DVD playing in the DVD player.
- **Play DVD.** Use this action to play a DVD in the DVD player.
- **Play Movies.** Use this action to play the specified movies, in a loop. Input can be any set of movie files. Results in a set of movie files.
- **Resume Capture.** Use this action to resume any paused QuickTime capture. Input can be anything. Results in anything.
- **Resume DVD Playback.** Use this action to resume playing a DVD in the DVD player.
- **Start Capture.** Use this action to start an audio or video capture. Input can be anything. Results in anything.
- **Stop Capture.** Use this action to stop an audio or video capture and returns the newly created movie.
- **Stop DVD Playback.** Use this action to stop playing a DVD in the DVD player.

Music

The Music actions allow you to work directly with music files in iTunes and with any devices you might have, like iPods. Figure D.8 provides a visual summary of the actions in this group.

The following is a list of Music actions:

- **Add Songs to iPod.** Use this action to upload iTunes songs to your iPod. You must have an iPod connected to your computer and mounted in iTunes. Input can be any set of songs passed from the previous action. Results in transferred songs.
- **Add Songs to Playlist.** Use this action to add songs to an iTunes playlist. Input can be any set of songs. Results in songs assigned to the designated playlist.
- **Ask for Songs.** Use this action to let the user choose songs from a dialog.
- **Change Case of Song Names.** Use this action to change the capitalization of song names. Input can be any set of songs. Results in those songs having their names changed to match capitalization scheme.

FIGURE D.8

The Music actions

- **Delete all iPod Notes.** Use this action to remove the notes from the specified iPod's Notes folder.
- **Eject iPod.** Use this action to eject an iPod that is currently mounted.
- **Export Movies.** Use this action to export movie files viewable in different formats (iPhone, iPod, or AppleTV). Input is any set of files or folders. Results in formatted movie files.
- **Filter iTunes Items.** Use this action to filter iTunes items (songs, playlists, sources) using certain criteria. Input can be any set of songs, playlists, or sources. Results in items filtered to match your criteria.
- **Find iTunes Items.** Use this action to search for iTunes items using various criteria. Input can be any set of songs, playlists, or sources. Results in items that match your criteria.
- **Get Selected iTunes Items.** Use this action to pass selected items to the next action. Input can be any set of songs, playlists, or sources. Results in items passed to the next action.
- **Get Specified iTunes Items.** Use this action to pass specified iTunes items to the next action. Input can be any iTunes items. Results in those items being passed to the next action.

- **Get the Current Song.** Use this action to retrieve the iTunes song currently being played. Results in the song being passed to the next song.
- **Import Audio Files.** Use this action to import audio files into iTunes. Input can be audio files of different types (AAC, Apple Lossless, AIFF, Wave, or MPEG). Results in songs being encoded and added to the iTunes library.
- **Import Files into iTunes.** Use this action t add music files to an iTunes playlist. Input can be audio files of different types (AAC, Apple Lossless, AIFF, Wave, or MPEG). Results in songs added to the iTunes library.
- **New iPod Note.** Use this action to add a new note to the iPod's Note folder. Input can be any text. Results in text.
- **New iTunes Playlist.** Use this action to create a new playlist. Input can be any set of songs. Results in a new playlist.
- **Pause iTunes.** Use this action to pause any songs being played on iTunes.
- **Play iTunes Playlist.** Use this action to play songs from an iTunes playlist. Input can be any playlist. Results in the songs from that playlist being played.
- **Remove Empty Playlists.** Use this action to remove any empty playlists from the main library in iTunes.
- **Set info of iTunes Songs.** Use this action to fill in information about iTunes songs. Input can be any set of iTunes songs. Results in those songs getting their information filled in.
- **Set iTunes Equalizer.** Use this action to set the equalizer levels and presets in iTunes.
- **Set iTunes Volume.** Use this action to set the volume level in iTunes.
- **Set Options of iTunes Songs.** Use this action to set the options for specified iTunes songs. Input can be any set of iTunes songs. Results in options set for those songs.
- **Start iTunes Playing.** Use this action to begin playing songs in iTunes.
- **Start iTunes Visuals.** Use this action to turn on visual effects in iTunes.
- **Stop iTunes Visuals.** Use this action to display visual effects in iTunes.
- **Text to Audio File.** Use this action to convert text to a sound file. Input can be any text. Results in audio files.
- **Update iPod.** Use this action to update an iPod. iPod must be connected and configured to allow automatic updating.

PDFs

The PDFs actions allow you to work directly with PDF documents. Figure D.9 provides a visual summary of the actions in this group.

FIGURE D.9

The PDFs actions

The following is a list of PDFs actions:

- **Add Grid to PDF Documents.** Use this action to place a grid of a chosen color into the passed in PDF documents. Input can be any set of PDF files. Results in PDF files.
- **Apply Quartz Filter to PDF Documents.** Use this action to apply a selected Quartz Filter to PDF documents. Input can be any set of PDF files. Results in PDF files.
- **Combine PDF Pages.** Use this action to combine pages from passed in PDFs. Input can be any set of PDF files. Results in a single PDF file.
- **Compress Images in PDF Documents.** Use this action to compress the images within the passed in PDFs. Input can be any set of PDF files. Results in PDF files.
- **Encrypt PDF Documents.** Use this action to encrypt PDF documents using the provided passwords. Input can be any set of PDF files. Results in encrypted PDF files.
- **Extract Odd & Even Pages.** Use this action to extract either the even or odd pages (or both) from passed in PDFs. Input can be any set of PDF files. Results in PDF files.
- **Extract PDF Annotations.** Use this action to extract annotations from PDFs. Input can be any set of PDF files. Results in extracted text.
- **Extract PDF Text.** Use this action to extract text from PDFs. Input can be any set of PDF files. Results in extracted text.
- **Get PDF Metadata.** Use this action to extract metadata from PDFs. Input can be any set of PDF files. Results in extracted text.

- **New PDF Contact Sheet.** Use this action to create a PDF contact sheet of images. Input can be any set of image files. Results in a PDF with scaled versions of images.
- **New PDF from Images.** Use this action to create a PDF from a number of images. Input can be any set of images (JPEG, PNG, TIFF, PDF). Results in a PDF file.
- **PDF to Images.** Use this action to create a new PDF from each page of PDFs passed to the action. Input can be any set of PDF files. Results in a set of PDF files.
- **Rename PDF Documents.** Use this action to rename passed in PDF documents. Input can be any set of PDF files. Results in PDF files.
- **Render PDF Pages as Images.** Use this action to render PDF pages as images. Input can be any set of PDF files. Results in a set of images.
- **Search PDFs.** Use this action to search for PDF documents. Input can be any files or folders to search. Results in PDFs that match the search criteria.
- **Set PDF Metadata.** Use this action to set the metadata on passed in PDFs. Input can be any set of PDF files. Results in PDF files that are rewritten in place.
- **Watermark PDF Documents.** Use this action to draw an image into each page in a PDF document. Input can be any set of PDF files. Results in PDF files. You can select the watermark image, place it on the page, and set other factors like angle and opacity.

Photos

The Photos actions allow you to work directly with images and photos in iPhoto. Figure D.10 provides a visual summary of the actions in this group.

FIGURE D.10

The Photos actions

The following is a list of Photos actions:

- **Add Photos to Album.** Use this action to add photos to an iPhoto album. Input can be any set of iPhoto images. Results in photos added to the designated album.
- **Add Thumbnail Icon to Image Files.** Use this action to create a reduced thumbnail image from the original passed in images and add them to the file as an icon. Input can be any set of image files. Results in thumbnail icons added to image files.
- **Apply ColorSync Profile to Images.** Use this action to apply a selected ColorSync profile to images. Input can be any set of image files. Results in image files.
- **Apply Quartz Color to Image Files.** Use this action to apply a selected Quartz Composition filter to a set of image files. Input can be any set of image files. Results in image files.
- **Ask for Photos.** Use this action to allow the user to choose photos from a dialog.
- **Change Type of Images.** Use this action to convert images to the selected image format. Input can be image files (BMP, GIF, JPEG, JPEG 2000, PDF, PICT, PNG, or TIFF). Results in converted image files.
- **Create Thumbnail Images.** Use this action to create thumbnail images of specified image files. Input can be image files (BMP, GIF, JPEG, JPEG 2000, PDF, PICT, PNG, or TIFF). Results in thumbnail image files.
- **Crop Images.** Use this action to crop images to specified dimensions. Input can be image files. Results in cropped images.
- **Download Pictures.** Use this action to download images from a camera. Requires a digital camera or card reader connected to your computer. Results in images downloaded to a designated folder.
- **Filter iPhoto Items.** Use this action to filter items by certain criteria. Input can be iPhoto albums and images. Results in a set of items that meet the criteria.
- **Find iPhoto Items.** Use this action to find items by certain search criteria. Input can be iPhoto albums and images. Results in a set of items that meet the search criteria.
- **Flip Images.** Use this action to flip images on the vertical or horizontal axis. Input can be any image files. Results in flipped image files.
- **Get Selected iPhoto Items.** Use this action to pass selected items to the next action. Input can be any iPhoto album or photos. Results in those items being passed to the next action.
- **Get Specified iPhoto Items.** Use this action to pass specified items to the next action. Input can be any iPhoto album or photos. Results in those items being passed to the next action.
- **Import Files into iPhoto.** Use this action to import files into an iPhoto album. Input can be image files (BMP, GIF, JPEG, JPEG 2000, PDF, PICT, PNG, or TIFF). Results in images added to the specified album.

- **New iPhoto Album.** Use this action to create a new iPhoto album. Input can be any album or photo in iPhoto. Results in a new album with any input items placed in it.
- **New PDF Contact Sheet.** Use this action to create a PDF contact sheet of images. Input can be any set of image files. Results in a PDF with scaled versions of images.
- **Open Images in Preview.** Use this action to open specified images in Preview. Input can be any image files. Results in image files opened in Preview.
- **Pad Images.** Use this action to pad the canvas around a set of images. Input can be any set of image files. Results in image files that have been padded.
- **Play iPhoto Slideshow.** Use this action to start an iPhoto slideshow. Input can be any iPhoto album. Results in a played slideshow.
- **Print Images.** Use this action to print specified images. Input can be any set of image files. Results in printed images.
- **Render Quartz Compositions to Image Files.** Use this action to render Quartz Composition files as PNG files. Input can be any set of files or folders. Results in image files.
- **Review Photos.** Use this action to review photos and mark them. Input can be any set of image files. Results in image files.
- **Rotate Images.** Use this action to rotate specified images. Input can be any set of image files. Results in rotated image files.
- **Scale Images.** Use this action to scale specified images. Scaling can be set by pixels or as a percentage of the current image. Input can be any set of image files. Results in scaled image files.
- **Take Picture.** Use this action to use an attached camera to take a picture. Results in an image file saved to a location on your Mac.
- **Take Video Snapshot.** Use this action to take a video snapshot using an attached digital camera or iSight camera. Results in an image file saved to a location on your Mac.

Text

The Text actions allow you to work directly with text files. Figure D.11 provides a visual summary of the actions in this group.

The following is a list of Text actions:

- **Ask for Text.** Use this action to display a dialog with question and answer. Input can be text. Results in text.
- **Combine Text Files.** Use this action to combine text from specified files. Input can be any set of text files. Results in text.
- **Filter Paragraphs.** Use this action to find paragraphs that meet specific criteria. Input can be any text. Results in text that meets the criteria.

