Mastering Xcode
SECOND EDITION
Covers Xcode 4
DEVELOP AND DESIGN
Maurice Kelly
Joshua Nozzi
To my wife, Fiona, and our beautiful daughters, Aoibhínn and Caoimhe—
I thank you for your support and patience while I’ve been distracted by this book.

To my parents—thank you for buying our first family computer and
setting me on the course that led me to do what I love.

— Maurice Kelly

Thanks to my peers, my friends, my family, and my partner
for their enthusiastic support. Special thanks to my coauthor:
May our Caoimhes live long, happy lives.

— Joshua Nozzi
ABOUT THE AUTHOR

Maurice Kelly has been a software engineer since leaving university in 2001. After spending a decade working on carrier-grade server software in C, C++, and Java, he decided to take a career departure and switched to developing Mac and iOS software. As well as being an eager consumer of all things tech, he has a passion for consuming and creating music. He lives with his wife and children just outside Dromara, a small village in the small country of Northern Ireland.

ACKNOWLEDGMENTS

I would like to thank Josh for giving me the opportunity to work on this project in ever-increasing capacities. The first edition of this book gave me inspiration to change my career, and it has been an honor to work with him in producing the second edition. I would also like to thank my employers, Andrew Gough and Andrew Cuthbert at GCD Technologies (www.gcdtech.com), for allowing me to take on this side-project and for introducing me to Mark Goody—a newfound friend, an amazing developer, and an excellent technical editor. I would like to extend many thanks to Robyn Thomas for her encouragement and for guiding this ship through patches of both stormy water and dead calm!

While Xcode is the day-to-day tool of my trade, there were a number of tools without which this book could not have been completed:

- Sublime Text 2 (www.sublimetext.com/2). Combined with Brett Terpstra’s Markdown-Editing package, this makes a much better writing environment than Xcode’s editor!
- Marked (http://markedapp.com). For previewing Markdown output, there is no better application than Marked.
- ScreenFloat (www.screenfloatapp.com). Capturing and managing the quantity of screenshots required for this book needed an app like ScreenFloat.
ABOUT THE AUTHOR

Joshua Nozzi is a self-taught technologist who has been developing software for the Mac platform since Mac OS X 10.0 debuted. He's been using Xcode since version 1 for publishing, increasing productivity, and building scientific research applications. He's haunted several developer communities over the years, offering help and snark in equal measures. He loves to teach technology to others. Josh lives with his partner in Southern Virginia, where he toils in obscurity, usually in sweatpants and little else.

ACKNOWLEDGMENTS

I wish to thank the following people, whose work I used while writing this book.

Cyril Godefroy: Cyril's masterfully broken code examples demonstrated some nice highlights of the Clang static analyzer. You can find them at http://xcodebook.com/cgodefroy.

Colin Wheeler: Colin's Xcode shortcut cheat sheet has saved me loads of tedium on many projects. You can find the original, downloadable version that Colin maintains at http://xcodebook.com/cwheeler.
CONTENTS

Introduction .......................................................... x
Welcome to Xcode .................................................. xii

PART I  Getting Started

CHAPTER 1  INSTALLING XCODE ............................................ 2
  Downloading .......................................................... 4
  Getting With the Program ......................................... 5
  Even More Stuff ...................................................... 5
  Wrapping Up .......................................................... 5

CHAPTER 2  EXPLORING THE ENVIRONMENT ....................... 6
  You Get One Window ................................................ 8
  Creating a Project .................................................... 8
  The Workspace Window ............................................ 10
  The Navigator Area ................................................... 11
  The Jump Bar .......................................................... 17
  The Editor Area ....................................................... 18
  The Utility Area ....................................................... 22
  The Debug Area ....................................................... 23
  The Activity Viewer .................................................. 23
  Tabbed Coding ........................................................ 24
  The Organizer Window .............................................. 25
  Wrapping Up .......................................................... 25

CHAPTER 3  GETTING HELP ............................................ 26
  The Help Menu ....................................................... 28
  The Organizer's Documentation Tab ................................ 29
  The Source Editor .................................................... 30
  Community Help and Feedback ................................... 32
  Wrapping Up .......................................................... 32
# PART II  Building Applications

## CHAPTER 4  Setting up Your Workspace

- Workspaces Defined ........................................... 38
- When to Use a Workspace .................................... 39
- Creating the Lighting Suite Workspace .................. 40
- Wrapping Up ..................................................... 41

## CHAPTER 5  Adding Resources and Code

- Working with Files ............................................. 44
- Adding Files to Lamp .......................................... 48
- Working with the Source Editor ............................ 48
- Wrapping Up ..................................................... 55

## CHAPTER 6  Version Control with Xcode Snapshots

- Xcode Snapshots ................................................. 58
- Wrapping Up ..................................................... 61

## CHAPTER 7  Building User Interfaces

- Understanding Nibs ............................................. 64
- Getting Familiar with Interface Builder .................. 66
- Adding User Interface Elements ............................ 73
- Storyboards ....................................................... 81
- Wrapping Up ..................................................... 87

