Deitel® Series Page

Deitel® Developer Series

Android for Programmers: An App-Driven Approach, 2/E, Volume 1
C for Programmers with an Introduction to C11
C++11 for Programmers
C# 2012 for Programmers
iOS® 8 for Programmers: An App-Driven Approach with Swift™
Java™ for Programmers, 3/E
JavaScript for Programmers

How To Program Series

Android How to Program, 2/E
C++ How to Program, 9/E
C How to Program, 7/E
Java™ How to Program, Early Objects Version, 10/E
Java™ How to Program, Late Objects Version, 10/E
Internet & World Wide Web How to Program, 5/E
Visual C++® 2008 How to Program, 2/E
Visual Basic® 2012 How to Program, 6/E
Visual C#® 2012 How to Program, 5/E

Simply Series

Simply C++: An App-Driven Tutorial Approach
Simply Java™ Programming: An App-Driven Tutorial Approach

(continued from previous column)
Simply C#: An App-Driven Tutorial Approach

CourseSmart Web Books

www.deitel.com/books/CourseSmart/
C++ How to Program, 8/E and 9/E
Simply C++: An App-Driven Tutorial Approach
Java™ How to Program, 9/E and 10/E
Visual Basic® 2012 How to Program, 6/E
Visual Basic® 2010 How to Program, 5/E
Visual C#® 2012 How to Program, 5/E
Visual C#® 2010 How to Program, 4/E

LiveLessons Video Learning Products

www.deitel.com/books/LiveLessons/
Android App Development Fundamentals, 2/e
C++ Fundamentals
Java™ Fundamentals, 2/e
C# 2012 Fundamentals
C# 2010 Fundamentals
iOS® 6 App Development Fundamentals
JavaScript Fundamentals
Swift Fundamentals

To receive updates on Deitel publications, Resource Centers, training courses, partner offers and more, please join the Deitel communities on

- Facebook®—facebook.com/DeitelFan
- Twitter®—@deitel
- Google+™—google.com/+DeitelFan
- YouTube™—youtube.com/DeitelTV
- LinkedIn®—linkedin.com/company/deitel-&-associates

and register for the free Deitel® Buzz Online e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

To communicate with the authors, send e-mail to:
deitel@deitel.com

For information on Dive-Into® Series on-site seminars offered by Deitel & Associates, Inc. worldwide, write to us at deitel@deitel.com or visit:

www.deitel.com/training/

For continuing updates on Pearson/Deitel publications visit:

www.deitel.com
www.pearsonhighered.com/deitel/

Visit the Deitel Resource Centers that will help you master programming languages, software development, Android and iOS app development, and Internet- and web-related topics:

www.deitel.com/ResourceCenters.html
In Memory of Amar G. Bose, MIT Professor and Founder and Chairman of the Bose Corporation:

It was a privilege being your student—and members of the next generation of Deitels, who heard our dad say how your classes inspired him to do his best work.

You taught us that if we go after the really hard problems, then great things can happen.

Harvey Deitel
Paul and Abbey Deitel
Trademarks

Deitel, the double-thumbs-up bug and DIVE-INTO are registered trademarks of Deitel & Associates, Inc.

Apple, iOS, iPhone, iPad, iPod touch, Xcode, Swift, Objective-C, Cocoa and Cocoa Touch are trademarks or registered trademarks of Apple, Inc.

Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Google, Android, Google Play, Google Maps, Google Wallet, Nexus, YouTube, AdSense and AdMob are trademarks of Google, Inc.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screen shots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.
Preface xix
Before You Begin xxvii

I Introduction to iOS 8 App Development and Swift 1
1.1 Introduction 2
1.2 iPhone and iPad Sales Data 3
1.3 Gestures 4
1.4 Sensors 5
1.5 Accessibility 6
1.6 iPhone 6 and iPhone 6 Plus 7
1.7 iOS Operating System History and Features 8
1.7.1 iPhone Operating System 9
1.7.2 iPhone OS 2: Introducing Third-Party Apps and the App Store 9
1.7.3 iPhone OS 3 9
1.7.4 iOS 4 9
1.7.5 iOS 5 11
1.7.6 iOS 6 12
1.7.7 iOS 7 15
1.8 iOS 8 16
1.9 Apple Watch 18
1.10 App Store 19
1.11 Objective-C 20
1.12 Swift: Apple’s Programming Language of the Future 20
1.12.1 Key Features 20
1.12.2 Performance 22
1.12.3 Error Prevention 22
1.12.4 Swift Standard Library 23
1.12.5 Swift Apps and the Cocoa® and Cocoa Touch® Frameworks 23
1.12.6 Swift and Objective-C Interoperability 23
1.12.7 Other Apple Swift Resources 24
1.13 Can I Use Swift Exclusively? 24
1.13.1 Objective-C Programmers Who Are Developing New iOS Apps in Swift 24
1.13.2 Objective-C Programmers Who Are Enhancing Existing iOS Apps with Swift 25
1.13.3 Java, C++ and C# Programmers Who Are New to iOS App Development 25
1.13.4 Rapid Evolution Expected 25
1.13.5 Mixing Swift and Objective-C Code 25
1.14 Cocoa Touch ® iOS Frameworks 25
1.15 Xcode 6 ® Integrated Development Environment 31
1.16 Object Oriented-Programming Review 33
  1.16.1 Automobile as an Object 34
  1.16.2 Methods and Classes 34
  1.16.3 Instantiation 34
  1.16.4 Reuse 34
  1.16.5 Messages and Method Calls 35
  1.16.6 Attributes and Properties 35
  1.16.7 Encapsulation and Information Hiding 35
  1.16.8 Inheritance 35
  1.16.9 Protocols 35
  1.16.10 Design Patterns 36
1.17 Test-Driving the Tip Calculator App in the iPhone and iPad Simulators 36
1.18 What Makes a Great App? 38
1.19 iOS Security 40
1.20 iOS Publications and Forums 41
1.21 Wrap-Up 42