- **Get Contents of TextEdit Document.** Use this action to extract the text from the front-most TextEdit document. You must have an open TextEdit document. Results in text.
- **Get Definition of Word.** Use this action to retrieve the definition of a word from the selected dictionary. Input can be text — optimally, a single word. Results in text — the definition you want.

FIGURE D.11

The Text actions

- **Get Specified Text.** Use this action to pass text to the next action. Results in text.
- **New Text File.** Use this action to create a new text file in a specified folder. Input can be any text. Results in a new text file.
- **New TextEdit Document.** Use this action to create a new TextEdit document. Input can be any text. Results in a new TextEdit document.
- **Set Contents of TextEdit Document.** Use this action to send passed in text to the front-most TextEdit document. You must have an open TextEdit document. Input can be any text. Results in text.
- **Speak Text.** Use this action to speak the specified text using Text to Speech. Input can be any text. Results in spoken words.
- **Text to Audio File.** Use this action to convert text to a sound file. Input can be any text. Results in audio files.

Utilities

The Utilities actions allow you to work directly with common utilities like CD burning, loops, and variables. Figure D.12 provides a visual summary of the actions in this group.

The following is a list of Utilities actions:

- **Ask for Confirmation.** Use this action to ask the user to select the affirmative button to continue a workflow.
- **Burn a Disc.** Use this action to burn a CD or DVD. Input can be any files or folders passed in from the previous action. Results in burned files.

FIGURE D.12

The Utilities actions

- **Choose from List.** Use this action to display a dialog with a list of items to choose from. Input can be any text list of items to choose from. Results in selected items passed to the next action.
- **Copy to Clipboard.** Use this action to copy information to the clipboard. Input can be any text passed in from the previous action. Results in anything.
- **Get Contents of Clipboard.** Use this action to retrieve text from the clipboard. Input can be any text. Results in text.
- **Get Value of Variable.** Use this action to retrieve a value of an Automator variable. Results in variable passed to the next action.

- **Hide All Applications.** Use this action to hides all applications except the frontmost one.
- **Launch Application.** Use this action to launch a chosen application.
- **Loop.** Use this action to insert a loop into a workflow.
- **Move Finder Items to Trash.** Use this action to move files and folders to the Trash. Input is a set of files or folders. Results in those items being moved to the Trash.
- **Pause.** Use this action to pause a workflow.
- **Print Finder Items.** Use this action to print specified Finder items. Input can be any files or folders. Results in printed items.
- **Quit All Applications.** Use this action to quit all open applications.
- **Quit Application.** Use this action to quit a selected application.
- **Run AppleScript.** Use this action to run a specified AppleScript.
- **Run Shell Script.** Use this action to execute a Unix shell script. Input can be text. Results in text.
- **Run Workflow.** Use this action to run an Automator workflow document.
- **Set Computer Volume.** Use this action to set the volume level on your computer.
- **Set Value of Variable.** Use this action to set a value for an Automator variable.
- **Show Growl Notification.** Use this action to show a Growl notification.
- **Speak Text.** Use this action to speak the specified text using Text to Speech. Input can be any text. Results in spoken words.
- **Spotlight.** Use this action to start a Spotlight search. Input can be text. Results in files/folders that match the search query.
- **Start Screen Saver.** Use this action to start the screen saver selected in the Desktop& Screen Saver preference pane.
- **System Profile.** Use this action to create a report detailing your computer's configuration.
- **Take Screenshot.** Use this action to take a screen shot of your computer. Results in an image file.
- **View Results.** Use this action to view the results of a previous action in a workflow. Use it to test and troubleshoot workflows.
- **Wait for User Action.** Use this action to display a dialog that waits for the user to click Continue before continuing. You can add an optional timeout.
- **Watch Me Do.** Use this action to record and play back mouse and keyboard events.

Most Used

The Most Used smart group provides easy access to your most commonly used actions. For example, in Figure D.13, you can see that I've been using certain actions a lot: opening Finder items, asking for text, setting variables, finding people with birthdays, and so on.

FIGURE D.13

The Most Used actions

Variables

The standard set of Automator variables are arranged in the following categories or groups:

- Date & Time
- Locations
- System
- Text & Data
- User
- Utilities

Date & Time

The Date & Time variables store date/time data. Figure D.14 provides a visual summary of the variables in this group.

The following is a list of Date & Time variables:

- **Current day.** Represents the day of the current month.
- **Current month.** Represents the name of the current month.
- **Current time.** Represents the current time.
- **Current weekday.** Represents the name of the current weekday (Monday, Tuesday, and so on).

- **Current year.** Represents the current year.
- **Today's date.** Represents today's date.

FIGURE D.14

The Date & Time variables

Locations

The location variables store data specific to computer locations, such as important folders. Figure D.15 provides a visual summary of the variables in this group.

FIGURE D.15

The location variables

The following is a list of Locations variables:

- **Applications.** Represents the path to the Applications folder.
- **Audio Support.** Represents the path to the Audio Supports folder.
- **Documentation.** Represents the path to the Documentation folder.
- **Documents.** Represents the path to the Documents folder.
- **Downloads.** Represents the path to the Downloads folder.
- **Favorites.** Represents the path to the Favorites folder.
- **Home.** Represents the path to the user's Home folder.
- **Movies.** Represents the path to the user's Movies folder.
- **Music.** Represents the path to the user's Music folder.
- **Path.** Specifies a new path variable.
- **Pictures.** Represents the path to the user's Pictures folder.
- **Public.** Represents the path to the user's Public folder.
- **Scripting Additions.** Represents the path to the Scripting Additions folder.
- **Scripts.** Represents the path to the Scripts folder.
- **Shared.** Represents the path to the user's Shared folder.
- **Sites.** Represents the path to the user's Sites folder.
- **User Library.** Represents the path to the user's Library folder.
- **User Preferences.** Represents the path to the user's Preferences folder.
- **Users.** Represents the path to the Users folder.
- **Utilities.** Represents the path to the Utilities folder.

System

The System variables store system-related data. Figure D.16 provides a visual summary of the variables in this group.

The following is a list of System variables:

- **Computer name.** Represents the name of the computer.
- **Computer uptime.** Represents the computer's current uptime.
- **Host name.** Represents the computer's host name.
- **IP Address.** Represents the computer's IP address.
- **Operating system version.** Represents the operating system version.

FIGURE D.16

The System variables

Text & Data

The Text & Data variables store data related to text. Figure D.17 provides a visual summary of the variables in this group.

FIGURE D.17

The Text & Data variables

The following is a list of Text & Data variables:

- **Storage.** Specifies a new storage variable.
- **Text.** Specifies a new text variable.

User

The User variables store user-related data. Figure D.18 provides a visual summary of the variables in this group.

FIGURE D.18

The User variables

The following is a list of User variables:

- **First name.** Represents the user's first name.
- **Home email address.** Represents the user's home email address.
- **Home phone number.** Represents the user's home phone.
- **Homepage.** Represents the user's homepage on the Web.
- **Last name.** Represents the user's last name.
- **MobileMe name.** Represents the user's MobileMe name.
- **My name.** Represents the full name of a user.
- **Short user name.** Represents the short user name.
- **User name.** Represents the user name.

Utilities

The Utilities variables store useful data you can use in various contexts. Figure D.19 provides a visual summary of the variables in this group.

The following is a list of Utilities variables:

- **AppleScript.** Specifies a new AppleScript variable.
- **Random identifier.** Creates a randomly generated identifier.
- **Random number.** Creates a randomly generated number.
- **Shell script.** Specifies a new shell script variable.

FIGURE D.19

The Utilities variables

Summary

In this appendix, you got more in depth exposure to the standard Automator actions and variables.

Index

Symbols

-- (double dash) characters, AppleScript Editor comments, 114–115
- (subtraction) operator, AppleScript, 472
& (concatenation) operator, AppleScript, 160, 165, 473, 473
(and) (parentheses) characters
  operator precedence order, 168
  subroutines, 216
* (multiplication) operator, AppleScript, 472
" and " (asterisk/quotes) characters, multi-line comments, 114–115
/ (division) operator, AppleScript, 472
^ (exponentiation) operator, AppleScript, 165, 473
+ (addition) operator, AppleScript, 164, 472
< (less than) character, less than operator, 163, 471
= (equality), logical operator, AppleScript, 162, 470
> (greater than) character, greater than operator, 163, 471

A

Accessibility options, enabling, 102–103
accounting, small business task, 57
action reference, Automator, 485–508
actions
  adding new, 6
  AppleScript objects, 129–130
  Apply ColorSync Profile to Images, 336–339, 342
  Ask for Finder Items, 302
  Ask for Text, 367–368
  Automated Workflows, 436–437
  Automator submissions, 72–73
  Calendar, 63–65
  Change Type of Images, 353
  collapsing/expanding, 77
  Contacts, 65
  Crop Images, 342–343
  description display, 76
  Download URLs, 95–96
  enabling/disabling, 77
  Fetch FTP Upload, 6–7
  Files & Folders, 65
  Find Finder Items, 361
  Get Folder Contents, 96, 298–299
  Get Image URLs from Webpage, 95–96
  Get Specified URLs, 95–96
  Group Mailer, 406–408
  Import Files into iPhoto, 96
  information display, 69
  Internet, 65
  Loop, 97
  Music, 65
  New Mail Message, 415–416
  New Text File, 415–416
  Open Finder Items, 105–106
  option associations, 8
  OttoMate, 435
  PDF, 65
  Photos, 65
  results display, 76
  Safari Suite, 138–139
  searches, 63–65
  Set Contents of TextEdit, 333–334
  Set Spotlight Comments for Finder Items, 13
  Spotlight, 9–16
  Text, 65
  Watch Me Do, 105
  workflow building blocks, 74
  workflow passing process, 74–76
  workflow process, 6–8
  Zap Gremlins, 397
Actions library, Automator interface element, 63–65
Activate Fonts, Font actions, 491

Add Attachments to Front Message, Mail actions, 494
Add Color Profile, Movie actions, 496
Add Grid to PDF Documents, PDF actions, 500
Add Photos to Album, Photos actions, 502
Add Songs to iPod, Music actions, 497
Add Songs to Playlist, Music actions, 497
Add Thumbnail Icon to Image Files, Photos actions, 502
Add to Font Library, Font actions, 491
Adding attachments to messages project, 416–419
Adding audio files to an iPod project, 332–334
Adding songs to a playlist project, 312–315
addition (+) operator, AppleScript, 164, 472
Add-new item alert.scpt, Folder Action Script, 245
Address Book. *See also* contacts
    automation task types, 50–51
    birthday reminders, 403–405
    contact information display, 488
    contact search, 400–403
    Contacts action, 65
    group Mail message, 488
    item filter, 487
    item retrieval, 488
    item search, 487
    smart group e-mail, 406–408
    vCard exporting, 487
alarms
    Calendar reminders, 404–405
    new e-mail message retrieval, 412–413
albums
    creating, 503
    photo addition, 502
alerts, new item, 245
alias object, AppleScript, 452–453
aliases
    creating, 490
    files and folders, 286–289
ampersand (&) character, concatenation operator, 165, 473
`and` (logical conjunction) operator, AppleScript, 161, 470
annotations, extracting from a PDF file, 421–422, 500
AppleScript
    adding audio files to an iPod, 333
    adding songs to a playlist, 314–315
Address Book contact search, 401–402
AppleScript Editor, 112–123
asking for text from the user, 377–378
automating taking pictures w/digital camera, 369
Boolean data type, 110
calendar item search, 410
`choose folder` command, 26–27
class reference, 452–466
coercion process, 168–169
combining text files, 388–389
command reference, 466–469
concatenation operators, 160, 165
conditional tests, 179–185
`considering` clause, 169–170
containment operators, 166
control statements reference, 474–478
Creating a basic workflow to accept any files, 277
Creating a basic workflow to process specific files, 275
cropping/resizing images, 344–347
data types, 110
default script folder, 27
development history, 110–112
dictionaries, 133–141
`end tell` command, 18–26
error numbers, 483
expressions, 170–172
file label stripping, 25–26
file/folder aliases, 288
filtering Finder items, 290–291
Finder dictionary display, 24–25
finding/renaming files and folders, 282–284
finding/trashing files and folders, 285–286
flipping/rotating images, 355–357
folder content retrieval, 295–296
`foldername` variable, 23–25
`get items` command, 22–25
getting items, 22–25
gremlin removal, 397
handler reference, 478–479
identifiers, 109
if/else constructions, 110
`ignoring` clause, 169–170
image format conversion, 352–353