## CHAPTER 8  Creating Core Data Models

- Introducing Core Data ........................................ 90
- Using the Data Model Editor ................................. 92
- Creating a Basic Data Model for Lamp .................. 94
- Wrapping Up ..................................................... 99

## CHAPTER 9  Debugging Your Applications

- Interactive Debugging ....................................... 102
- Debugging Flashlight ........................................ 107
- Static Analysis .................................................. 109
- Wrapping Up ..................................................... 112

## CHAPTER 10  Deploying

- Archiving ........................................................ 116
- Validating Your Application ................................ 118
- Distribution Channels ....................................... 120
- Alternatives to Archiving ................................... 129
- Wrapping Up ..................................................... 130
PART III Further Exploration

CHAPTER 11 USING OLDER PROJECTS IN XCODE 4 .............................. 134
  Project Modernization Methods ........................................... 136
  Code Modernization Methods .............................................. 140
  Wrapping Up ................................................................. 145

CHAPTER 12 ADVANCED EDITING .............................................. 146
  Renaming Symbols ........................................................... 148
  Refactoring ................................................................. 149
  Organizing with Macros .................................................... 153
  Changing Editor Key Bindings ............................................. 154
  Adjusting Project Settings .................................................. 156
  Using the Search Navigator ................................................ 157
  Searching Within Files ..................................................... 162
  Wrapping Up ................................................................. 163

CHAPTER 13 THE BUILD SYSTEM .............................................. 164
  An Overview ................................................................. 166
  Working with Targets ........................................................ 168
  Working with Schemes ...................................................... 184
  Entitlements and Sandboxing .............................................. 196
  Wrapping Up ................................................................. 198

CHAPTER 14 WORKING WITH FRAMEWORKS .............................. 200
  What Are Libraries, Frameworks, and Bundles? ...................... 202
  Using Existing Libraries and Frameworks ................................ 204
  Creating a Framework ....................................................... 210
  Wrapping Up ................................................................. 215

CHAPTER 15 IMPROVING CODE QUALITY ................................ 216
  Debugging ................................................................. 218
  Instruments ................................................................. 224
  Unit Testing ................................................................. 236
  Wrapping Up ................................................................. 247
INTRODUCTION

This book is an intermediate-level introduction to Xcode 4, Apple’s integrated development environment. It assumes you have some development experience and are familiar with Objective-C and the Cocoa and Cocoa Touch APIs. It won’t teach you how to write code or much at all about the frameworks needed to develop OS X and iOS applications. There are other books for that. This one is strictly focused on how to use Xcode itself, whatever your development endeavors.

Of course, since Xcode is most often used with the Cocoa and Cocoa Touch APIs and Objective-C, there are basic introductions to concepts and a few code samples sprinkled here and there to illustrate various points. In these cases, you will be pointed to the documentation that Apple provides (to save you some trouble looking it up), but remember that the focus of this book is really on getting the most out of the tools and not necessarily on what you’ll be building with them.

A MOVING TARGET

When the first edition of this book was released, Xcode 4.0 had just become Xcode 4.1. It was hard for anyone to predict how rapidly Apple would iterate, but in a short space of time there have been five major versions of Xcode 4 released. Each version brings enhancements, fixes, and new tools—and headaches to the authors of this book.

So just as in the introduction to the first edition, we will once again make our excuses up front and say that this book was current when we wrote it, and may or may not be by the time you read it. We hope that, however Apple chooses to change Xcode, our guidance is still relevant for the foreseeable future and that this book will be a trusty companion for up-and-coming developers for some time to come.
WHAT YOU WILL LEARN

This book is divided into three major parts and includes appendixes.

PART I: GETTING STARTED
In very short order, you'll install Xcode and take a tour around its interface's major points of interest, and you'll learn where to look for answers when you need help.

PART II: BUILDING APPLICATIONS
Next, you'll dive into the process of building OS X and iOS applications. Through the development of a pair of basic apps, you will learn how to create projects and workspaces; manage resources and code; build and edit user interfaces; and debug and deploy your work.

PART III: FURTHER EXPLORATION
Then you'll dive a little deeper and find out how to bring older Xcode projects into the modern era, tackle advanced editing and refactoring, and unravel the complexity of Xcode's build system. You'll work with libraries and frameworks, and you'll improve the quality of your work using a combination of profiling, analysis, advanced debugging, and unit testing. You will investigate the extension possibilities offered by Xcode scripting support and command-line interfaces, and you'll wrap up with an overview of Xcode's integrated source code management support.

APPENDIXES
Appendix A helps you manage your iOS devices. Appendix B shows you how to manage Xcode documentation updates. Appendix C provides you with Apple and third-party resources for additional information.
WELCOME TO XCODE

Whether you are a complete newcomer or a seasoned programmer, Xcode can be an intimidating environment for a developer getting involved in Apple development for the first time. Under its shiny, easy-to-use interface, a lot of power lurks. Xcode 4 lets you write and manage your code, design and build user interfaces, analyze and debug your apps, and more.