2 Welcome App 43
Dive-Into® Xcode: Introducing Visual User Interface Design with Cocoa Touch, Interface Builder, Storyboarding and Auto Layout, Universal Apps, Accessibility, Internationalization
2.1 Introduction 44
2.2 Technologies Overview 45
  2.2.1 Xcode and Interface Builder 45
  2.2.2 Labels and Image Views 45
  2.2.3 Asset Catalogs and Image Sets 46
  2.2.4 Running the App 46
  2.2.5 Accessibility 46
  2.2.6 Internationalization 46
2.3 Creating a Universal App Project with Xcode 46
  2.3.1 Launching Xcode 47
  2.3.2 Projects and App Templates 47
  2.3.3 Creating and Configuring a Project 48
2.4 Xcode Workspace Window 49
  2.4.1 Navigator Area 50
2.4.2 Editor Area 50
2.4.3 Utilities Area and Inspectors 51
2.4.4 Debug Area 51
2.4.5 Xcode Toolbar 51
2.4.6 Project Navigator 52
2.4.7 Keyboard Shortcuts 52

2.5 Storyboarding the Welcome App’s UI 52
2.5.1 Configuring the App for Portrait and Landscape Orientations 53
2.5.2 Providing an App Icon 53
2.5.3 Creating an Image Set for the App’s Image 55
2.5.4 Overview of the Storyboard and the Xcode Utilities Area 56
2.5.5 Adding an Image View to the UI 58
2.5.6 Using Inspectors to Configure the Image View 58
2.5.7 Adding and Configuring the Label 60
2.5.8 Using Auto Layout to Support Different Screen Sizes and Orientations 62

2.6 Running the Welcome App 64
2.6.1 Testing on the iOS Simulator 64
2.6.2 Testing on a Device (for Paid Apple iOS Developer Program Members Only) 67

2.7 Making Your App Accessible 67
2.7.1 Enabling Accessibility for the Image View 67
2.7.2 Confirming Accessibility Text with the Simulator’s Accessibility Inspector 68

2.8 Internationalizing Your App 69
2.8.1 Locking Your UI During Translation 70
2.8.2 Exporting Your UI’s String Resources 71
2.8.3 Translating the String Resources 72
2.8.4 Importing the Translated String Resources 72
2.8.5 Testing the App in Spanish 73

2.9 Wrap-Up 74

3 Tip Calculator App 75
Introducing Swift, Text Fields, Sliders, Outlets, Actions, View Controllers, Event Handling, NSDecimalNumber, NSNumberFormatter and Automatic Reference Counting

3.1 Introduction 76
3.2 Technologies Overview 77
3.2.1 Swift Programming 77
3.2.2 Swift Apps and the Cocoa Touch® Frameworks 78
3.2.3 Using the UIKit and Foundation Frameworks in Swift Code 79
3.2.4 Creating Labels, a Text Field and a Slider with Interface Builder 79
3.2.5 View Controllers 79
3.2.6 Linking UI Components to Your Swift Code 79
## Contents

3.2.7 Performing Tasks After a View Loads 80  
3.2.8 Financial Calculations with NSDecimalNumber 80  
3.2.9 Formatting Numbers as Locale-Specific Currency and Percentage Strings 82  
3.2.10 Bridging Between Swift and Objective-C Types 82  
3.2.11 Swift Operator Overloading 82  
3.2.12 Variable Initialization and Swift Optional Types 82  
3.2.13 Value Types vs. Reference Types 83  
3.2.14 Code Completion in the Source-Code Editor 84  
3.3 Building the App's UI 85  
3.3.1 Creating the Project 85  
3.3.2 Configuring the Size Classes for Designing a Portrait Orientation iPhone App 86  
3.3.3 Adding the UI Components 86  
3.3.4 Adding the Auto Layout Constraints 93  
3.4 Creating Outlets with Interface Builder 96  
3.5 Creating Actions with Interface Builder 99  
3.6 Class ViewController 100  
3.6.1 import Declarations 101  
3.6.2 ViewController Class Definition 101  
3.6.3 ViewController’s IBOutlet Properties 102  
3.6.4 Other ViewController Properties 103  
3.6.5 Overridden UIViewController method viewDidLoad 103  
3.6.6 ViewController Action Method calculateTip 104  
3.6.7 Global Utility Functions Defined in ViewController.swift 107  
3.7 Wrap-Up 109

## 4 Twitter® Searches App 111


4.1 Introduction 112  
4.2 Test-Driving the App 113  
4.3 Technologies Overview 120  
4.3.1 Master-Detail Application Template 120  
4.3.2 WebView—Displaying Web Content in an App 120  
4.3.3 Swift: Array and Dictionary Collections 120  
4.3.4 UserDefaults—Local Key–Value Pair Storage for App Settings 122  
4.3.5 iCloud Key–Value Pair Storage with NSUbiquitousKeyValueStore 122  
4.3.6 Social Framework 123  
4.3.7 Model-View-Controller (MVC) Design Pattern 123  
4.3.8 Swift: Conforming to Protocols 124  
4.3.9 Swift: Exposing Methods to Cocoa Touch Libraries 125  
4.3.10 UIAlertController for Alert Dialogs 125
4.3.11UILongPressGestureRecognizer 125
4.3.12iOS Design Patterns Used in This App 125
4.3.13Swift: External Parameter Names 126
4.3.14Swift: Closures 127

4.4Building the App’s UI 128
4.4.1Creating the Project 128
4.4.2Examining the Default Master-Detail Application 129
4.4.3Configuring the Master and Detail Views 131
4.4.4Creating class Model 131

4.5Class Model 131
4.5.1ModelDelegate Protocol 132
4.5.2Model Properties 132
4.5.3Model Initializer and synchronize Method 133
4.5.4Methods tagAtIndex, queryForTag and queryForTagAtIndex, and Property count 136
4.5.5Method deleteSearchAtIndex 137
4.5.6Method moveTagAtIndex 137
4.5.7Method updateUserDefaults 138
4.5.8Method updateSearches 139
4.5.9Method performUpdates 140
4.5.10Method saveQuery 141