image group color changes, 339–342
instantiated variables, 110
iTunes song information, 324–327
iTunes song playback, 309–310
iTunes volume setting, 319–320
language guide, 451–452
list data type, 110
literal expressions, 170–172
logical operators, 160–163
loops, 185–192
mathematical operators, 164–165
`myfiles` variable, 25–27
new e-mail message retrieval, 411
non case-sensitive, 109
number data type, 110
object data type, 110
object methods, 110
object reference operator, 167
object-oriented language, 126–127
objects, 125–133
`open` command, 18–21
`open location` command, 20–21
opening a file with the proper application, 297–298
opening a folder, 18–19
opening a Web page, 20–21
opening text files, 375
operator precedence, 167–168
operators, 159–170, 469–473
pausing/playing iTunes songs, 322
properties, 153–154
record data type, 110
`repeat` command, 25–26
repeat loops, 110
reserved keywords, 480–482
running, 506
saving, 27
Script Editor environment, 18–19
scripting language, 17, 109–110
server connections, 294
`set` command, 23–26
space to tab conversion, 396
specific character retrieval, 382
specific paragraph retrieval, 384–387
specific word retrieval, 379–381
Spotlight comments, 301–302
statements, 172–173
straightening curly quotes, 395
`tell application` command, 18–26
text data type, 110
text to audio file conversion, 331–332
Unicode character set, 109
Utilities variable, 511
variables, 143–152
who should use, 112
word definitions, 391–392
AppleScript Editor
AppleScript Editor menu, 118–120
button controls, 113–114
comments, 114–115
Edit menu, 121
error messages, 115–116
File menu, 120–121
Font menu, 122
footer, 113–114
Format menu, 122
Help menu, 123
information display, 113–114
launching, 113
opening a dictionary, 134
pinning to the Dock, 113
preference settings, 118–120
saving scripts, 116–117
Script menu, 121–122, 124
toolbar, 113–114
user interface elements, 113–114
View menu, 121
Window menu, 122
windowpane, 113–114
AppleScript Editor menu, AppleScript, 118–120
AppleScript Error dialog box, code errors, 116
AppleScript Studio
application framework/development environment, 255
Command Finder, 258
Currency Converter, 258
launching, 256–257
Mail Search, 258
OpenSafari project, 259–266
Plain Text Editor, 258
pros/cons, 259
Simple Shell, 258

AppleScript Suite, command reference, 466
AppleScript Utility, enabling the Scripts menu, 101
applets
  Get file size, 233
  minimal user interface application, 227
  `on run` statement, 228
  Open a Web page in Safari, 228–231
  rules, 228
  Run a shell script, 231–232
  saving as an application, 228
application launching, 40–41
application object, AppleScript, 453–454
Applications folder
  AppleScript Studio access, 256–257
  path representation, 509
applications
  AppleScript Studio, 258
  file associations, 490
  hiding, 506
  image flipping, 358–359
  image rotation, 357–358
  launching, 506
  Locations variable, 509
  opening a file, 297–299
  quitting all, 506
  quitting selected, 506
  saving a workflow as, 79
  saving an applet, 228
  user input selections, 210–211
Apply ColorSync Profile to Images, Photos actions, 336–339, 342, 502
Apply Quartz Color to Image Files, Photos actions, 502
Apply Quartz Filter to PDF Documents, PDF actions, 500
Applying color changes to groups of images project, 336–342
archives, creating ZIP file, 489
arguments, subroutines, 215–216
Ask for Finder Items, Files & Folders action, 302, 489
Ask for Movies, Movie actions, 496
Ask for Photos, Photos actions, 502
Ask for Servers, Files & Folders action, 489
Ask for Songs, Music actions, 497
Ask for Text action
  PDF contact sheet reviews, 367–368
  text retrieval, 503
Asking for Text from the User project, 376–379
assistive devices, Accessibility options, 102–103
asterisk/quotes (*" and "*) characters, multi-line comments, 114–115
attachments, e-mail addition, 416–419
audio alerts, `beep` command, 198
audio files
  adding to an iPod, 332–334
  importing to iTunes, 499
  QuickTime movie capture, 497
  text conversion, 330–332, 499, 504
Audio Support, Locations variable, 509
Audio Supports folder, path representation, 509
Automated Workflows
  action packs, 436–437
  AppleScript training, 448–449
  Automator training, 438
Automating taking pictures with a digital camera project, 368–369
automation
  accounting, 57
  Address Book tasks, 50–51
  application launching, 40–41
  backups, 43
  communications, 41–42
  data analysis, 37
  data collection, 36
  data entry, 36
  data extraction, 39
  data integration, 39–40
  data munging, 37–39
  data transformation, 39
  databases, 58
  defined, 33
  development history, 33–36
  effort saver, 44–45
  e-mails, 58
  error reduction, 45
  facilities, 57
  finance, 57
  Finder task types, 49–50
  human resources (HR), 57
  iCal task types, 50–51
  importance of, 2–3

iPhoto task types, 51
iTunes task types, 52
Mail task types, 49–50
manpower saver, 46
marketing, 57
memos, 58
money saver, 46
photographers, 55–56
podcasters, 55–56
presentations, 58
process simplification, 45
production, 57
pros/cons, 35–36
reminders, 42
reports, 58
return on investment (ROI), 59–60
rules, 48–49
Safari task types, 52
sales, 57
script launching, 40–41
security, 57
small business owners, 56–60
software developers, 53–54
spreadsheets, 58
system administrators, 54–55
system maintenance, 43
task types, 48–49
templates, 58
time saver, 43–44
versus mechanization, 34
videographers, 55–56
visual designers, 55–56
who should use, 3, 31–33
Automator
action information display, 69
action search, 63–65
action submissions, 72–73
actions listing, 6–8, 485–508
Actions library, 63–65
adding audio files to an iPod, 333
adding songs to a playlist, 313–314
Address Book contact search, 400–401
Apply ColorSync Profile to Images action, 336–339, 342
asking for text from the user, 376–377
automating taking pictures w/digital camera, 368–369
backing up documents, 6–7
birthday reminders, 403–405
Calendar actions, 63–65, 486–487
calendar item search, 409
changing case of song names, 328–329
combining e-mail messages, 413–414
combining text files, 387–388
Contacts actions, 65, 487–488
Creating a basic workflow to accept any files, 276–277
Creating a basic workflow to process specific files, 273–275
cropping/resizing images, 342–343
CSV data to SQL conversion, 58
Date & Time variables, 507–508
Description button, 76
downloading specific URLs, 423–425
e-mail attachment addition, 416–419
extracting text from a PDF file, 419–421
file/folder aliases, 287–288
Files & Folders actions, 65, 488–490
filtering Finder items, 289–290
filtering iTunes songs, 316–317
Finder selection workflow, 70–71
finding/renaming files and folders, 280–281
finding/trashing files and folders, 284–285
first time startup, 61
flipping/rotating images, 354–355
Folder Action plug-ins, 250–254
folder content retrieval, 294–295
Fonts actions, 490–492
Footer, 67–69
gremlin removal, 396–397
group e-mails, 406–408
hiding/displaying libraries, 63
Icon view, 10
image format conversion, 352
image group color changes, 336–339
image search, 359–360
Internet actions, 65, 492–494
iPhoto image export, 364–365

Automator *(continued)*
iPhoto image import, 361–364
iTunes song information, 323–324
iTunes song playback, 306–309
iTunes volume setting, 319
Library pane, 5
List view, 10–11
Locations variables, 508–509
loops, 94–98
Mail actions, 494–495
manual event recording, 102–106
manually recording workflows, 63
media file display, 63
Most Used actions, 506–507
Movies actions, 495–497
Music actions, 65, 497–499
new e-mail message retrieval, 410–411
opening a file with the proper application, 297
opening text files, 373–375
Options button, 76
pausing/playing iTunes songs, 322
PDF actions, 65, 499–501
PDF contact sheet reviews, 366–367
PDF keyword search, 9–16
PDF page to image conversion, 422–423
Photos actions, 65, 501–503
plugins, 98–101
Record button, 103–106
removing empty playlists, 327–328
resizing actions/variables pane, 69
Results button, 76
running workflows, 63
scaling images before cropping, 348–349
server connections, 293–294
Smart Groups, 67–68
space to tab conversion, 395–396
specific paragraph retrieval, 383–384
Spotlight Comments, 12–13, 15–16, 299–301
stopping workflows, 63
straightening curly quotes, 394
System variables, 509–510
Table view, 10–11
templates, 81–82
Text & Data variables, 510–511
Text actions, 65, 503–504
text to audio file conversion, 330–331
thumbnail creation, 349–351
Toolbar, 4–5, 62–63
user interface, 4–6, 62–69
User variables, 511
Utilities actions, 505–506
Utilities variables, 511–512
variable information display, 69
Variable list display, 69
variables, 8–9, 507–512
Variables library, 65–67
word definitions, 390–394
Workflow Log display, 69
Workflow pane, 66–67
workflows, 5–6, 9–17

B

backgrounds, desktop, 237–239
backups
automation use, 43
documents, 6–7
software developers, 54
BBEdit
gremlin removal, 396–397
space to tab conversion, 395–396
straightening curly quotes, 394
`beep` command, user input, 198
binary operators, AppleScript, 159
birthdays
Address Book search, 488
contact search criteria, 403–405
Mail greeting, 495
Boolean object, AppleScript, 110, 454–455
branches, conditional tests, 180
Browse Movies, Movie actions, 496
build scripts, software developers, 54
button arrays, user input, 204–206
button controls, AppleScript Editor, 113–114
`buttons` clause, `display dialog` command, 200–202
buttons, user input value capture, 204–206