INTERFACE BUILDER
Build and edit rich user interfaces with Interface Builder. Drag and drop outlets and actions directly into your code using the Assistant editor.

CLANG STATIC ANALYZER
Find subtle errors in your programs with the Clang static analyzer. Follow the blue arrows through your code as the problem is broken down step by step.
INSTRUMENTS
Trace and profile your code with Instruments. Follow your application’s activity through time to find and analyze performance problems and more.

SOURCE CODE MANAGEMENT
Manage your source code with the integrated source code management features. Branch, merge, pull, push, and resolve conflicts—all from within Xcode.
This page intentionally left blank
CHAPTER 4

Setting Up Your Workspace
In Part I, the Tour.app project was just that—an Xcode project. The collection of files and folders in an Xcode project are bound by an `.xcodeproj` file that contains all the project-wide settings (such as a description of your schemes and targets). So far, the word “workspace” has been used as a general description of the project and the window that contains it all. A true workspace in Xcode, however, is a container that encompasses multiple projects that share common resources.

From this point forward, we’re going to need something a bit more complex than a single project to demonstrate Xcode’s organizational capabilities. We’re going to create an application suite consisting of an OS X application, an iOS application, and a shared framework that encapsulates all the common components.

For this book’s demonstration, we’ll create our own version of one of the most innovative and popular uses of a bright white screen—the comical “flashlight” app. The suite will be composed of Flashlight.app (for iOS devices) and Lamp.app (for OS X). We’ll call the workspace “Lighting Suite.”
WORKSPACES DEFINED

Xcode 4 introduced the idea of a workspace as a kind of project binder—a container for multiple projects. A project groups its related files and settings; a workspace binds multiple related projects. A workspace merely contains pointers to Xcode projects. Projects remain distinct in that you can remove them from a workspace without affecting the project’s content or settings. In other words, the project can still be opened and edited outside its workspace. Workspaces give you several advantages over projects that reference files and built products from other projects.

Projects contained within the same workspace share a common build location. This makes it possible for one project to use another’s built products. This one feature makes a world of difference for managing complex applications and application suites. It makes it far easier, for example, to include the built product of a common framework project into one or more of your application projects.

The automatic dependency detection that you’ll learn about in Chapter 13 extends to the workspace level as well. This means that including a product’s framework in the target of an application project within the same workspace usually requires no additional work for Xcode to recognize the dependency. As with dependent targets within the same project, Xcode will see this dependency and build the framework before building the application. In other words, you don’t have to copy shared libraries into each project folder in which you intend to use the library.

NOTE: Xcode may not be able to detect complex dependencies automatically. In such cases, you’ll need to disable the Find Implicit Dependencies setting of the affected scheme and add and sort the interdependent targets manually.

Another benefit of a workspace is shared indexing. A project index is used primarily for features such as code completion (sometimes referred to as “code sense”). Xcode’s automatic code completion and refactoring facilities will take the symbols of all projects included in the workspace into account. This means code completion will automatically find your framework project’s symbols and make them available to you when you’re editing source files in the application project that uses the framework.

Still another benefit of workspaces pertains to schemes (Chapter 13). A standalone project might contain a primary scheme for building, testing, and profiling a primary product in addition to schemes for smaller, dependent targets (such as a Spotlight plug-in). In a workspace, you may only want to see the scheme for each project’s primary product. Using the Manage Schemes panel that you’ll explore in Chapter 13 you can specify whether the schemes for those smaller “sub-targets” are visible at the workspace level or only when the project is opened individually. This can help keep the list of schemes short and manageable, hiding unnecessary detail within the workspace.
WHEN TO USE A WORKSPACE

It's hopefully obvious that a workspace is useless without two or more projects. Less obvious but just as important is that a workspace doesn’t help with multiple unrelated projects. A workspace is only helpful for two or more projects that share each other’s code and resources. Let’s look at two real-world examples.

DISTINCT APPLICATIONS

Imagine Acme Corporation has a host of unrelated desktop (and even mobile) applications. Here, unrelated means a calculator application, a calendar application, and an address book application. Each of these applications has only one thing in common: They're products of Acme Corporation.

Being the property of the same business entity, the applications presumably use the same software registration system, company logo, contact information, and so on. They may even be able to share user data among them. This means each application would use the same code, the same resources, or both.

A change to the Person and Event classes, for example, might need to be updated in both the address book and calendar applications. While these classes may or may not be wrapped in a library or framework, it makes little sense to maintain two copies of Person and Event (one in each project). Here, a separate project that at least contains the common model-layer classes (and corresponding unit tests) makes sense. A separate framework project makes even better sense.