4.6Class MasterViewController 141
4.6.1MasterViewController Properties and modelDataChanged Method 141
4.6.2Method awakeFromNib 143
4.6.3Overridden UIViewController Method viewDidLoad and Method addButtonPressed 143
4.6.4Methods tableViewCellLongPressed and displayLongPressOptions 145
4.6.5Method displayAddEditSearchAlert 147
4.6.6Method shareSearch 149
4.6.7Overridden UIViewController Method prepareForSegue 150
4.6.8Method urlEncodeString 151
4.6.9UITableViewDataSource Callback Methods 151

4.7Class DetailViewController 154
4.7.1Overridden UIViewController Method viewDidLoad 156
4.7.2Overridden UIViewController Method viewWillAppear 156
4.7.3Overridden UIViewController Method willDisappear 156
4.7.4UIViewDelegate Protocol Methods 156

4.8Wrap-Up 157

5Flag Quiz App 158
UISegmentedControls, UISwitches, Outlet Collections, View Animations, UINavigationController, Segues, NSBundle, Scheduling Tasks with Grand Central Dispatch
## Contents

5.1 Introduction 159
5.2 Test-Driving the **Flag Quiz** App 161
5.3 Technologies Overview 165
  5.3.1 Designing a Storyboard from Scratch 165
  5.3.2 `UINavigationController` 165
  5.3.3 Storyboard Segues 165
  5.3.4 `UISegmentedControls` 165
  5.3.5 `UISwitches` 165
  5.3.6 Outlet Collections 166
  5.3.7 Using the App’s Main `NSBundle` to Get a List of Image Filenames 166
  5.3.8 Using Grand Central Dispatch to Perform a Task in the Future 166
  5.3.9 Applying an Animation to a `UIView` 167
  5.3.10 Darwin Module—Using Predefined C Functions 167
  5.3.11 Random-Number Generation 167
  5.3.12 Swift Features Introduced 168
5.4 Building the GUI 170
  5.4.1 Creating the Project 170
  5.4.2 Designing the Storyboard 171
  5.4.3 Configuring the View Controller Classes 173
  5.4.4 Creating the UI for the `QuizViewController` 173
  5.4.5 Auto Layout Settings for the `QuizViewController` UI 175
  5.4.6 `QuizViewController` Outlets and Actions 175
  5.4.7 Creating the UI for the `SettingsViewController` 176
  5.4.8 `SettingsViewController` Outlets and Actions 177
  5.4.9 Creating Class `Model` 178
  5.4.10 Adding the Flag Images to the App 178
5.5 `Model` Class 178
  5.5.1 `ModelDelegate` Protocol 178
  5.5.2 `Model` Properties 179
  5.5.3 `Model` Initializer and regionsChanged Method 180
  5.5.4 `Model` Computed Properties 182
  5.5.5 `Model` Methods $toggleRegion$, $setNumberOfGuesses$ and $notifyDelegate$ 182
  5.5.6 `Model` Method newQuizCountries 183
5.6 `QuizViewController` Class 184
  5.6.1 Properties 184
  5.6.2 Overridden `UIViewController` Method `viewDidLoad`, and Methods settingsChanged and resetQuiz 185
  5.6.3 Methods nextQuestion and countryFromFilename 186
  5.6.4 Method submitGuess 188
  5.6.5 Method shakeFlag 190
  5.6.6 Method displayQuizResults 191
  5.6.7 Overridden `UIViewController` Method `prepareForSegue` 192
  5.6.8 Array Extension shuffle 193
5.7 `SettingsViewController` Class 193
  5.7.1 Properties 193
6 Cannon Game App

Xcode Game Template, SpriteKit, Animation, Graphics, Sound, Physics, Collision Detection, Scene Transitions, Listening for Touches

6.1 Introduction 199
6.2 Test-Driving the Cannon Game App 202
6.3 Technologies Overview 203
   6.3.1 Xcode Game Template and SpriteKit 203
   6.3.2 Adding Sound with the AVFoundation Framework and AVAudioPlayer 204
   6.3.3 SpriteKit Framework Classes 204
   6.3.4 SpriteKit Game Loop and Animation Frames 205
   6.3.5 Physics 206
   6.3.6 Collision Detection and the SKPhysicsContactDelegate Protocol 206
   6.3.7 CGGeometry Structures and Functions 207
   6.3.8 Overriding UIResponder Method touchesBegan 208
   6.3.9 Game-Element Sizes and Velocities Based on Screen Size 208
   6.3.10 Swift Features 208
   6.3.11 NSLocalizedString 209
6.4 Creating the Project and Classes 209
6.5 Class GameViewController 211
   6.5.1 Overridden UIViewController Method viewDidLoad 212
   6.5.2 Why Are the AVAudioPlayer Variables Global? 213
   6.5.3 Autogenerated Methods That We Deleted from Class GameViewController 213
6.6 Class Blocker 213
   6.6.1 BlockerSize enum and Class Blocker’s Properties 214
   6.6.2 Blocker Initializers 214
   6.6.3 Methods startMoving, playHitSound and blockerTimePenalty 217
6.7 Class Target 218
   6.7.1 TargetSize and TargetColor enums 218
   6.7.2 Class Target Properties 219
   6.7.3 Target Initializers 219
   6.7.4 Methods startMoving, playHitSound and targetTimeBonus 220
6.8 Class Cannon 221
   6.8.1 Cannon Properties 221
   6.8.2 Cannon Initializers 222
   6.8.3 Method rotateToPointAndFire 223
   6.8.4 Methods fireCannonball and createCannonball 224
6.9 Class GameScene 226
   6.9.1 CollisionCategory struct 226
6.9.2 GameScene Class Definition and Properties  227
6.9.3 Overridden SKScene Method didMoveToView  228
6.9.4 Method createLabels  230
6.9.5 SKPhysicsContactDelegate Method didBeginContact and Supporting Methods  231
6.9.6 Overridden UIResponder Method touchesBegan  233
6.9.7 Overridden SKScene Method update and Method gameOver  234