C

Calendar
birthday reminder, 404–405
item searches, 409–410
plugin, 99–100
Calendar actions, Automator, 63–65, 486–487
card readers, downloading images, 502
Change Case of Song Names, Music actions, 497
Change Type of Images, Photos actions, 353, 502
Changing case of song names project, 328–330
`character` command, specific character retrieval, 382–383
character sets, Unicode, 109
Check if folder exists, sample script, 193–194
Checking for e-mail validity, sample script, 173–175
`choose application` command, 210–211
`choose file` command, 207–208
`choose file name` command, user input file saves, 207–208
`choose folder` command, 26–27, 206–207
`choose from list` command, 208–210
Choose from List, Utilities actions, 505
class object, AppleScript, 455–456
class property
alias object, 452
application object, 453
Boolean object, 454
class object, 455
constant object, 456
date object, 457
integer object, 459
list object, 459
number object, 461
real object, 462
record object, 464
text object, 465
class reference, AppleScript, 452–466
Clipboard
AppleScript command reference, 466
content retrieval, 505
copying information to, 505
Close-close sub-folders.scpt, Folder Action Script, 245
CNET, AppleScript downloads, 447–448
coercion
alias object, 453
application object, 454
Boolean object, 454–455
class object, 456
constant object, 456
date object, 457–458
file object, 458
integer object, 459
list object, 460
number object, 461
POSIX file, 461–462
real object, 463
record object, 465
script object, 465
text object, 466
color changes, applying to image groups, 336–342
color profiles, QuickTime movies, 496
colors
AppleScript Editor preferences, 119
FavoriteColor variable, 90
Finder label items, 12, 15
ColorSync profiles, images, 502
Combine Mail Messages, Mail actions, 495
Combine PDF Pages, PDF actions, 500
Combine Text Files, Text actions, 503
Combining Mail Messages project, 413–416
Combining Text Files project, 387–389
Command Finder, AppleScript Studio application, 258
command reference, AppleScript, 466–469
comma-separated values (CSV), SQL conversion, 58
comments
adding to Spotlight Comments, 12–13
AppleScript Editor, 114–115
asterisk,/quotes (*" and "*) characters, 114–115
displaying, 245
double dash (--) characters, 114–115
file/folder, 299–302
Finder item, 490
image information addition, 245
PDF annotation extraction, 421–422
communications, 41–42
comparisons, considering/ignoring clauses, 169–170
Compress Images in PDF Documents, PDF actions, 500

Computer name, System variable, 509
Computer uptime, System variable, 509
computers, volume settings, 506
concatenation (&) operator, AppleScript, 160, 165, 473
conditional tests
  AppleScript, 179–185
  `if`, 180–181
  `if-else`, 181–183
  `if-else` if-else, 184–185
  subroutines, 218
Connect to Servers, Files & Folders action, 489
Connecting to a server project, 293–294
`considering` clause, operators, 169–170
constant object, AppleScript, 456
contact sheets
  creating, 503
  image reviews, 365–368
  PDF creation, 501
contacts. *See also* Address Book
  Address Book information display, 488
  Address Book search, 400–403, 487
  birthday as contact search criteria, 403–405
  birthday search, 488
  smart group e-mail, 406–408
  vCard exporting, 487
Contacts actions, Automator, 65, 487–488
containers, objects, 172
containment operator, AppleScript, 166, 473
containment view, dictionaries, 139–140
content pane, dictionaries, 136
continuation character, scripts, 189
control statements, AppleScript reference, 474–478
Convert Quartz QuickTime Movies, Movie actions, 496
Converting a basic workflow to accept any files project, 276–279
Converting images project, 352–353
Converting text to audio files project, 330–332
Convert-PostScript to PDF.scpt, Folder Action Script, 245
`copy` command
  adding items to a list, 149
  variable creation method, 144
Copy files between directories, sample script, 155–156
Copy Finder Items, Files & Folders action, 489
Copy to Clipboard, Utilities actions, 505
`count` command, counting list items, 149
Count items in a folder, subroutine, 225–226
Create a folder, subroutine, 221–223
Create Archive, Files & Folders action, 489
Create Thumbnail Images, Photos actions, 502
Creating a basic workflow to process specific files project, 273–275
Creating a Group Mailer project, 406–408
Creating aliases for files and folders project, 286–289
Creating thumbnails project, 349–351
Crop Images, Photos actions, 342–343, 502
Cropping and resizing images project, 342–349
Currency Converter, AppleScript Studio application, 258
current date, object returns, 132–133
Current Day, Date & Time variable, 507
Current Month, Date & Time variable, 507
Current Time, Date & Time variable, 507
Current Weekday, Date & Time variable, 507
Current Year, Date & Time variable, 508

D

data analysis, 37
data collection, 36
data entry, 36
data extraction, 39
data integration, 39–40
data munging, 37–39
data requests, software developers, 54
data transformation, 39
data types, AppleScript, 110
databases, small business task, 58
Date & Time variables, Automator, 8, 87–88, 507–508
date object, AppleScript, 457–458
date string property, date object, 457
dates, current date return, 132
day property, date object, 457
day, current day display, 507
Deactivate Fonts, Font actions, 491
debugging, workflows, 10–11
`default` clause, `display dialog` command, 200–201
`default answer` clause, `display dialog` command, 203–204

`default location` clause, `choose folder` command, 206–207
Delete all iPod Notes, Music actions, 498
Delete iCal Events, Calendar action, 486
Description button, Automator, 76
desktop backgrounds
  picture selection, 490
  setting from a droplet, 237–239
Develop section, AppleScript Editor preferences, 119
Developer folder, AppleScript Studio access, 256–257
dialog boxes
  button arrays, 204–206
  `display dialog` command, 200–202
dictionaries
  AppleScript Editor preferences, 121
  AppleScript knowledge repositories, 133
  content pane, 136
  displaying in Finder, 24–25
  icons, 137
  Image Events, 339–340
  information lookups, 141–142
  information panes, 136
  opening, 134–135
  pane sizing, 137–138
  Safari Suite, 138–139
  Standard Suite view, 138–140
  suite pane, 136
  switching between views, 138
  Text Suite view, 138
  viewing by containment, 139–140
  viewing by inheritance, 140–141
  viewing by suite, 138–139
  word definition retrieval, 504
Dictionary application, word definition retrieval, 390–391
digital cameras
  automating picture taking, 368–369
  downloading images, 502
  taking pictures, 503
  video snapshots, 503
directories, copying files between, 155–156
discussion groups
  Automator World, 432–433
  MacScripter, 445–446
disk image
  creating, 490
  mounting, 489
Disk-Folder-File Suite, Image Events, 339
disks, ejecting, 489
`display dialog` command, user input, 200–204
Display Mail Messages, Mail actions, 495
Display Web Pages, Internet actions, 493
div (integral division) operator, AppleScript, 472
dividers, dictionary pane sizing, 137–138
division (`/`) operator, AppleScript, 164–165, 472
Documentation folder, path representation, 509
Documentation, Locations variable, 509
Documents folder, path representation, 509
documents
  backing up, 6–7
  TextEdit font display, 492
Documents, Locations variable, 509
double dash (--) characters, single-line comments, 114–115
Download Pictures, Photos actions, 502
Download URLs, Internet actions, 95–96, 493
Downloading Specific URLs project, 423–425
Downloads folder, path representation, 509
Downloads, Locations variable, 509
droplets
  `applet w/on open` statement, 234
  combining text files, 389
  Get file size, 234–237
  Set the desktop background, 237–239
Duplicate Finder Items, Files & Folders action, 489
DVD, QuickTime playback, 497

E

eBooks, PDF keyword search, 9–16
Edit menu, AppleScript, 121
Editing preferences, AppleScript Editor menu, 119
Educate Quotes function, straightening curly quotes, 394
effort saver, automation reason, 44–45
Eject Disk, Files & Folders action, 489
Eject iPod, Music actions, 498

e-mail. *See also* Mail
attachment addition, 416–419
automation task types, 49–50
combining messages, 413–416
group mailers, 406–408
group messages, 488
new message retrieval, 410–413
small business task, 58
validity checking, 173–175
e-mail addresses, exporting, 194–195
Enable or Disable Tracks, Movie actions, 496
Encrypt PDF Documents, PDF actions, 500
encryption, PDF documents, 500
`end tell` command, AppleScript, 18–26
`entab` command, space to tab conversion, 396
equality (`=`), logical operator, 162, 470
equalizer, iTunes settings, 499
error messages, AppleScript Editor, 115–116
error numbers, AppleScript, 483
error reduction, automation advantage, 3, 45
Event Log history, AppleScript Editor preferences, 120
Event Summary, Calendar action, 486
events
deleting from iCal, 486
iCal creation, 487
iCal retrieval, 487
iCal summary, 486
manual recording, 102–106
new e-mail message retrieval, 412–413
viewing today's, 156–157
exponentiation (`^`) operator, AppleScript, 165, 473
Export e-mail addresses, sample script, 194–195
Export Font Files, Font actions, 491
Export Movies, Movie actions, 496, 498
Export vCards, Contacts action, 487
Exporting images from iPhoto project, 364–365
expressions
lexical elements, 170
literal expressions, 170–172
Extract Odd & Even Pages, PDF actions, 500
Extract PDF Annotations, PDF actions, 500
Extract PDF Text, PDF actions, 500
Extracting PDF Pages project, 422–423
Extracting Text from PDFs project, 419–422

F

facilities, small business task, 57
factorial calculation, recursive subroutine, 218–219
FavoriteColor variable, workflow colors, 90
Favorites folder, path representation, 509
Favorites, Locations variable, 509
Fetch FTP Upload action, backing up documents, 6–7
file deletions, system administrator, 54–55
File menu, AppleScript, 120–121
file object, AppleScript, 458
files
aliases, 286–289, 490
AppleScript command reference, 467
AppleScript Editor preferences, 120–121
application associations, 490
copying between directories, 155–156
duplicating, 489
Finder plugin processing, 99
finding/renaming, 280–284
finding/trashing, 284–286
font associations, 491–492
font file removal, 492
font validation, 492
get file size with an applet, 233
get file size/filename with a droplet, 234–237
importing, 96
importing to iTunes, 499
iPhoto import, 502
label stripping, 25–26
labels, 489
moving, 489
moving to Trash, 490
new item alert, 245
Open Finder Items action, 105–106
opening text files, 373–375
opening w/proper application, 297–299
processing workflow, 273–275
renaming, 2, 6
specific word retrieval, 379–381
Spotlight comments, 299–302
user input selections, 207–208
workflow conversion, 276–279
ZIP archive creation, 489

Files & Folders actions, Automator, 65, 488–490
Filter Address Book Items, Contacts action, 487
Filter Articles, Internet actions, 493
Filter Finder Items, Files & Folders action, 489
Filter Font Book Items, Font actions, 491
Filter Fonts by Font Type, Font actions, 491
Filter iCal Items, Calendar action, 486
Filter iPhoto Items, Photos actions, 502
Filter iTunes Items, Music actions, 498
Filter Mail Items, Mail actions, 495
Filter Paragraphs, Text actions, 503
Filter URLs, Internet actions, 493
Filtering Finder items project, 289–293
Filtering iTunes songs project, 316–318
finance, small business task, 57
Find Address Book Items, Contacts action, 487
Find Finder Items, Files & Folders action, 361, 489
Find Font Book Items, Font actions, 491
Find iCal Items, Calendar action, 486
Find iPhoto Items, Photos actions, 502
Find Mail Items, Mail actions, 495
Find People with Birthdays, Contacts action, 488
Finder
  automation task types, 49–50
  close all subfolders, 245
  copying items, 489
  dictionary display, 24–25
  e-mail attachment addition, 416–419
  extracting text from a PDF file, 419–422
  filtering items, 289–293
  item display, 490
  item duplication, 489
  item filtering, 489
  item labels, 489
  item retrieval, 489
  item searches, 489
  labels, 12, 15
  moving items, 489
  moving items to Trash, 490, 506
  opening a folder script, 18–19
  opening a Web page script, 20–21
  opening items, 490
  plugins, 98–99
  printing items, 506
  renaming items, 490
  retrieving AppleScript items, 22–25
  selection workflow, 70–71
  Smart Folder creation, 15–16
  sorting items, 490
  user input folder selections, 206–207
Finding files and folders and renaming them project, 280–284
Finding files and folders and trashing them project, 284–286
Finding People with Birthdays project, 403–405
Finding Specific Calendar Items project, 409–410
Finding Specific Contacts in Address Book project, 400–403
Finding Specific Images project, 359–361
firewall rules, system administrator, 55
First name, User variable, 511
`flip` command, flipping images, 355–356
Flip Images, Photos actions, 502
Flipping and rotating images project, 354–359
Folder Action Scripts
  Add-new item alert.scpt, 245
  Close-close sub-folders.scpt, 245
  Convert-PostScript to PDF.scpt, 245
  Image-Add Icon.scpt, 245
  Image-Duplicate as JPEG.scpt, 245
  Image-Duplicate as PNG.scpt, 245
  Image-Duplicate as TIFF.scpt, 245
  Image-Flip Horizontal.scpt, 245–246
  Image-Flip Vertical.scpt, 245
  Image-Info to Comment.scpt, 245
  Image-Rotate Left.scpt, 245
  Image-Rotate Right.scpt, 245
  Image-Show comments in dialog.scpt, 245
  rules, 245
  updating existing scripts, 246–250
Folder Actions
  AppleScript/folder associations, 241
  enabling, 242–244
  Folder Action Scripts, 244–250
  plug-ins, 250–254
Folder Actions Setup application, 242–243
`foldername` variable, AppleScript, 23–25