Since the applications are otherwise unrelated, each application might have its own workspace that includes the application project and the shared framework project. The benefit of such a setup is that changes made to a project that belongs to a workspace are available to that workspace. For example, if you have two applications (each in its own workspace) that share a common framework, changes to the framework from one app are automatically available to the other by virtue of including the framework in their respective workspaces.

APPLICATION SUITES

Imagine Acme Corporation’s desktop calendar application has gone where no calendar application has gone before. Against all odds, it has become a best seller, and users are clamoring for mobile versions for their various devices. Acme Corporation, in addition to its other products that share company-wide resources, now has a product that supports two platforms, shares company-wide resources, and has a device synchronization library to let users share calendar information between their devices and their desktop computers.

In this case it would make sense to have a workspace containing the two application projects (OS X and iOS), their sync library project, and the company-wide resource project.
Creating a workspace is easy. Choose File > New > Workspace from the main menu. A new workspace window will appear, with a Save As sheet prompting you for a location in which to save it (Figure 4.1). Again, I recommend the desktop for convenience. Create a folder called Lighting Suite to hold everything, and select it. Name the workspace Lighting Suite and click Save.

You’ll be presented with an eerily empty workspace window, whose workspace file lives inside the Lighting Suite folder you created. Now on to the projects. As mentioned, we’ll need two projects: the Mac app and the iOS app. The projects will live inside the Lighting Suite folder, along with the workspace, for good organization.

Adding Projects to the Workspace

Now we’ll create the individual projects.

Let’s do iOS first. This project will contain a universal app that will work on iPhones, iPod touches, and iPads. Choose File > New > Project. Pick the iOS Application category, and select the Single View Application template. Click Next. We’ll call this product Flashlight.

Accept the rest of the defaults (Use Storyboards, Use Automatic Reference Counting, Include Unit Tests), but choose Universal from the Devices menu. Click Next. You can choose whether or not you want to create a Git repository for the project, but make sure that you choose Lighting Suite from the Add To menu (Figure 4.2). Also make sure the main Lighting Suite folder is selected, and click Create.
Now for the desktop version of Flashlight, Lamp.app. Choose File > New > Project, and select the OS X Application category. Pick Cocoa Application, and click Next. We’ll call this product Lamp. This time we want to select Use Core Data, Use Automatic Reference Counting, and Include Unit Tests (we don’t want a Spotlight importer). Click Next. Choose Lighting Suite from the Add To menu, and make sure the main Lighting Suite folder is selected to ensure the projects are kept together. This time, an extra Group option appears when you’re asked to save. Make sure the group is set to the Lighting Suite workspace. Again, you can select “Create local git repository for this project” if you wish. You should end up with a workspace that looks like Figure 4.3, all contained within the Lighting Suite folder in Finder.

WRAPPING UP

The idea of Xcode’s new workspaces feature can seem intimidating at first. You’ve seen that it’s really quite straightforward. A workspace provides a way of tying related projects together to take advantage of Xcode’s (usually) intelligent dependency discovery. In the next chapter, you’ll add some code and resources to Lighting Suite’s projects.
A

action connections, explained 68
Activity viewer, features 23–24
Add Files sheet
  Add to Targets option 45
  Destination check box 44–45
  Folders option 45
  using 44–45
Analyze action, invoking 110
API Reference, accessing 28
APIs vs. SDKs 278
  See also build settings
App Store
  code signing apps 123
  distributing iOS apps in 122–123
Apple’s developer forums 32
application data
  downloading 315
  uploading 315
application sandboxing
  activating 197
  explained 196
apps. See also iOS apps; Mac apps
  installing 314
  managing 314–315
  uninstalling 314
ARC (Automatic Reference Counting)
  converting targets to 144
  previewing conversions 145
architectures 278–279
archive action, described 277
Archive configuration, editing for beta
  scheme 266
archive files
  dSYM folder 117
  as packages 117
archives
  action of Validate button 118–119
  annotating in Organizer 116
  creating 115
  displaying with action buttons 118
dSYM files 116
  finding 116–117
  finding Lamp.app 118
  storing 117
archiving
  alternatives to 129–130
  build environment 130
  on release builds 257–259
arrays, subscript notation for 142–143
Assistant panes
  adding 19–20
  removing 19–20
Assistant tool
  changing behavior modes 21
  controls 21
  dragging connections into 79
  features 19
  layout options 20
  Manual mode 21
  opening files in 19
attributes, adding to data model 93–94
automatic snapshot, creating 137