6.10 Class GameOverScene  235
6.11 Programmatic Internationalization  237
6.12 Wrap-Up  240

7 Doodlz App  242
Multi-Touch Event Handling, Graphics, UIBezierPaths, Drawing with a Custom UIView Subclass,UIToolbar, UIBarButtonItem, Accelerometer Sensor and Motion Event Handling

7.1 Introduction  243
7.2 Test-Driving the Doodlz App  244
7.3 Technologies Overview  249
  7.3.1 Drawing with UIView Subclasses, Method drawRect, UIBezierPaths and the UIKit Graphics System  249
  7.3.2 Processing Multiple Touch Events  250
  7.3.3 Listening for Motion Events  250
  7.3.4 Rendering the Drawing as a UIImage  250
  7.3.5 Storyboard Loading Initialization  251
7.4 Building the App’s UI and Adding Its Custom Classes  251
  7.4.1 Creating the Project  251
  7.4.2 Creating the Initial View Controller’s User Interface  252
  7.4.3 Creating the Color View Controller’s User Interface  254
  7.4.4 Creating the Stroke View Controller’s User Interface  255
  7.4.5 Adding the Squiggle Class  257
7.5 ViewController Class  257
  7.5.1 ViewController Class Definition, Property and Delegate Methods  257
  7.5.2 Overridden UIViewController Method prepareForSegue  258
  7.5.3 ViewController Methods undoButtonPressed, clearButtonPressed and displayEraseDialog  259
  7.5.4 Overridden UIResponder Method motionEnded  260
  7.5.5 ViewController Method actionButtonPressed  260
7.6 Squiggle Class  261
7.7 DoodleView Class  262
  7.7.1 DoodleView Properties  262
  7.7.2 DoodleView Initializer  262
  7.7.3 DoodleView Methods undo and clear  263
  7.7.4 Overridden UIView Method drawRect  263
  7.7.5 Overridden UIResponder Methods for Touch Handling  264
8.6.4 Overridden UIViewController Method prepareForSegue 291
8.6.5 AddEditTableViewControllerDelegate Method didSaveContact 292
8.6.6 DetailViewControllerDelegate Method didEditContact 294
8.6.7 Method displayError 294
8.6.8 UITableViewDelegate Methods 294
8.6.9 Autogenerated NSFetchedResultsController and NSFetchedResultsControllerDelegate Methods 296

8.7 DetailViewController Class 299
8.7.1 DetailViewControllerDelegate Protocol 300
8.7.2 DetailViewController Properties 300
8.7.3 Overridden UIViewController Method viewDidLoad and Method didSaveContact 301
8.7.4 AddEditTableViewControllerDelegate Method didSaveContact 302
8.7.5 Overridden UIViewController Method prepareForSegue 302

8.8 AddEditTableViewController Class 303
8.8.1 AddEditTableViewControllerDelegate Protocol 303
8.8.2 AddEditTableViewController Properties 303
8.8.3 Overridden UIViewController Methods viewWillAppear and viewWillAppear 304
8.8.4 Overridden UIViewController Method viewDidLoad 305
8.8.5 Methods keyboardWillShow and keyboardWillHide 306
8.8.6 UITextFieldDelegate Method textFieldShouldReturn 307
8.8.7 @IBAction saveButtonPressed 308

8.9 AppDelegate Class 309
8.9.1 UIApplicationDelegate Protocol Method application:didFinishLaunchingWithOptions: 309
8.9.2 UISplitViewControllerDelegate Protocol Method 309
8.9.3 Properties and Methods That Support the App’s Core Data Capabilities 310

8.10 Wrap-Up 311

9 App Store and App Business Issues 312

Introducing the iOS Developer Program and iTunes® Connect 313
9.1 Introduction 313
9.2 iOS Developer Program: Setting Up Your Profile for Testing and Submitting Apps 313
  9.2.1 Setting Up Your Development Team 314
  9.2.2 Provisioning a Device for App Testing 315
  9.2.3 TestFlight Beta Testing 316
  9.2.4 Creating Explicit App IDs 317
9.3 iOS Human Interface Guidelines 317
9.4 Preparing Your App for Submission through iTunes Connect 318
9.5 Pricing Your App: Fee or Free 321
  9.5.1 Paid Apps 321
9.5.2 Free Apps 322
9.6 Monetizing Apps 324
9.6.1 Using In-App Purchase to Sell Virtual Goods 324
9.6.2 iAd In-App Advertising 325
9.6.3 App Bundles 326
9.6.4 Developing Custom Apps for Organizations 326
9.7 Managing Your Apps with iTunes Connect 327
9.8 Information You’ll Need for iTunes Connect 328
9.9 *iTunes Connect Developer Guide: Steps for Submitting Your App to Apple* 330
9.10 Marketing Your App 331
9.11 Other Popular Mobile App Platforms 336
9.12 Tools for Multiple-Platform App Development 336
9.13 Wrap-Up 337

Index 339
This page intentionally left blank
Welcome to the world of iOS® 8 app development with Apple’s new and rapidly evolving Swift™ programming language, the Cocoa Touch® frameworks and the Xcode® 6 development tools.

iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e presents leading-edge mobile computing technologies for professional software developers. At the heart of the book is our app-driven approach—we present concepts in the context of seven completely coded and fully tested iOS 8 apps rather than using code snippets. We’ve always favored teaching by example—in an app-development world, the best examples are real, working apps.

Chapters 2–8 each present one app. We begin each of these chapters with an introduction to the app, an app test-drive showing one or more sample executions and a technologies overview. Then we proceed with a detailed source code walkthrough. We don’t try to be exhaustive—our goal is to get you developing apps quickly with the Xcode 6 integrated development environment, the Swift programming language and the Cocoa Touch frameworks. All of the source code is available at

http://www.deitel.com/books/iOS8FP1

We recommend that you keep the code open in the IDE as you read the book. You should study the apps sequentially because each introduces technologies that are used in subsequent apps.

This book is Volume 1 of what will become a multi-volume set. Volume 1 presents seven fully coded apps of increasingly rich functionality. The apps cover a range of topics from simple visual programming (without code), to simple programming with Swift, to more involved programming.