folders
alias creation, 490
aliases, 286–289
AppleScript selections, 26–27
checking for existing, 193–194
content retrieval, 294–296, 489
creating, 490
`creating with repeat with` loop, 187–189
creating with Today's date variable, 87–88
creation subroutine, 221–223
duplicating, 489
finding/renaming, 280–284
item counting subroutine, 225–226
mapping with New Path variable, 88
moving, 489
moving to Trash, 490
new item alert, 245
opening script, 18–19
saved AppleScript scripts, 27
Spotlight comments, 299–302
user input selections, 206–207
views, 490
Font Book
filtering items, 491
font selections, 492
item retrieval, 492
item search, 491
Font Library, font addition, 491
Font menu, AppleScript, 122
fonts
activating, 491
adding to Font Library, 491
AppleScript Editor preferences, 119, 122
deactivating, 491
exporting, 491
file associations, 491–492
file removal, 492
file validation, 492
information display, 491
Postscript name, 492
TextEdit document display, 492
Fonts actions, Automator, 490–492
Footer
Automator interface element, 67–69
information display, 69
resizing actions/variables pane, 69
Smart Groups, 67–68
Variable list display, 69
Workflow Log, 69
footers, AppleScript Editor, 113–114
Format menu, AppleScript, 122
Formatting preferences, AppleScript Editor menu, 119
forums
AppleScript for Absolute Beginners, 445–446
Automator World, 432–433
MacScripter, 445–446
frontmost property, application object, 453

G

General preferences, AppleScript Editor menu, 118
`get` command, variable value display, 146
Get Attachments from Mail Messages, 495
Get Contact Information, Contacts action, 488
Get Contents of Clipboard, Utilities action, 505
Get Contents of TextEdit Document, Text actions, 504
Get Current Webpage from Safari, Internet actions, 493
Get Definition of Word, Text actions, 504
Get Enclosure URLs from Articles, Internet actions, 493
Get Feeds from Mail, Internet actions, 493
Get Feeds from Safari, Internet actions, 493
Get Feeds from URLs, Internet actions, 493
Get file size, applet, 233
Get file size, droplet, 234–237
Get Files for Fonts, Font actions, 491
Get Folder Contents, Files & Folders action, 96, 298–299, 489
Get Font Info, Font actions, 491
Get Fonts from Font Files, Font actions, 492
Get Fonts from TextEdit Document, Font actions, 492
Get Image URLs from Articles, Internet actions, 493
Get Image URLs from Webpage, Internet actions, 95–96, 493
`get items` command, AppleScript, 22–25
Get Link URLs from Articles, Internet actions, 493
Get Link URLs from Webpages, Internet actions, 493
Get New Mail, Mail actions, 495
Get PDF Metadata, PDF actions, 500
Get Permalinks of Articles, Internet actions, 493
Get Postscript name of Font, Font actions, 492

Get Selected Address Book Items, Contacts action, 488
Get Selected Finder Items, Files & Folders action, 489
Get Selected Font Book Items, Font actions, 492
Get Selected iPhoto Items, Photos actions, 502
Get Selected iTunes Items, Music actions, 498
Get Selected Mail Items, Mail actions, 495
Get Specified Address Book Items, Contacts action, 488
Get Specified Finder Items, Files & Folders action, 489
Get Specified iCal Items, Calendar action, 487
Get Specified iPhoto Items, Photos actions, 502
Get Specified iTunes Items, Music actions, 498
Get Specified Mail Items, Mail actions, 495
Get Specified Movies, Movie actions, 496
Get Specified Servers, Files & Folders action, 489
Get Specified Text, Text actions, 504
Get Specified URLs, Internet actions, 95–96, 493
Get Text from Articles, Internet actions, 493
Get Text from Webpage, Internet actions, 494
Get the Current Song, Music actions, 499
Get Value of Variable, Utilities actions, 505
Getting a Specific Character of Text project, 382–383
Getting a Specific Paragraph of Text project, 383–387
Getting a Specific Word project, 379–382
Getting Folder Contents project, 294–296
Getting the Definition of a Word project, 390–393
`global` command, 152
global variables, setting, 152
Google Code, OttoMate actions, 435
greater than (>) character, logical operator, 163, 471
greater than or equal to (Option+.), logical operator, 163, 471
grids, PDF documents, 500
Group Mailer, Contacts action, 406–408, 488, 495
groups, versus Smart Groups, 68
Growl notifications, showing, 506

H

handler reference, AppleScript, 478–479
Help menu, AppleScript, 123
Hide All Applications, Utilities actions, 506
Hide Library button, Automator Toolbar, 62–63
Hint Movies, Movie actions, 496
History preferences, AppleScript Editor menu, 120
Home email address, User variable, 511
Home folder, path representation, 509
Home phone number, User variable, 511
Home, Locations variable, 509
Homepage, User variable, 511
Host name, System variable, 509
human resources (HR), small business task, 57
HyperCards, AppleScript development, 110–111

I

iCal
- automation task types, 50–51
- birthday reminder, 404–405
- Calendar plugins, 99–100
- calendar summary, 486
- deleting events, 486
- event creation, 487
- item filtering, 486
- item searches, 409–410, 486
- new calendar creation, 487
- new e-mail message retrieval, 412–413
- retrieving events, 487
- to do item creation, 487
- today's event viewing, 156–157

iCal Alarm plugins, saving a workflow, 99–100
Icon view, workflow results display, 10
icons
- adding thumbnail icon to any dropped image, 245
- dialog box display, 202
- dictionaries, 137

id property
- application object, 453
- text object, 465

identifiers
- AppleScript, 109
- variable naming conventions, 144–145

`if` statement, conditional test, 180–181, 474–475
if/else constructions, AppleScript, 110
`if-else`, two-value conditional test, 181–183
`if-else if-else`, multi-value conditional test, 184–185
`ignoring` clause, operators, 169–170
Image Capture plugin, Image Capture application extender, 101
Image Events Suite, Image Events, 339

Image Events, script suites, 339–340
Image Suite, Image Events, 339
Image-Add Icon.scpt, Folder Action Script, 245
Image-Duplicate as JPEG.scpt, Folder Action Script, 245
Image-Duplicate as PNG.scpt, Folder Action Script, 245
Image-Duplicate as TIFF.scpt, Folder Action Script, 245
Image-Flip Horizontal.scpt, Folder Action Script, 245–246
Image-Flip Vertical.scpt, Folder Action Script, 245
Image-Info to Comment.scpt, Folder Action Script, 245
Image-Rotate Left.scpt, Folder Action Script, 245
Image-Rotate Right.scpt, Folder Action Script, 245
images
  adding information to Spotlight comments field, 245
  color change application, 336–342
  ColorSync profiles, 502
  cropping, 342–349, 502
  dimensional crops, 348–349
  displaying Spotlight comments, 245
  downloading, 95–96
  downloading from a camera, 502
  duplicating as JPEG format, 245
  duplicating as PNG format, 245
  duplicating as TIFF format, 245
  flipping, 502
  flipping horizontally, 245–246
  flipping vertically, 245
  flipping/rotating, 354–359
  format conversions, 352–353
  iPhoto export, 364–365
  iPhoto import, 361–364
  padding, 503
  PDF contact sheet reviews, 365–368
  PDF creation, 501
  PDF document compression, 500
  PDF page conversion, 422–423
  PDF page rendering, 501
  Photoshop Action Pack, 434–435
  previewing, 503
  printing, 503
  Quartz Composition filter, 502–503
  resizing, 342–349
  retrieving from a Webpage, 95
  reviewing, 503
  rotating, 503
  rotating left/right, 245
  scaling, 503
  scaling before cropping, 348–349
  searches, 359–361
  selections, 502
  setting desktop background from a droplet, 237–239
  thumbnails, 502
  URL retrieval, 493
Image-Show comments in dialog.scpt, Folder Action Script, 245
Import Audio Files, Music actions, 499
Import Files into iPhoto, Photos actions, 96, 502
Import Files into iTunes, Music actions, 499
Importing images to iPhoto project, 361–364
inequality (Option+=), logical operator, 162, 470–471
`info for` command, get file size with an applet, 233
information panes, dictionaries, 136
inheritance view, dictionaries, 140–141
Inspect the Trash, sample script, 175–178
integer object, AppleScript, 459
integral division (div) operator, AppleScript, 165, 472
Interface Builder, OpenSafari project, 262–264
Internet, AppleScript command reference, 467
Internet actions, Automator, 65, 492–494
IP address variable, dynamic text file naming, 88–89
IP Address, System variable, 509
iPhoto
  album image addition, 502
  automation task types, 51
  image editor, 339
  image export, 364–365
  image import, 96, 361–364, 502
  item filtering, 502
  item search, 502
  item selections, 502
  new album creation, 503
  slideshows, 503
iPod
  audio file addition, 332–334
  deleting notes, 498
  ejecting, 498
  note creation, 499
  song addition, 497
  updating, 499

`is equal`, equality operator, 162
`is equal to`, equality operator, 162
iSight camera
automating picture taking, 369
video snapshots, 503
items
Address Book filter, 487
Address Book retrieval, 488
Address Book search, 487
Calendar searches, 409–410
comments, 490
copying Finder, 489
counting in a folder subroutine, 225–226
displaying in Finder, 490
duplicating files/folders, 489
filtering, 289–293
filtering Font Book, 491
filtering iTunes, 498
Finder filtering, 489
Finder retrieval, 489
Finder search, 489
Font Book retrieval, 492
Font Book search, 491
iCal filtering, 486
iCal searches, 486
iPhoto filtering, 502
iPhoto search, 502
iPhoto selections, 502
iTune search, 498
iTunes retrieval, 498
labeling, 489
list selections, 505
Mail filtering, 495
Mail search, 495
Mail selections, 495
moving Finder, 489
moving to the Trash subroutine, 223–224
moving to Trash, 490, 506
opening, 490
Printing from Finder, 506
renaming, 490
retrieving from a list, 146–151
retrieving in AppleScript, 22–25
sorting, 490
to do, 487, 495
iTunes
adding song to playlist, 497
adding songs to a playlist, 312–315
automation task types, 52
changing case of song names, 328–329
Doug's AppleScripts, 446–447
equalizer settings, 499
filtering items, 498
filtering songs, 316–318
importing audio files, 499
item retrieval, 498
item search, 498
music file import, 499
pausing song playback, 499
pausing/playing songs, 321–323
playlist creation, 499
playlist playback, 499
random track playback script, 154–155
removing empty playlists, 327–328, 499
song information, 323–327, 499
song options, 499
song playback, 306–312, 499
song retrieval, 499
visuals, 499
volume setting, 318–321, 499