B

Beta Release configuration, using 267
BETABUILD macro, defining 264–267
BetaBuilder app, using with iOS apps 124
blocks of code 263
bookmarks, storing in framework project 212
Bookmarks framework
  downloading 208
  embedding 209
  linking against 209
  using in code 209–210
Bookmarks mode, using in Organizer 30
Bookmarks project, creating 210–211
branches
  creating 296–297, 303–304
  merging locally 304
  switching 296–297
breakpoint editor
  Action tool 220
  Condition field 219
  Enable/Disable check box 219
  Ignore directive 220
  Options 220
Breakpoint navigator
  Edit Breakpoint option 105
  features 15
  setting exceptions 105
  setting symbolic 105
  Share Breakpoint option 105
  using 105
breakpoints. See also debugging
  customizing 219–220
  enabling 105
  managing in Source editor 105
pausing at 108
use of 104
watchpoints as 218–219
browsing history 298–299
build actions
archive 277
build 277
clean 277
install 277
installsrc 277
test 277
build environment
macros in 264–267
manipulating for iOS apps 130
build phases 167
build rules 167
build settings 166
See also APIs vs. SDKs
build steps, viewing failures of 273
build system. See also schemes; targets
configurations 167
entitlements 196–198
run destinations 167
sandboxing 196–198
schemes 166
targets 166
build time vs. runtime 268
builds, triggering 9
bundles, loadable 203–204

call stack viewing in Debug bar 103
Clang static analyzer, using 110–112
CI (continuous integration) 274
Clang compiler. See also pragma directives
ignoring warnings 262
removing warnings 263
warnings 262
classes. See subclasses
clean action, described 277
Clean command, using 277
Cocoa applications. See also Core Data
memory leaks 111
nibs 64–65
Cocoa Auto Layout Guide, accessing 77
Cocoa Dev Central, described 324
Cocoa Fundamentals Guide 63
CocoaDev Wiki, described 324
code
adding automatically 50–54
focusing on 49–50
folding 50
string creation 141–142
updating 141–144
code blocks, excluding at build time 263
code completion, using 50–52
code focus ribbon 10
See also tabbed coding
code modernization, updating tools 140–141
code quality. See debugging; Instruments
application; unit testing
code signing 197
ad-hoc distribution of iOS apps 124
automatic identities 122
identity for Mac App Store 127
iOS apps for App Store 123
requirement 121
Code Snippet library
accessing 52
using 53–54
Command key. See keyboard shortcuts
command line
architectures 278–279
build options 277
building from 274–279
CI (continuous integration) 274
projects 276
schemes 275–276
SDKs 278
targets 276
workspaces 275–276
xcodebuild 274–276
command line tools
accessing 280–281
failing build steps 273
log viewer 272
xcrun tool 280
Command Line Tools package, installing 281
command sets, managing 155–156
command-line interface. See debugger console commands
executing in terminal window 273
getting help with 223
community help, Apple’s developer forums 32
Compile step detail 273
compile time vs. runtime 268
compiling projects 275
conditionals
excluding blocks of code 263
using in preprocessor 263–264
configurations
defining for targets 169–170
described 167
connections
dragging into Assistant 79
making 78–81
connections window 68
console logs, accessing for devices 316
See also debugger console
constraints
  adding 77–78
  adjusting properties of 76
  using 77
continue-to-here button, using 106
Control key. See keyboard shortcuts
Convert to Modern Objective-C Syntax 152
Convert to Objective-C ARC 152
converting projects 143–145
copying items into project folders 208
Core Data. See also Cocoa applications; data model
  entities 91, 98–99
  managed object contexts 91
  MOMs (managed object models) 90
  persistent stores 91
CPP (C preprocessor) 153
  See also preprocessor; scripting example
Create Superclass feature, using 150

D
data, managing 314–315
data model. See also Core Data
  attribute 94
  building 94–95
  clicking On/Off button 94–95
  Event entity 94
  generating subclasses 96–99
  planning 94
  purpose of 90
  using 94
Data Model editor
  attributes 93
  deleting items 92
  editor area 92
  graph style 93
  inspector 93
  interface 91
  jump bar 92
  model graph mode 93
  MOM (managed object model) in 91
  outline 92
  relationships 93
  table style 93
Debug area
  customizing 23
  features 23
Debug bar
  execution controls 103
  locating 10
  Pause/Continue button 103
  Show/Hide button 103
  Step Into button 103
  Step Out button 103
  Step Over button 103
  Threads and Stacks navigator 103
  varying button functionality 103
debug logs, selecting 16
Debug navigator
  features 14–15
  using 107
debugger. See also Source editor
  attachment of 102
  breakpoints 104–105
  pausing 104
  Variables pane 104
debugger console. See also console logs
  command structure 222
  command-line interface 220
  dot notation 222
  extending LLDB with Python 223
  getting help with commands 223
  logging returned strings 221
  printing objects 221–222
  printing values 221–222
  program execution 223
  using dot notation 222
  working in 104
debugging. See also breakpoints; LLDB debugger;
  static analysis
  Flashlight app 107–109
  modifying variables 218–219
  observing variables 218–219
debugging symbols, getting copy of 116
deleting
  items in Data Model editor 92
  schemes 185
deployment. See also distribution channels
  alternatives to archiving 129–130
  archiving 116–118
  validating applications 118–120
developer forums, getting help from 32
Developer ID-signed apps 128
developer preview version, switching to 280
development certificates, creating 308–309
device logs
  removing 316
  reviewing 315–316
device screenshots
  comparing 312
  list 311
  taking 312
  using as default images 313
development
  console logs 316
  installing iOS on 311
dictionaries, subscript notation for 142–143
distribution channels. See also deployment code signing 121–122
iOS apps 122–126
Mac apps 126–129
provisioning profiles 120–121
documentation
  feedback about errors 32
  searching for selected text 31
  viewing information about 321
documentation preferences, setting 320–321
Documentation section, accessing 29
documentation sets
  adding third-party 321
  information panel 321
documentation updates
  disabling automatic updating 321
  forcing update checks 321
dot notation, using in debugger console 222
dSYM files, explained 116
dSYM folder, locating 117
duplicating
  Release configuration 265
  schemes 186
dynamic libraries 202