Explosive Growth of the iPhone and iPad Is Creating Opportunity for Developers

iPhone and iPad device sales have been growing exponentially, creating significant opportunities for iOS app developers. The first-generation iPhone, released in June 2007, sold 6.1 million units in its initial five quarters of availability.1 The iPhone 5s and the iPhone 5c, released simultaneously in September 2013, sold over nine million combined in the first three days of availability.2 The most recent iPhone 6 and iPhone 6 Plus, announced in September 2014, pre-sold four million combined in just one day—double the number of

iPhone 5 pre-sales in its first day of pre-order availability.\(^3\) Apple sold 10 million iPhone 6 and iPhone 6 Plus units combined in their first weekend of availability.\(^4\)

Sales of the iPad are equally impressive. The first generation iPad, launched in April 2010, sold 3 million units in its first 80 days of availability\(^5\) and over 40 million worldwide by September 2011.\(^6\) The iPad mini with Retina display (the second-generation iPad mini) and the iPad Air (the fifth-generation iPad) were released in November 2013. In just the first quarter of 2014, Apple sold a record 26 million iPads.\(^7\)

There are over 1.3 million apps in the App Store\(^8\) and over 75 billion iOS apps have been downloaded.\(^9\) The potential for iOS app developers is enormous.

**SafariBooksOnline e-Book and LiveLessons Videos**

If you have a subscription to Safari Books Online (www.safaribooksonline.com), check out the e-book and LiveLessons video versions of *iOS® 8 for Programmers: An App-Driven Approach with Swift*. Safari is a subscription service popular with large companies, colleges, libraries and individuals who would like access to video training and electronic versions of print publications.

**Copyright Notice and Code License**

All of the code and iOS apps in the book are copyrighted by Deitel & Associates, Inc. The sample iOS apps are licensed under a Creative Commons Attribution 3.0 Unported License (http://creativecommons.org/licenses/by/3.0), with the exception that they may not be reused in any way in educational tutorials and textbooks, whether free or for a fee and whether in print or digital format. Additionally, the authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs. You’re welcome to use the apps in the book as shells for your own apps, building on their existing functionality. If you have any questions, contact us at deitel@deitel.com.

**Intended Audience**

This book is part of the *Deitel Developer Series* intended for experienced programmers who know object-oriented programming in a C-based programming language such as Objective-C, Java, C# or C++. Objective-C experience is helpful, but not specifically required. If you have not worked in any of these languages, you should still be able to learn a good amount of iOS 8 app development and object-oriented programming in Swift and Cocoa.

---

9. [http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/](http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/).
Touch by reading the code and our code walkthroughs, running the apps and observing the results. We review the basics of object-oriented programming in Chapter 1. We also assume that you’re comfortable with OS X, as you’ll need to work on a Mac to develop iOS apps. The book does not include exercises.

This book is not a Swift tutorial, but it presents a significant amount of Swift in the context of iOS 8 app development. If you’re interested in learning Swift, check out our publications:

- *Swift for Programmers* print book ([www.deitel.com/books/swiftfp](http://www.deitel.com/books/swiftfp)). This book is also available as an e-book on SafariBooksOnline.com, Informit.com, Amazon® Kindle® and a growing number of other electronic platforms.

**Academic Bundle iOS® 8 for Programmers and Swift™ for Programmers**

The Academic Bundle iOS® 8 for Programmers and Swift™ for Programmers is designed for professionals, students and instructors interested in learning or teaching iOS 8® app development with a broader and deeper treatment of Swift. You can conveniently order the Academic Bundle with one ISBN: 0-13-408775-5. The Academic Bundle includes:

- *Swift™ for Programmers* (print book)
- *iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e* (print book)
- Access Code Card for Academic Package to accompany *Swift™ for Programmers*
- Access Code Card for Academic Package to accompany *iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e*

The two Access Code Cards for the Academic Packages (when used together) give you access to the companion websites, which include self-review questions (with answers), short-answer questions, programming exercises, programming projects and selected videos chosen to get you up to speed quickly with Xcode 6, visual programming and basic Swift-based, iOS 8 programming.

**Ordering the Books and Supplements Separately**

The print books and Access Code Cards may be purchased separately using the following ISBNs:

- Standalone access code card for Academic Package to accompany *Swift™ for Programmers*: ISBN 0-13-405818-6
Instructor Supplements
Instructor supplements are available online at Pearson’s Instructor Resource Center (IRC). The supplements include:

- Solutions Manual with selected solutions to the short-answer exercises.
- Test Item File of multiple-choice examination questions (with answers).
- PowerPoint® slides with the book’s source code and tables.

Please do not write to us requesting access to the Pearson Instructor’s Resource Center. Certified instructors who adopt the book for their courses can obtain password access from their regular Pearson sales representatives (www.pearson.com/relocator). Solutions are not provided for “project” exercises.

Key Features of iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e

Here are some of this book’s key features:

App-Driven Approach. Chapters 2–8 each present one completely coded app—we discuss what the app does, show screen shots of the app in action, test-drive it and overview the technologies and architecture we’ll use to build it. Then we build the app’s GUI and resource files, present the complete code and do a detailed code walkthrough. We discuss the Swift programming concepts and demonstrate the functionality of the Cocoa Touch APIs used in the app.

Swift Programming Language. Swift was arguably the most significant announcement at Apple’s Worldwide Developers Conference in 2014. Although apps can still be programmed in Objective-C, Swift is Apple’s language of the future for app development and systems programming.

We’ve programmed all the book’s apps in Swift—previous editions were programmed in Objective-C. Swift is a contemporary language with simpler syntax than Objective-C. It enables a clean, concise coding style and has a strong focus on error prevention. Our own experience with Swift has been that we can develop apps faster and with significantly less code than when we program in Objective-C.

At the time of this writing, Apple had not as yet published coding guidelines for Swift—we’ll conform to them when they appear. We use a mix of Apple’s Objective-C coding guidelines and Deitel coding guidelines for this edition.