J

JPEG format, duplicating any image as, 245

K

`keystroke` command, word definitions, 392–393
key-value pairs, records, 151
keywords
AppleScript reserved, 480–482
container specification, 172
workflow search, 9–16

L

Label Finder Items, Files & Folders action, 489
labels
Finder items, 12, 15, 489
stripping from files, 25–26
Last name, User variable, 511

Launch Applications, Utilities actions, 506
length property
list object, 459
record object, 464
text object, 465
less than (<) character, logical operator, 163, 471
less than or equal to (Option+,) logical operator, 163, 471
libraries
hiding/displaying, 63
subroutines, 219–221
Variable, 65–67
Library pane, Automator, 5
Library window, opening a dictionary, 135
line wrapping, AppleScript Editor preferences, 119
link URL, RSS feed retrieval, 493
list object, AppleScript, 459–460
List view, workflow results display, 10–11
lists
adding items, 149
AppleScript data type, 110
counting items, 149
item retrieval, 146–151
item selections, 505
`repeat with` loop, 191–192
user input selections, 208–210
value resetting, 148–149
literal expressions, AppleScript, 170–172
local scope, variables, 152
Locations variables, Automator, 8, 88, 508–509
log files
`detecting with repeat while` loop, 190–191
system administrator, 54
logical conjunction, `and` operator, 161, 470
logical disjunction, `or` operator, 161, 470
logical operators, AppleScript, 160–163, 470–471
Loop, Utilities actions, 97, 506
loops
delay increments, 97
logical construct, 94
`repeat` statement, 475–477
repeating forever, 185
repeating until something is true, 189–190
repeating w/defined start/stop values, 187–189
repeating w/list, 191–192
repeating while something is true, 190–191
repeating x number of times, 186
repetitive expressions, 180
running, 97
setup, 95–98
subroutines, 218
workflow insertion, 506

M

Macintosh, Accessibility options setup, 102–103
MacResearch, AppleScript tutorials, 439–441
MacScripter
AppleScript forums, 445–446
AppleScript tutorials, 441–443
MacTech, AppleScript articles, 442
Mail. *See also* e-mail
attachment addition, 494
attachment retrieval, 495
automation task types, 49–50
birthday greetings, 495
check for new messages, 495
combining messages, 495
group messages, 488, 495
item filtering, 495
item search, 495
item selections, 495
message display, 495
new message creation, 495
new message retrieval, 410–413
RSS feed retrieval, 493
sending outgoing messages, 495
to do item creation, 495
Mail actions, Automator, 494–495
Mail Search, AppleScript Studio application, 258
mailing lists, Automator-dev, 434
manpower saver, automation reason, 46
Mark Articles, Internet actions, 494
marketing, small business task, 57
mathematical operators, AppleScript, 164–165, 472–473
mechanization, versus automation, 34
Media button, Automator Toolbar, 4–5, 62–63
media files, Automator display, 63

memos, small business task, 58
metadata
PDF files, 500–501
Spotlight Comments, 12–13, 15
miscellaneous, AppleScript command reference, 468
MobileMe name, User variable, 511
mod (remainder) operator, AppleScript, 472
modulus (mod) operator, AppleScript, 165
money saver, automation reason, 46
month property, date object, 457
month, current month display, 507
Most Used actions, Automator, 506–507
Mount Disk Image, Files & Folders action, 489
Move an item to the Trash, subroutine, 223–224
Move Finder Items to Trash
Files & Folders action, 490
Utilities actions, 506
Move Finder Items, Files & Folders action, 489
movies. *See also* QuickTime movies
browsing, 496
exporting, 496, 498
Locations variable, 509
playback, 497
retrieving, 496
selecting, 496
Movies actions, Automator, 495–497
Movies folder, path representation, 509
multi-line comments, asterisk/quotes (*" and "*) characters, 114–115
multiplication (`*`) operator, AppleScript, 472
multi-value conditional test, `if-else if-else`, 184–185
Music actions, Automator, 65, 497–499
music files, iTunes import, 499
Music folder, path representation, 509
Music, Locations variable, 509
My name, User variable, 511
`myfiles` variable, AppleScript, 25–27

N

name property, application object, 454
negation, `not` operator, 161, 470
New Aliases, Files & Folders action, 490
New Audio Capture, Movie actions, 497
New Calendar, Calendar action, 487
New Disk Image, Files & Folders action, 490
New Folder, Files & Folders action, 490
New iCal Events, Calendar action, 487
New iPhoto Album, Photos actions, 503
New iPod Note, Music actions, 499
New iTunes Playlist, Music actions, 499
New Mail Message, Mail actions, 415–416, 495
New Path variable, folder mapping, 88
New PDF Contact Sheet
PDF actions, 501
Photos actions, 503
New PDF from Images, PDF actions, 501
New Safari Documents, Internet actions, 494
New Screen Capture, Movie actions, 497
New Text File, Text actions, 415–416, 504
New Text variable, name/value assignment, 89–90
New TextEdit Document, Text actions, 504
New To Do Item
Calendar action, 487
Mail actions, 495
New Video Capture, Movie actions, 497
`not` (negation), logical operator, 161, 470
number object, AppleScript, 460–461
number
data type, 110
subroutines, 216–217

O

object reference operator, AppleScript, 167
objects
actions, 129–130
alias, 452–453
AppleScript class reference, 452–466
AppleScript data type, 110
application, 453–454
Boolean, 454–455
class, 455–456
coercion, 168–169
constant, 456
containers, 172
current date return, 132–133
date, 457–458
described, 125–126

objects *(continued)*
file, 458
instantiation of a class, 126–127
integer, 459
list, 459–460
number, 460–461
POSIX file, 461–462
properties, 129–130
real, 462–464
record, 463–465
script, 465
text, 465–466
top-level scripts, 130–131
types, 127–128
`on open` statement, droplets 234
`on run` statement, applets 228
online documentation, AppleScript Editor preferences, 123
`open` command, opening files, 18–21, 375
Open a Web page in Safari, applet, 228–231
Open Finder Items, Files & Folders action, 105–106, 490
Open Images in Preview, Photos actions, 503
`open location` command, AppleScript, 20–21
Opening files with the proper application project, 297–299
Opening Text Files project, 373–375
OpenSafari project, AppleScript Studio, 259–266
Operating system version variable, dynamic text file naming, 88–89
Operating system version, System variable, 509
operators
addition (+), 164, 472
`and` (logical conjunction), 161, 470
AppleScript reference, 469–473
binary, 159
Boolean objects, 454
class object, 456
coercion process, 168–169
concatenation (&), 160, 165, 473
`considering` clause, 169–170
constant object, 456
containment, 473
date object, 457
division (/), 164–165, 472
equality (=), 162, 470
exponentiation (^), 165, 473
greater than (>), 163, 471
greater than or equal to (Option+.), 163, 471
`ignoring` clause, 169–170
inequality (Option+=), 162, 470–471
integer objec, 459
integral division (div), 165, 472
`is equal`, 162
`is equal to`, 162
less than (<), 163, 471
less than or equal to (Option+,), 163, 471
logical, 160–163, 470–471
mathematical, 164–165, 472–473
modulus (mod), 165
multiplications (*), 164, 472
`not` (negation), 161, 470
number object, 461
object reference, 167
`or` (logical disjunction), 161, 470
precedence order, 167–168
real object, 463
record object, 464
remainder (mod), 472
subtraction (-), 164, 472
unary, 159
Options button, Automator, 76
options, action associations, 8
`or` (logical disjunction) operator, 161, 470
`osascript` command, pausing/playing iTunes songs, 323
OttoMate, Automator actions, 435

P

package building scripts, software developers, 54
Pad Images, Photos actions, 503
`paragraph` command, specific paragraph retrieval, 384–387
paragraphs
filtering, 503
specific paragraph retrieval, 383–387
parentheses (and) characters
operator precedence order, 168
subroutines, 216

Path folder, path representation, 509
Path, Locations variable, 509
Pause Capture, Movie actions, 497
Pause DVD Playback, Movie actions, 497
Pause iTunes, Music actions, 499
Pause, Utilities actions, 506
Pausing and Playing iTunes project, 321–323
PDF actions, Automator, 65, 499–501
PDF contact sheets
  creating, 501, 503
  photo reviews, 365–368
PDF documents
  encrypting, 500
  grids, 500
  image compression, 500
  Quartz Filter, 500
  renaming, 501
  watermarks, 501
PDF files
  annotation extraction, 421–422, 500
  creating from images, 501
  extracting text with a Folder Action plug-in, 250–254
  metadata, 500–501
  odd/even page extraction, 500
  page extraction, 422–423
  PostScript conversion, 245
  searches, 501
  text extraction, 419–422, 500
PDF keywords, searches, 9–16
PDF pages
  combining, 500
  rendering as images, 501
PDF to Images, PDF actions, 501
performance, automation enhancement, 3
Photo Booth application, iSight camera control, 369
photographers, automation tasks, 55–56
Photos actions, Automator, 65, 501–503
Photoshop Action Pack, Automator workflows, 434–435
pictures
  desktop, 490
  Locations variable, 509
Pictures folder, path representation, 509
`ping` command, running from a shell script, 231–232
Plain Text Editor, AppleScript Studio application, 258
`play` command, iTunes song playback, 310–312
Play DVD, Movie actions, 497
Play iPhoto Slideshow, Photos actions, 503
Play iTunes Playlist, Music actions, 499
Play Movies, Movie actions, 497
Play random iTunes track, sample script, 154–155
Playing a specific iTunes song project, 306–312
playlists
  adding songs, 313–315
  removing empty, 327–328
`playpause` command, pausing/playing iTunes songs, 323
plug-ins
  AppleScript Editor preferences, 120
  application sharing, 98
  Calendar, 99–100
  Finder, 98–99
  Folder Action, 250–254
  iCal Alarm, 99
  Image Capture, 101
  preferences, 118–120
  Print, 101
  saving a workflow as, 80–81
  Script, 101
PNG format, duplicating any image as, 245
podcasters, automation tasks, 55–56
POSIX file object, AppleScript, 461–462
POSIX path property, alias object, 452
PostScript
  font name display, 492
  PDF file conversion, 245
Preferences dialog box, AppleScript Editor, 118–120
presentations, small business task, 58
Preview, image editor, 339
Print Finder Items, Utilities actions, 506
Print Images, Photos actions, 503
printers, Print plugin, 101
printing, Finder items, 506
process simplification, automation reason, 45
production, small business task, 57
programming languages, AppleScript Editor preferences, 118

projects. *See also* scripts
Adding attachments to messages, 416–419
Adding audio files to an iPod, 332–334
Adding songs to a playlist, 312–315
Applying color changes to groups of images, 336–342
Asking for Text from the User, 376–379
Automating taking pictures with a digital camera, 368–369
Changing case of song names, 328–330
Combining Mail Messages, 413–416
Combining Text Files, 387–389
Connecting to a server, 293–294
Converting a basic workflow to accept any files, 276–279
Converting images, 352–353
Converting text to audio files, 330–332
Creating a basic workflow to process specific files, 273–275
Creating a Group Mailer, 406–408
Creating aliases for files and folders, 286–289
Creating thumbnails, 349–351
Cropping and resizing images, 342–349
Downloading Specific URLs, 423–425
Exporting images from iPhoto, 364–365
Extracting PDF Pages, 422–423
Extracting Text from PDFs, 419–422
Filtering Finder items, 289–293
Filtering iTunes songs, 316–318
Finding files and folders and renaming them, 280–284
Finding files and folders and trashing them, 284–286
Finding People with Birthdays, 403–405
Finding Specific Calendar Items, 409–410
Finding Specific Contacts in Address Book, 400–403
Finding Specific Images, 359–361
Flipping and rotating images, 354–359
Getting a Specific Character of Text, 382–383
Getting a specific paragraph of text, 383–387
Getting a Specific Word, 379–382
Getting folder contents, 294–296
Getting New Mail Messages, 410–413
Getting the Definition of a Word, 390–393
Importing images to iPhoto, 361–364
Opening files with the proper application, 297–299
Opening Text Files, 373–375
OpenSafari, 259–266
Pausing and Playing iTunes, 321–323
Playing a specific iTunes song, 306–312
Removing empty playlists, 327–328
Reviewing photos in PDF Contact Sheet, 365–368
Setting Information on iTunes songs, 323–327
Setting iTunes volume, 318–321
Setting Spotlight comments for files and folders, 299–302
Using BBEdit: Convert Spaces to Tabs, 395–396
Using BBEdit: Working with quotes, 393–395
Using BBEdit: Zapping Gremlins, 396–397
properties
alias object, 452
AppleScript objects, 129–130
application object, 453–454
Boolean object, 454
class object, 455
constant object, 456
date object, 457
integer object, 459
list object, 459–460
number object, 461
POSIX file, 461
real object, 462
record object, 464
script object, 465
special purpose variables, 153–154
text object, 465
`properties` command, Address Book associations, 402–403
`property` command, setting variable properties, 154
Public folder, path representation, 509
Public, Locations variable, 509