E
Edit All in Scope command, using 148
editing. See also Source editor
  adjusting project settings 156–157
  organizing with macros 153–154
  refactoring 149–153
  renaming symbols 148
  searching within files 162
  using Search navigator 157–161
Editor area
  Assistant 19
  changing layout behavior 20
  features 18
  Source editor 18
editor key bindings. See Key Bindings preferences panel
Encapsulate tool, using 151
enterprise distribution 125–126
entities
  Event for Lamp 94
  explained 91
  inverse of relationships 91
  planning 98–99
  relationships 91
entitlements
  activating 197
  Apps controls 198
  file system 198

Hardware controls 198
iCloud settings 198
Network check boxes 197–198
setting 197–198
Entitlements controls 196
Entitlements option, using in Project editor 171
environment variables
  accessing 254
  for post-action scripts 251
  for pre-action scripts 251
  using with iOS apps 130
errors. See issues
Event entity, creating 94
exporting
  Mac app creating 94
files
  Add Files sheet 44–45
  adding to Lamp project 48
  choosing subclasses 46
  creating 46
  dragging and dropping 45
  opening in Assistant 19
  removing from projects 47
  searching within 162
  sharing between targets 183
Find options in Search navigator
  Find In 159
  find scopes 159–160
  Help Books 160
  Hits Must 159
  Match Case 159
  Style 159
finding
  phrases 162
  quickly 162
  schemes 184
  system-defined macros 261
  system-defined symbols 261
  words 162
Flashlight app
  background color 109
  debugging 107–109
  inspecting data 108
  iOS view controller 107–108
  pausing at breakpoint 108
focused code 10
focusing on code 49–50
folding code 50
framework project
adding code to 212–213
code visibility 214
configuring headers 213–215
Copy Headers phase 214
creating 210–211
file structure 215
installation directory 214
storing bookmarks 212
target 211
test application targets 215
workspace 211
frameworks. See also system framework
element 212–213
example
adding to projects 208–210
Bookmarks project 210–211
expanding in Project navigator 210
explained 203
SenTestingKit 238–239
third-party 208–210
Full Screen button, accessing 74
functions, using macros as 261

G
Gatekeeper 126
GC (garbage collection), use of 144
GCC compiler 140
GDB debugger 140
Git. See also version control
comparing to SVN (Subversion) 284
hosted services 300–303
repositories 286–289
reset options 300
Git branches
creating 296–297
switching 296–297
Git repository
merging changes 295–296
pulling changes 295–296
pushing changes 295
updating changes 295–296
Google.com, loading into web view 206–207
groups
identifying 11
nesting 11

H
help
getting from community websites 32
getting in Utility area 30–31
Help Books, accessing 160
help command, using with console 223
Help menu
displaying 28
User Guide 28
history
browsing 298–299
comparing 298–299
HockeyApp service, using with iOS apps 124
hosted Git services
adding projects 300–303
creating repositories 301–302
feature branches 303–305
pull requests 304–305
synchronizing repositories 303

I
.icns file, contents of 119
importing
.ipa files into iTunes 124
provisioning profiles 120
schemes 186
Info.plist file, editing information in 172
Inspector pane 10
Inspector selector bar 10
install action, described 277
installing
apps 314
iOS on devices 311
installsrc action, described 277
Instruments application
Attach to Process menu 226
call tree 234
Call Tree segment 235
CPU strategy 229
Detail view 231–232
Extended Detail view 232–233
Inspection Range controls 227
instrument chooser pop-up 229
Instruments view 230
launching 224–226
Library control 228
list of templates 225
mutable array 233
percentages for time profiling 235
Record control 227
scrubber 227
Search control 228
source code with heat map 234
specifying points in time 227
status control 227
strategy bar 228–230
system libraries 235
Target control 227
Target control’s menu 225
iOS devices
   adding to portal 309
devices
development certificates 308–309
Devices tab 308–310
provisioning 308
provisioning profiles 310
registering 309
iOS view controller 107–108
   .ipa files, importing into iTunes 124
Issue navigator
   checking 139
   features 13–14
issues
   checking code for 109
defined 109
   expanding into steps 110
   highlighting in Source editor 109
iTunes, importing .ipa files into 124