Cocoa Touch Frameworks. Cocoa Touch is the groups of reusable components (known as frameworks) for building iOS apps. Throughout this edition, we use many of the Cocoa Touch features and frameworks, even though they’re programmed mostly in Objective-C. Apple has made this easy with a technique called “bridging.” We simply call Cocoa Touch methods and receive the returns transparently—it feels as if Cocoa Touch is written in Swift.

iOS SDK 8. Between Volumes 1 and 2 of iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e, we cover a broad range of the features included in iOS Software Development Kit (SDK) 8.
**Xcode 6.** Apple’s Xcode integrated development environment (IDE) and its associated tools for Mac OS X, combined with the iOS 8 Software Development Kit (SDK), provide all the software you need to develop and test iOS 8 apps.

**Instruments.** The Instruments tool, which is packaged with the SDK, is used to inspect apps while they’re running to check for memory leaks, monitor processor (CPU) usage and network activity, and review the objects allocated in memory.

**iOS Human Interface Guidelines.** We encourage you to read Apple’s *iOS Human Interface Guidelines* (HIG) and follow them as you design and develop your apps. The HIG discusses human interface principles, app design strategies, user experience guidelines, iOS technology usage guidelines and more. We gradually introduce HIG issues as we encounter them in the apps we develop. Section 9.3 overviews the HIG, discusses features and functionality required to get your app accepted on the App Store and lists reasons why Apple rejects apps.

**Multimedia.** The apps use iOS 8 multimedia capabilities, including graphics, images, animation and audio. We’ll present video capabilities in Volume 2.

**iOS App Design Patterns.** This book adheres to Apple’s app coding standards, including design patterns, such as Model-View-Controller (MVC), Delegation, Target-Action and Observer.

**Features**

**Syntax Coloring.** For readability, we syntax color the code, similar to Xcode’s use of syntax coloring. Our syntax-coloring conventions are as follows:

- comments appear in green
- keywords appear in blue
- constants and literal values appear in light blue
- all other code appears in black

**Code Highlighting.** We highlight the key code segments in each app that exercise the new technologies the app features.

**Using Fonts for Emphasis.** We place key terms and the index’s page reference for each term’s defining occurrence in **bold maroon** text for easier reference. We emphasize on-screen components in the **bold Helvetica** font (e.g., the **File** menu) and emphasize Swift program text in the Lucida font (for example, `var x = 5`).

**Source Code.** All of the source-code examples are available for download from:

http://www.deitel.com/books/iOS8FP1/

**Documentation.** All of the manuals that you’ll need to develop iOS 8 apps are available free at [http://developer.apple.com/ios](http://developer.apple.com/ios).

**Chapter Objectives.** Each chapter begins with a list of objectives.

**Figures.** Abundant tables, source-code listings and iOS screen shots are included.

**Index.** We include an extensive index, which is especially useful when you use the book as a reference. Defining occurrences of key terms are highlighted with a **bold** page number.
Preface

iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 2

Volume 2 of this series will contain additional app-development chapters. For the status of Volume 2 and for continuing book updates, visit http://www.deitel.com/books/iOS8fp2

iOS® 8 Fundamentals LiveLessons Video Training Products

Our iOS 8 Fundamentals LiveLessons videos show you what you need to know to start building robust, powerful iOS apps with the iOS Software Development Kit (SDK) 8, the Swift programming language, Xcode and Cocoa Touch. It will include approximately 10+ hours of expert training synchronized with iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e. For additional information about Deitel LiveLessons video products, visit www.deitel.com/livelessons or contact us at deitel@deitel.com. You can also access our LiveLessons videos if you have a subscription to Safari Books Online (www.safaribooksonline.com). You can get a free 10-day subscription to SafariBooksOnline at http://www.safaribooksonline.com/register

Acknowledgments

We’d like to thank Barbara Deitel for long hours spent researching iOS 8 and its many related technologies.

Pearson Education Team

We’re fortunate to have worked on this project with the dedicated publishing professionals at Prentice Hall/Pearson. We appreciate the extraordinary efforts and 19-year mentorship of our friend and professional colleague Mark L. Taub, Editor-in-Chief of Pearson Technology Group. Kim Boedigheimer recruited distinguished members of the iOS community to review the manuscript and she managed the review process. We selected the cover art and Chuti Prasertsith designed the cover. John Fuller managed the book’s publication.

Reviewers

We wish to acknowledge the efforts of our current and recent editions reviewers. They scrutinized the text and the programs and provided countless suggestions for improving the presentation.

iOS 8 edition reviewers: Scott Bossak (Lead iOS Developer, Thrillist Media Group), Charles E. Brown (Independent Contractor affiliated with Apple and Adobe), Matt Galloway (iOS Developer and author of Effective Objective-C 2.0), Michael Haberman (Software Engineer, Instructor at University of Illinois), Rob McGovern (Indie Developer) and Rik Watson (Technical Team Lead, HP Enterprise Services).

Earlier iOS editions reviewers: Cory Bohon (Indie Developer at CocoaApp.com and Writer at Mac|Life), Scott Gustafson (Owner/Developer, Garlic Software LLC), Firoze Lafeer (Master Developer, Capital One Labs), Dan Lingman (Partner, www.nogotog-
ames.com), Marcantonio Magnarapa (Chief Mobile Officer, www.bemyeye.com), Nik Saers (iOS Developer, SAERS), Zach Saul (Founder, Retronyms) and Rik Watson (then a Senior Software Engineer, Lockheed Martin).

Keeping in Touch with the Authors
As you read the book, we’d appreciate your comments, criticisms, corrections and suggestions for improvement. Please address all correspondence to:

deitel@deitel.com

We’ll respond promptly. For updates on this book, visit

http://www.deitel.com/books/iOS8FP1

subscribe to the Deitel® Buzz Online newsletter at

http://www.deitel.com/newsletter/subscribe.html

and join the Deitel social networking communities on

- Facebook® (http://www.deitel.com/deitelfan)
- Twitter® (@deitel)
- LinkedIn® (http://linkedin.com/company/deitel-&-associates)
- Google+™ (http://google.com/+DeitelFan)
- YouTube® (http://youtube.com/DeitelTV)

Well, there you have it! We hope you enjoy working with iOS® 8 for Programmers: An App-Driven Approach with Swift, Volume 1 as much as we enjoyed writing it!