Q

Quartz Composition file, QuickTime movie conversion, 496
Quartz Composition filter, images, 502–503
Quartz Filter, PDF documents, 500

QuickTime movies. *See also* movies
audio captures, 497
color profiles, 496
DVD playback, 497
enabling/disabling tracks, 496
movie hints, 496
Quartz Composition file conversion, 496
screen captures, 497
video capture, 497
Quit All Applications, Utilities actions, 506
Quit Application, Utilities actions, 506
quoted form property, text object, 465
quotes, straightening in BBEdit, 393–395

R

Random identifier, Utilities variable, 511
Random number, Utilities variable, 91–92, 511
real object, AppleScript, 462–464
Record button
Automator Toolbar, 5, 62–63
manual event recording, 103–106
record object, AppleScript, 463–465
record, AppleScript data type, 110
recordings
manual events, 102–106
stopping, 104
records
key-value pairs, 151
value retrieval, 151–152
recursive subroutines, self-calling subroutine, 218–219
reference guides, AppleScript Editor preferences, 123
remainder (mod) operator, AppleScript, 472
reminders
automation use, 42
birthdays, 404–405
Remove Empty Playlists, Music actions, 499
Remove Font Files, Font actions, 492
Removing empty playlists project, 327–328
Rename Finder Items, Files & Folders action, 490
Rename PDF Documents, PDF actions, 501
Render PDF Pages as Images, 501
Render Quartz Compositions to Image Files, Photos actions, 503
`repeat` command, AppleScript, 25–26
`repeat` loop, combining text files, 388–389
`repeat` statement, AppleScript, 110, 475–477
`repeat loop`, repeating forever, 185
`repeat until` statement, AppleScript, 189–190, 476–477
`repeat while` loop, repeating while something is true, 190–191
`repeat with` loop, repeating with a list, 191–192
`repeat with` statement, defined start/stop values, 187–189
`repeat X times`, repeating a set number of times, 186
reports, small business task, 58
reserved keywords, AppleScript, 480–482
resources
online community, 432–434, 445–446
tools/downloads, 434–437, 446–448
training, 437–438, 448–449
Web-based tutorials, 429–432, 439–444
rest property, list object, 460
Results button
Automator, 76
debugging workflows, 10–11
Resume Capture, Movie actions, 497
Resume DVD Playback, Movie actions, 497
return on investment (ROI), small business automation, 59–60
Reveal Finder Items, Files & Folders action, 490
reverse property, list object, 460
Review Photos, Photos actions, 503
Reviewing photos in PDF Contact Sheet project, 365–368
`rotate` command, rotating images, 356–357
Rotate Images, Photos actions, 503
RSS feeds
article enclosures, 493
filtering articles, 493
link URL retrieval, 493
marking articles, 494
permalink retrieval, 493
text retrieval, 493
rules
applet creation, 228
automation, 48–49
Folder Action Scripts, 245
Run a shell script, applet, 231–232

Run AppleScript, Utilities actions, 506
Run button, Automator Toolbar, 62–63
Run Shell Script, Utilities actions, 506
Run Web Service, Internet actions, 494
Run Workflow, Utilities actions, 506
running property, application object, 454

S

Safari
- automation task types, 52
- browsing movies, 496
- new document opening, 494
- opening a Web page applet, 228–231
- opening a Web page script, 20–21
- OpenSafari project, 259–266
- RSS feed retrieval, 493
- Web page display, 493

Safari Suite, dictionaries, 138–139
sales, small business task, 57
`say` command, user input, 199–200
`scale` command, resizing images, 344–347
Scale Images, Photos actions, 503
screen captures, QuickTime, 497
screen savers, starting, 506
screenshots, taking, 506
Script Assistant, AppleScript Editor preferences, 119
Script Editor
- AppleScript editing, 18–19
- opening a folder, 18–19
- opening a Web page, 20–21

script launching, 40–41
Script menu, AppleScript, 121–122, 124
script object, AppleScript, 465
Scripting Additions folder, path representation, 509
Scripting Additions, Locations variable, 509
scripts. *See also* projects
- AppleScript command reference, 468
- AppleScript Editor preferences, 120–121
- AppleScript error numbers, 483
- Check if folder exists, 193–194
- Checking for e-mail validity, 173–175
- continuation character, 189
- Copy files between directories, 155–156
- Count items in a folder subroutine, 225–226
- Create a folder subroutine, 221–223
- e-mail address retrieval, 403
- Export e-mail addresses, 194–195
- Get file size, 233
- Get File size droplet, 234–237
- Inspect the Trash, 175–178
- iTunes volume setting, 320–321
- Locations variable, 509
- Move an item to the Trash subroutine, 223–224
- new e-mail message retrieval, 412–413
- Open a Web page in Safari, 228–231
- pausing/playing iTunes songs, 323
- Play random iTunes track, 154–155
- Run a shell script, 231–232
- saving to a file, 116–117
- Set the desktop background droplet, 237–239
- top-level script objects, 130–131
- updating to a Folder Action Script, 246–250
- View today's events in iCal, 156–157
- word definitions, 392–393

Scripts folder, path representation, 509
Scripts menu, enabling, 101
Search PDFs, PDF actions, 501
searches
- actions, 63–65
- Address Book birthday, 488
- Address Book contacts, 400–403
- Address Book items, 487
- AppleScript Editor Help menu, 123
- Automator variables, 66–67
- Calendar items, 409–410
- Command Finder, 258
- contact birthdays, 403–405
- dictionary information lookup, 141–142
- Finder items, 489
- Font Book items, 491
- group e-mail, 406–408
- iCal items, 486
- images, 359–361
- iPhoto items, 502
- iTunes items, 498
- Mail items, 495
- PDF files, 501
- PDF keyword, 9–16
- Spotlight, 506
- system administrator, 54

security, small business task, 57
Select Fonts in Font Book, Font actions, 492
Send Birthday Greetings, Mail actions, 495
Send Outgoing Messages, Mail actions, 495
servers
  connections, 293–294, 489
  retrieving, 489
  selecting, 489
`set` command
  AppleScript, 23–26
  variable creation method, 144
Set Application for Files, Files & Folders action, 490
Set Computer Volume, Utilities actions, 506
Set Contents of TextEdit action, adding audio files to an iPod, 333–334
Set Contents of TextEdit Document, Text actions, 504
Set Folder Views, Files & Folders action, 490
Set info of iTunes Songs, Music actions, 499
Set iTunes Equalizer, Music actions, 499
Set iTunes Volume, Music actions, 499
Set Options of iTunes Songs, Music actions, 499
Set PDF to Metadata, PDF actions, 501
Set Spotlight Comments for Finder Items, Files & Folders action, 13, 490
Set the desktop background, droplet, 237–239
Set the Desktop Picture, Files & Folders action, 490
Set Value of Variable, Utilities actions, 506
`set volume` command, audio alerts, 198
Setting Information on iTunes songs project, 323–327
Setting iTunes volume project, 318–321
Setting Spotlight comments for files and folders project, 299–302
Shared folder, path representation, 509
Shared, Locations variable, 509
Shell script, Utilities variable, 91–92, 511
shell scripts
  combining text files, 389
  file/folder aliases, 288–289
  filtering Finder items, 293
  finding/trashing files and folders, 286
  processing specific files, 275
  running, 506
  running from an applet, 231–232
  software developers, 53
  system administrator, 55
Short user name, User variable, 91, 511
Show Growl Notification, Utilities actions, 506
Simple Shell, AppleScript Studio application, 258
single-line comments, double dash (--) characters, 114–115
Sites folder, path representation, 509
Sites, Locations variable, 509
slideshows, iPhoto, 503
small business, automation task types, 56–60
Smart Folder, creating from Spotlight Comments, 15–16
Smart Groups
  Automator, 67–68
  group e-mail, 406–408
  versus groups, 68
smart quotes, straightening, 394
software developers, automation tasks, 53–54
`some item` command, list item retrieval, 150
songs
  adding to a playlist, 312–315
  changing name case, 328–330
  choosing from a catalog, 497
  iPod addition, 497
  iTunes filtering, 316–318
  iTunes information display, 323–327, 499
  iTunes playback, 306–312
  iTunes playlist addition, 497
  iTunes retrieval, 499
  name case changing, 497
  pausing/playing in iTunes, 321–323
  removing empty playlists, 327–328
Sort Finder Items, Files & Folders action, 490
sorts, Finder items, 490
spaces, tab conversion, 395–396
Speak Text
  Text actions, 504
  Utilities actions, 506
Spotlight
  displaying comments, 245
  file/folder comments, 299–302
  Finder item comments, 490
  image information comments, 245
  searches, 506
  Utilities actions, 506

Spotlight action, PDF keyword search, 9–16
Spotlight Comments
file metadata, 12–13
Smart Folder creation, 15–16
user input, 15–16
spreadsheets, small business task, 58
Standard Suite
AppleScript command reference, 468
dictionary view, 138–140
Image Events, 339
Start Capture, Movie actions, 497
Start iTunes Playing, Music actions, 499
Start iTunes Visuals, Music actions, 499
Start Screen Saver, Utilities actions, 506
statements
AppleScript, 172–173, 474–478
considering characteristics, 474
`if`, 474–475
ignoring characteristics, 474
`on open`, 234
`repeat`, 475–477
`repeat until`, 189–190, 476–477
`repeat with`, 187–189
`tell`, 477–478
`try`, 478
Stop button, Automator Toolbar, 62–63
Stop Capture, Movie actions, 497
Stop DVD Playback, Movie actions, 497
Stop iTunes Visuals, Music actions, 499
Storage, Text & Data variable, 510
`straighten quotes` command, straightening curly quotes, 395
Straighten Quotes function, straightening curly quotes, 394
strings, AppleScript command reference, 469
Structured Query Language (SQL), CSV data conversion, 58
subroutines
arguments, 215–216
benefits, 214–215
calls, 216–218
conditional tests, 218
Count items in a folder, 225–226
Create a folder, 221–223
format elements, 215
libraries, 219–221
loops, 218
Move an item to the Trash, 223–224
numbers as values, 216–217
parentheses (and) characters, 216
recursive, 218–219
reusable code bits, 213–214
variables, 218
subtraction (-) operator, AppleScript, 164, 472
suite pane, dictionaries, 136
suite view, dictionaries, 138–139
system administrators, automation tasks, 54–55
System Events application, word definitions, 391–392
system maintenance, 43
system monitoring, system administrator, 55
System Preferences
enabling Accessibility options, 102–103
system voices, 199–200
System Profile
information display, 506
Utilities actions, 506
System variables, Automator, 8, 88–89, 509–510
system voices, `say` command, 199–200