J
jump bar
   locating 17
   managing pop-up 153–154

K
Key Bindings preferences panel
   command sets 155–156
   opening 154
   shortcut keys 154–155
keyboard shortcuts
   builds 9
   customizing 20, 154–156
   finding quickly 162
   locating 154–155
   running unit tests 245
   searching within files 162
   tab use 24
KVC (Key-Value Coding), using 96

L
Lamp archive, displaying in Organizer 116
Lamp project
   adding files to 48
code signing 122
creation of 94
   pausing in debugger 102
Lamp.storedata file, contents of 95
layout behavior, changing 20
libraries
   comparing to loadable bundles 204
defined 202
dynamic 202
static 202
MOMs (managed object models) 90
in Data Model editor 91
in model graph mode 93
Mountain Lion, displaying Library folder in 95
Move Down tool, using 151
Move Up tool, using 151

N
navigation bar 10
navigation selection bar 10
Navigator area
Breakpoint navigator 15
Debug navigator 14–15
Issue navigator 13–14
Log navigator 16
Project navigator 11–14
Search Navigator 13
Symbol navigator 12
nibs
actions 64–65
compartmentalization 65
controller objects 64
File’s Owner 64
in Interface Builder 66
outlets 64–65
owners 64
nil, managing 142

O
Objective-C ARC, converting to 152
Objective-C syntax, updating 143
OCUnit
assertions 239
SenTestingKit framework 238–239
test case classes 238–239
test results 240
test suite runs 240
test targets 238–239
unit test failures 240
On/Off button, clicking 94–95
on-off switch, adding 73–74
OOP (object-oriented programming),
encapsulation 97–99
Organizer
annotating archives 116
Archives tab 116–117
Bookmarks mode 30
Documentation section 29
Explore mode 29
opening 25
Repositories tab 285
Search mode 30
OS X 3
P

packages
archive files as 117
Command Line Tools 281
persistent stores 91
phrases, finding 162
poison directive, using 261–262
post-action scripts
evironment variables 251
managing 250–252
Run Script action 251
pragma directives. See also Clang compiler
diagnostic ignored 261
ignore 263
poison 261
#pragma mark directive, using 153–154
pre-action scripts
evironment variables 251
managing 250–252
Run Script action 251
preprocessor. See also CPP (C preprocessor)
conditionals 263–264
poison directive 261–262
preprocessor macros. See also macros
BETABUILD 264–267
in build environment 264–267
#define directive 260–261
duplicating Release configuration 265
managing schemes 266
placing 260
using 153
using as functions 261
project content, renaming 157
Project editor. See also targets
App Icons 172
Build Phases tab 177–180
Build Rules tab 180
Build Settings tab 174–177
choosing keys for targets 173
Compile Sources phase 178
Copy Bundle Resources phase 178, 182
Copy Files phase 178–179
Copy Headers phase 179
defining UTIs for targets 174
Deployment Target 170
Document Types for targets 173
Info tab 172–174
Info.plist file 172
Link Binary With Libraries phase 178
Linked Frameworks and Libraries 170, 172
Run Script phase 180
sandbox entitlements 171
Summary tab 170–172
system services for targets 174
Target Dependencies phase 178
Target Summary tab 171
URL types for targets 174
project folders, copying items into 208
project modernization. See also versions of Xcode
automatic snapshot 137
considering 137
Validate Project Settings warning 136
Project navigator
features 11–12
SCM status badges 291
project settings
adjusting 156–157
File inspector 156
ProjectBuilder 3
projects
adding to workspaces 40–41
choosing templates 9
compiling 275
converting 143
creating 8–9
searching 13
setting options for 9
source files 10
using in workspaces 276
provisioning devices 308–310
provisioning profiles 310
creation of 120
downloading 121
explained 120
importing 120
maintenance 120
Q
Quick Help utility, using 30–31
R
Refactor preview 152–153
Refactor tools
Convert to Modern Objective-C Syntax 152
Convert to Objective-C ARC 152
Create Superclass 150
Encapsulate 151
Extract 150
Move Down tool 151
Move Up tool 151
Rename 149
relationships
inverses 91
requirements 91
release builds, archiving on 257–259
Release configuration, duplicating 265
Release mode 115
removing Assistant panes 19
scheme containers, defining 186
Scheme Editor sheet
Analyze action, invoking 194
Archive action 194
Build action 188–189
controls 187–188
post-action scripts 195
pre-action scripts 195
Profile action 193–194
Run action 189–191
Test action 192
Scheme Manager sheet 185
schemes. See also build system
auto-creating 186
building from workspaces 275–276
creating 185
described 166, 184
duplicating 186
exporting 186
finding 184
importing 186
managing for macros 266
removing 185
reordering 186
run destinations 184
sharing 186
SCM (source code management). See Git; SVN (Subversion); version control
SCM servers, SSH key 290
SCM status
checking in Project navigator 291–293
filtering Project navigator by 292
SCM tasks
browsing history 298–299
checking status 291–293
committing changes 293–295
comparing history 298–299
creating branches 296–297
discarding local changes 300
folders in SVN (Subversion) 291
merging changes 295–296
pushing changes 295–296
reverting local changes 300
revisions jump bar 298
scrubbing revision timeline 298
switching branches 296–297
updating changes 295–296
screenshots. See device screenshots
scripting example. See also CPP (C preprocessor)
Archive action 256
archive post-action run script 258–259
archiving on release builds 257–259
conditional archive 257–258
creating script 255–256
extending 259