Paul, Harvey and Abbey Deitel

About the Authors
Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. He holds the Java Certified Programmer and Java Certified Developer designations, and is an Oracle Java Champion. Paul was also named as a Microsoft® Most Valuable Professional (MVP) for C# in 2012–2014. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses worldwide to clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey Deitel, are the world’s best-selling programming-language textbook/professional book/video authors.

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has over 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels’ publications
have earned international recognition, with translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to corporate, academic, government and military clients.

Abbey Deitel, President of Deitel & Associates, Inc., is a graduate of Carnegie Mellon University’s Tepper School of Management where she received a B.S. in Industrial Management. Abbey has been managing the business operations of Deitel & Associates, Inc. for 17 years. She has contributed to numerous Deitel & Associates publications including Swift™ for Programmers and, together with Paul and Harvey, is the co-author of iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e, Android for Programmers: An App-Driven Approach, 2/e, Internet & World Wide Web How to Program, 5/e, Visual Basic 2012 How to Program, 6/e and Simply Visual Basic 2010, 5/e.

About Deitel® & Associates, Inc.
Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in mobile app development, computer programming languages, object technology and Internet and web software technology. The company’s training clients include many of the world’s largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including Swift™, Objective-C and iOS® app development, Java™, Android app development, C++, C, Visual C#®, Visual Basic®, Python®, object technology, Internet and web programming and a growing list of additional programming and software development courses.

Through its 40-year publishing partnership with Pearson/Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks and professional books in print and a wide range of e-book formats, and LiveLessons video courses. Deitel & Associates, Inc. and the authors can be reached at:

deitel@deitel.com

To learn more about Deitel’s Dive-Into® Series Corporate Training curriculum, visit:

http://www.deitel.com/training

To request a proposal for worldwide on-site, instructor-led training at your organization, send an e-mail to deitel@deitel.com.

Individuals wishing to purchase Deitel books and LiveLessons video training can do so through www.deitel.com. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

This section contains information you should review before using this book. Updates will be posted at:

http://www.deitel.com/books/iOS8FP1

Font and Naming Conventions
We use fonts to distinguish between on-screen components (such as menu names and menu items) and Swift code. Our convention is to emphasize on-screen components in a sans-serif bold Helvetica font (for example, File menu) and to emphasize Swift code and commands in a sans-serif Lucida font (for example, import UIKit). When building user interfaces (UIs) using Xcode’s Interface Builder, we also use the bold Helvetica font to refer to property names for UI components (such as a Label’s Text property).

Conventions for Referencing Menu Items in a Menu
We use the > character to indicate selecting a menu item from a menu. The notation File > Open… indicates that you should select the Open… menu item from the File menu.

Software Used in this Book
To execute our apps and write your own iOS 8 apps, you must install Xcode 6. You can install the currently released Xcode version for free from the Mac App Store. When you open Xcode for the first time, it will download and install additional features required for development. For the latest information about Xcode, visit

https://developer.apple.com/xcode

A Note Regarding the Xcode 6 Toolbar Icons
We developed this book’s examples with Xcode 6 on OS X Yosemite. If you’re running OS X Mavericks, some Xcode toolbar icons we show in the text may differ on your screen.

Becoming a Registered Apple Developer
Registered developers have access to the online iOS documentation and other resources. Apple also now makes Xcode pre-release versions (such as the next point release or major version) available to all registered Apple developers. To register, visit:

https://developer.apple.com/register

To download the next pre-release Xcode version, visit:

Before You Begin

Once you download the DMG (disk image) file, double click it to launch the installer, then follow the on-screen instructions.

Fee-Based Developer Programs

iOS Developer Program
The fee-based iOS Developer Program allows you to load your iOS apps onto iOS devices for testing and to submit your apps to the App Store. If you intend to distribute iOS apps, you’ll need to join the fee-based program. You can sign up at

https://developer.apple.com/programs

iOS Developer Enterprise Program
Organizations may register for the iOS Developer Enterprise Program at


which enables developers to deploy proprietary iOS apps to employees within their organization.

iOS Developer University Program
Colleges and universities interested in offering iOS app-development courses can apply to the iOS Developer University Program at


Qualifying schools receive free access to all the developer tools and resources. Students can share their apps with each other and test them on iOS devices.

Adding Your Paid iOS Developer Program Account to Xcode
Xcode can interact with your paid iOS Developer Program account on your behalf so that you can install apps onto your iOS devices for testing. If you have a paid iOS Developer Program account, you can add it to Xcode. To do so:

1. Select Xcode > Preferences....
2. In the Accounts tab, click the + button in the lower left corner and select Add Apple ID....
3. Enter your Apple ID and password, then click Add.

Obtaining the Code Examples
The final versions of the apps you’ll build in this book are available for download as a ZIP file from

http://www.deitel.com/books/iOS8FP1

under the heading Download Code Examples and Other Premium Content. When you click the link to the ZIP file, it will be placed by default in your user account’s Downloads folder. We assume that the examples are located in the iOS8Examples folder in your user account’s Documents folder. You can use Finder to move the ZIP file there, then double click the file to extract its contents.
Xcode Projects

For each app, we provide a project that you can open in Xcode by double clicking its project file, which has the .xcodeproj extension. You’ll use these projects to test-drive the apps before building them.

Configuring Xcode to Display Line Numbers

Many programmers find it helpful to display line numbers in the code editor. To do so:

1. Open Xcode and select Preferences... from the Xcode menu.
2. Select the Text Editing tab, then ensure that theEditing subtab is selected.
3. Check the Line Numbers checkbox.

Configuring Xcode’s Code Indentation Options

Xcode uses four space indents by default. To configure your own indentation preferences:

1. Open Xcode and select Preferences... from the Xcode menu.
2. Select the Text Editing tab, then ensure that the Indentation subtab is selected.
3. Specify your indentation preferences.