T

Table view, workflow results display, 10–11
tabs
AppleScript Editor preferences, 119
space conversion, 395–396
Take Picture, Photos actions, 503
Take Screenshot, Utilities actions, 506
Take Video Snapshot, Photos actions, 503
tasks
accounting, 57
Address Book, 50–51
automation advantages, 2–3
automation types, 48–49
databases, 58
e-mails, 58
facilities, 57
finance, 57
Finder, 49–50

human resources (HR), 57
iCal, 50–51
iPhoto, 51
iTunes, 52
Mail, 49–50
marketing, 57
memos, 58
photographers, 55–56
podcasters, 55–56
presentations, 58
production, 57
reports, 58
Safari, 52
sales, 57
security, 57
small business, 56–60
software developers, 53–54
spreadsheets, 58
system administrators, 54–55
templates, 58
videographers, 55–56
visual designers, 55–56
TECSoft, training, 438
`tell` statement, AppleScript, 477–478
`tell application` command, AppleScript, 18–26
tell construct, script target identifier, 110
templates
Automator, 81–82
small business task, 58
test scripts, software developers, 53
text
AppleScript data type, 110
audio file conversion, 330–332, 499, 504
extracting from PDF files, 419–422
PDF file extraction, 500
retrieving, 503–504
RSS feed retrieval, 493
speaking specified text, 504, 506
Text & Data variable, 510
Web page retrieval, 494
Text & Data variables, Automator, 8, 89–90, 510–511
Text actions, Automator, 65, 503–504
text alignments, AppleScript Editor preferences, 122
text files
combining, 387–389, 503
creating, 504
dynamic naming, 88–89
gremlin removal, 396–397
opening, 373–375
specific character retrieval, 382–383
specific paragraph retrieval, 383–387
specific word retrieval, 379–381
word range retrieval, 381–382
text input, user input capture, 203–204
text object, AppleScript, 465–466
Text Suite view, dictionaries, 138
Text Suite, Image Events, 339
Text to Audio File
Music actions, 499
Text actions, 504
Text to Speech
speaking specified text, 504, 506
system voices, 199–200
TextEdit
asking for text from the user, 378–379
document content retrieval, 504
document content setting, 504
document creation, 504
document font display, 492
`thru` command, list item retrieval, 150
thumbnails
creating, 349–351
image files, 502
TIFF format, duplicating any image as, 245
time property, date object, 457
time saver, automation reason, 43–44
time string property, date object, 457
time, current time display, 507
to do items
iCal creation, 487
Mail creation, 495
Today's Date
Date & Time variable, 87–89, 508
displaying, 508
toolbar
AppleScript Editor, 113–114, 121–122
Automator interface, 4–5, 62–63

top-level script objects, AppleScript, 130–131
training classes
  Automated Workflows, 438, 448–449
  TECSoft, 438
Trash
  item listing, 175–178
  moving Finder items, 506
  moving items to, 490
  moving items to subroutine, 223–224
`try` statement, AppleScript, 478
tutorials
  AppleScript Editor preferences, 123
  Automator.us, 430–432
  MacResearch AppleScript, 439–441
  MacScripter, 441–442
  MacTech, 442–443
  videos, 444
  VTC Mac OS X Automator, 429–430
two-value conditional test, `if-else`, 181–183
Type Definitions Suite, Image Events, 339

U

unary operators, AppleScript, 159
Unicode characters
  AppleScript character set, 109
  text object, 465–466
uniform resource locators (URLs)
  downloading, 95–96, 493
  filtering, 493
  image retrieval, 493
  retrieving, 95–96
  RSS feed retrieval, 493
Universal Access, System Preferences, 102–103
Update iPod, Music actions, 499
user input
  AppleScript folder selection, 26–27
  application selections, 210–211
  asking for text from, 376–379
  `beep` command, 198
  button arrays, 204–206
  capturing text input, 203–204
  communicating choices, 197–198
  `display dialog` command, 200–204
  file selections, 207–208
  filtering iTunes songs, 317–318
  folder selections, 206–207
  label color selection, 15
  list selections, 208–210
  Options button, 76
  saving files, 207–208
  `say` command, 199–200
  Spotlight Comments, 15–16
  waiting for, 506
  workflow prompt, 14
user interaction, AppleScript command reference, 469
user interface
  AppleScript Editor, 113–114
  Automator, 4–6, 62–69
User Library folder, path representation, 509
User Library, Locations variable, 509
User name, User variable, 511
User Preferences folder, path representation, 509
User Preferences, Locations variable, 509
User variables, Automator, 8, 90, 511
Users folder, path representation, 509
Users, Locations variable, 509
`using` clause, `say` command, 199–200
Using BBEdit: Convert Spaces to Tabs project, 395–396
Using BBEdit: Working with quotes project, 393–395
Using BBEdit: Zapping Gremlins project, 396–397
Utilities actions, Automator, 505–506
Utilities folder, path representation, 509
Utilities, Locations variable, 509
Utilities variables, Automator, 8, 91–92, 511–512

V

Validate Font Files, Font actions, 492
values
  literal expressions, 170–172
  retrieving from a record, 152
  subroutine numbers, 216–217
  variable display, 146
Variable list, hiding/display, 69
Variable Options dialog box, adding variables to a workflow, 93
variable reference, Automator, 507–512
Variables button, built-in variable selections, 8

Variables library
Automator interface element, 65–67
variable selections, 66
variables
adding to a workflow, 93
AppleScript reserved keywords, 480–482
creation methods, 144–145
data containers, 84
Date & Time, 87–88
FavoriteColor, 90
`foldername`, 23–25
global, 152
information display, 69
IP address, 88
list display, 8
list item retrieval, 146–151
local scope, 152
Locations, 88
`myfiles`, 25–27
naming conventions, 144–145
New Path, 88
New Text, 89
non case-sensitive names, 144
Operating system version, 88–89
properties, 153–154
record value retrieval, 151–152
Shell Script, 91–92
Short user name variable, 91
subroutines, 218
System, 88–89
Text & Data, 89–90
Today's date, 87–88
User, 90
Utilities, 91–92
value retrieval, 505
value returns, 146
value setting, 506
value storage, 143
workflow activation, 93–94
vCards, exporting from Address Book, 487
version property, application object, 454
video snapshots, digital cameras, 503
videographers, automation tasks, 55–56
videos
AppleScript tutorials, 444
QuickTime capture, 497
View menu, AppleScript, 121
View Results, Utilities actions, 506
View today's events in iCal, sample script, 156–157
visual designers, automation tasks, 55–56
visuals, iTunes, 499
`voices, say` command, 199–200
volume
iTunes, 318–321, 499
setting, 506

W

Wait for User Action, Utilities actions, 506
Waldie, Ben, Automated Workflows, 436–438
watch folders, software developers, 53
Watch Me Do, Utilities actions, 105, 506
Watermark PDF Documents, PDF actions, 501
Web pages
displaying, 493
downloading specific URLs, 423–425
image URL retrieval, 493
link URL retrieval, 493
opening from an applet, 228–231
opening script, 20–21
OpenSafari project, 259–266
Safari display, 493
text retrieval, 494
Web service, running, 494
Web sites
action downloads, 6
Apple Developer Connection, 126
AppleScript Language Guide, 125, 451–452
Automated Workflows, 436–437, 448
Automator actions, 72
Automator World, 432–433
Automator.us, 430–432
Automator-dev group, 434
CNET, 447–448
Delicious, 438
Doug's AppleScripts for iTunes, 446–447
Flickr, 95

Web sites *(continued)*
Google AppleScript tutorials, 444
MacResearch tutorials, 439–441
MacScripter, 441–442, 445
MacTech, 442
OttoMate, 435
Photoshop Action Pack, 434–435
TECSoft, 438
VTC Mac OS X Automator tutorial, 429–430
weekday property, date object, 457
weekday, current weekday display, 507
Window menu, AppleScript, 122
windowpanes, AppleScript Editor, 113–114
`with icon` clause, `display dialog` command, 202
`with prompt` clause, `choose folder` command, 206–207
words
definition retrieval, 390–393, 504
specific word retrieval, 379–381
word range retrieval, 381–382
`words` command, word range retrieval, 381–382
Workflow log, hiding/displaying, 69
Workflow pane, Automator interface element, 66–67
workflows
action passing process, 74–76
adding/removing media files, 63
application, saving as, 79
Automator, 5–6
Automator actions, 6–8
Automator documents, 70
Automator variables, 8–9
build process, 70
collapsing/expanding actions, 77
creating from Automator, 9–13
creation conditions, 71
debugging, 10–11
enabling/disabling actions, 77
Finder selections, 70–71
Folder Action plug-ins, 250–254
hiding/displaying libraries, 63
iCal Alarm plugin, 99–100
label display, 12, 15
loop delay, 97
loop insertion, 506
manually recording, 63
pausing, 506
Photoshop Action Pack, 434–435
playing, 63
plugin, 80–81
project types, 72
results display, 506
running, 506
save options, 17
saving as a workflow, 78–79
Script plugin, 101
Shell Script variable, 91–92
Spotlight action search, 9–16
Spotlight Comments, 12–13, 15
startup methods, 70
stopping, 63
stopping manual recordings, 104
templates, 81–82
user input prompt, 14
variable activation, 93–94
variable addition, 93

X

Xcode
AppleScript Studio access, 256–257
OpenSafari project, 259–266

Y

year, current year display, 508
year property, date object, 457

Z

Zap Gremlins action, gremlin removal, 397
ZIP archive, creating, 489

If you use a Mac, you'll be amazed at what you can automate

Save time when using your Mac with Apple's powerful AppleScript and Automator automation tools, which come standard with Mac OS X. This comprehensive guide shows you how to tap both these handy features to streamline your workflows. Learn how to launch your e-mail every day at a set time, or extract data for analysis on a regular schedule, or dozens of other useful ways to automate routine computer tasks, avoid errors, and save yourself time.

- Explore the most current Automator tricks for Snow Leopard™
- Learn essential Automator concepts, such as actions, workflows, plugins, and Apple events
- Create a droplet and simple scripts while mastering AppleScript's syntax, variables, objects, and properties
- Automate routines you do all the time, such as opening iTunes® to a random track or making slideshows
- Manipulate files and e-mails, convert PDFs to images, find Calendar items, and discover other functions to automate

Companion Web Site

Visit www.wiley.com/go/applescriptbible for the author's code samples from the book.

Thomas Myer

is a consultant, technical author, and speaker. He owns Triple Dog Dare Media and specializes in many aspects of Linux- and Mac-based development, including Web content management systems, blogs, wikis, dashboard widgets, iPhone applications, UNIX systems programming, and Applescripting. He is the author of five books, including *Professional CodeIgniter* and *Mac OS X UNIX Toolbox*.

www.wiley.com/compbooks

WILEY
wiley.com

Shelving Category:
COMPUTERS / Operating Systems / Macintosh

Reader Level:
Beginning to Advanced

$44.99 USA
$53.99 Canada

ISBN 978-0-470-52586-9
54499
9 780470 525869