S
sandbox entitlements, configuring 171
sandboxing
activating 197
explained 196
saving workspaces 40
scenes, creating with storyboards 84–85
Rename operation, performing 149
renaming
project content 157
symbols 148
repositories. See also version control
adding manually 288–289
change inclusion control 294
checking out 290
cloning 290
commit review sheet 293
committing changes 293–295
creating 286–288
creating with hosted service 301–302
Git 286–287
merging changes 295–296
in Organizer window 285–286
pulling changes 295–296
pushing changes 295
remote Git 288
specifying remote locations 290
SSH key for authentication 290
SVN (Subversion) 287–288
synchronizing 303
updating changes 295–296
resource files, navigating 11
resources. See also websites
Apple's developer forums 324
Cocoa Dev Central 324
CocoaDev Wiki 324
mogenerator 324
Stack Overflow 324
third-party 324
Xcode-Users Mailing List 324
restoring snapshots 60
revision control. See version control
revision timeline, scrubbing 298
revisions, accessing history of 299
revisions jump bar 298
Run, clicking 272
Run Active Script example, creating 255
run destinations, described 167
run logs, selecting 16
Run Script build phases 252–254
runtime
vs. compile time 268
debugging 102

INDEX
split pane editor. See Assistant tool
Spotlight plug-in, using with targets 181–182
springs, using 75
SSH key, setting for SCM servers 290
Stack Overflow, described 324
static analysis. See also debugging
flagging logic issues 112
garbage collection 111
logic errors 112
static analyzer
memory leaks 111
output 111–112
using 110–112
static libraries 202
storyboard file, opening 82
storyboards
advantage 81
scenes 84–85
segues 85–87
user interface 83
string-creation methods 141–142
struts, using 75
subclasses. See also mogenerator tool
choosing 46
capsulation 97–99
generating 95–99
managing 99
moving symbols to 151
subscripting
array access with 143
dictionary access with 143
superclasses
creating 150
moving symbols from 151
SVN (Subversion). See also version control
changing branches 297
comparing to Git 284
folder management 291
locally hosted repository 287–288
repository layout 289
reverse merge 300
Symbol navigator 12
symbols
finding documentation for 31
finding system-defined 261
as hard errors 261
moving to subclasses 151
renaming 148
using poison directive with 261
system framework example. See also frameworks
creating 204–206
LinkedWeb project 204
linking against 206–207
web view 205
T

tabbed coding 24
See also code focus ribbon
target dependency, establishing 183
TargetExplorer project, creating 170
TargetExplorer target, Product Name field 176–177
TargetImporter target, building 182–183 targets. See also build system; Project editor
Add Target button 170
adding 181–183
applications 170
choosing keys for 173
Configurations group 169–170
controls in filter bar 176
defining UTIs for 174
Deployment Target group 169
described 166
finding 169
for framework project 211
Localization group 170
project-wide settings 169–170
property list in Project editor 173
resources 182
selecting 169
sharing files between 183
specifying from command line 276
Spotlight plug-in 181–182
template chooser 168
types 168
Validate Settings button 170
templates, choosing for projects 9
See also File Template library
test action, described 277
Test Build action, using for unit testing 236–237
TestFlight service, using with iOS apps 124
Threads and Stacks navigator, using 103
toolbar 10
Tour.app project
creating 8–9
running 9

V

Validate button, action of 118–120
Validate Project Settings warning, clicking 136
variables
inspecting in Source editor 106
modifying 218–219
observing 218–219
uninitialized 112
using watchpoints 218–219
Variables pane, displaying 104
version control. See also Git; repositories; SVN (Subversion)
explained 57
feature branches 303–305
snapshots feature 58–60
Version editor
Blame mode 299
displaying 298
Log mode 299
versions of Xcode. See also project modernization
developer preview 280
using 279–280
workspaces
adding projects to 40–41
benefits 38
building 275–276
creating 40
defined 38
guidelines for use of 39
saving 40

Xcode
archive distribution 126
changing active 280
developer preview version 280
downloading 4
tools from earlier versions 5
using multiple versions of 279–280
Xcode projects. See projects
Xcode versions. See project modernization;
versions of Xcode
Xcode window 8
See also workspace window
xcodebuild command
build actions 277
running 274–276
for SDKs 278
version option 279
xcrun tool, using 280
xibs
actions 64–65
compartmentalization 65
controller objects 64
File's Owner 64
in Interface Builder 66
outlets 64–65
owners 64