You’re now ready to begin working with iOS® 8 for Programmers: An App-Driven Approach with Swift™, Volume 1, 3/e. We hope you enjoy the book! If you have any questions, please email us at deitel@deitel.com.
This page intentionally left blank
Tip Calculator App

Introducing Swift, Text Fields, Sliders, Outlets, Actions, View Controllers, Event Handling, NSDecimalNumber, NSNumberFormatter and Automatic Reference Counting

Objectives

In this chapter you’ll:

- Learn basic Swift syntax, keywords and operators.
- Use object-oriented Swift features, including objects, classes, inheritance, functions, methods and properties.
- Use NSDecimalNumbers to perform precise monetary calculations.
- Create locale-specific currency and percentage Strings with NSNumberFormatter.
- Use Text Fields and Sliders to receive user input.
- Programmatically manipulate UI components via outlets.
- Respond to user-interface events with actions.
- Understand the basics of automatic reference counting (ARC).
- Execute an interactive iOS app.
Chapter 3  Tip Calculator App

3.1  Introduction

The Tip Calculator app (Fig. 3.1(a))—which you test-drove in Section 1.17—calculates and displays possible tips and bill totals for a restaurant bill amount. As you enter each digit of an amount by touching the numeric keypad, the app calculates and displays the tip amount and total bill amount for a 15% tip and a custom tip (Fig. 3.1(b)). You specify the custom tip percentage by moving a Slider’s thumb—this updates the custom tip percentage Labels and displays the custom tip and bill total in the righthand column of yellow Labels below the Slider (Fig. 3.1(b)). We chose 18% as the default custom percentage, because many restaurants in the U.S. add this tip percentage for parties of six people or more, but you can easily change this.

First, we’ll overview the technologies used to build the app. Next, you’ll build the app’s UI using Interface Builder. As you’ll see, Interface Builder’s visual tools can be used to connect UI components to the app’s code so that you can manipulate the corresponding UI components programmatically and respond to user interactions with them.

For this app, you’ll write Swift code that responds to user interactions and programmatically updates the UI. You’ll use Swift object-oriented programming capabilities, including objects, classes, inheritance, methods and properties, as well as various data types, operators, control statements and keywords. With our app-driven approach, we’ll present the app’s complete source code and do a detailed code walkthrough, introducing the Swift language features as we encounter them.
3.2 Technologies Overview

This section introduces the Xcode, Interface Builder and Swift features you’ll use to build the Tip Calculator app.

3.2.1 Swift Programming

Swift is Apple’s programming language of the future for iOS and OS X development. The app’s code uses Swift data types, operators, control statements and keywords, and other language features, including functions, overloaded operators, type inference, variables, constants and more. We’ll introduce Swift object-oriented programming features, including objects, classes, inheritance, methods and properties. We’ll explain each new Swift feature as we encounter it in the context of the app. Swift is based on many of today’s popular programming languages, so much of the syntax will be familiar to programmers who use C-based programming languages, such as Objective-C, Java, C# and C++. For a detailed introduction to Swift, visit:

3.2.2 Swift Apps and the Cocoa Touch® Frameworks

A great strength of iOS 8 is its rich set of prebuilt components that you can reuse rather than “reinventing the wheel.” These capabilities are grouped into iOS’s Cocoa Touch frameworks. These powerful libraries help you create apps that meet Apple’s requirements for the look-and-feel of iOS apps. The frameworks are written mainly in Objective-C (some are written in C). Apple has indicated that new frameworks will be developed in Swift.

Foundation Framework

The Foundation framework includes classes for basic types, storing data, working with text and strings, file-system access, calculating differences in dates and times, inter-app notifications and much more. In this app, you’ll use Foundation’s NSDecimalNumber and NSNumberFormatter classes. Foundation’s class names begin with the prefix NS, because this framework originated in the NextStep operating system. Throughout the book, we’ll use many Foundation framework features—for more information, visit:


UIKit Framework

Cocoa Touch’s UIKit framework includes multi-touch UI components appropriate for mobile apps, event handling (that is, responding to user interactions with the UI) and more. You’ll use many UIKit features throughout this book.

Other Cocoa Touch Frameworks

Figure 3.2 lists the Cocoa Touch frameworks. You’ll learn features from many of these frameworks in this book and in iOS 8 for Programmers: An App-Driven Approach, Volume 2. For more information on these frameworks, see the iOS Developer Library Reference (http://developer.apple.com/ios).

List of Cocoa Touch frameworks

<table>
<thead>
<tr>
<th>Cocoa Touch Layer</th>
<th>AssetsLibrary</th>
<th>AudioToolbox</th>
<th>OpenAL</th>
<th>OpenGLES</th>
<th>CoreLocation</th>
<th>CoreMedia</th>
<th>Social</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressBookUI</td>
<td>AudioUnit</td>
<td>CoreAudio</td>
<td>Photos</td>
<td>QuartzCore</td>
<td>CoreMotion</td>
<td>CoreTelephony</td>
<td>StoreKit</td>
</tr>
<tr>
<td>EventKitUI</td>
<td>CoreGraphics</td>
<td>CoreImage</td>
<td>SceneKit</td>
<td>SpriteKit</td>
<td>EventKit</td>
<td>Foundation</td>
<td>SystemConfiguration</td>
</tr>
<tr>
<td>GameKit</td>
<td>CoreMIDI</td>
<td>CoreText</td>
<td>CoreText</td>
<td>CoreVideo</td>
<td>HealthKit</td>
<td>HomeKit</td>
<td>UIAutomation</td>
</tr>
<tr>
<td>MapKit</td>
<td>GLKit</td>
<td>GameController</td>
<td>GLKit</td>
<td>GameController</td>
<td>JavaScriptCore</td>
<td>MobileCore-Services</td>
<td>WebKit</td>
</tr>
<tr>
<td>MessageUI</td>
<td>ImageIO</td>
<td>MediaController</td>
<td>ImageIO</td>
<td>MediaController</td>
<td>Accounts</td>
<td>Accounts</td>
<td>Accelerate</td>
</tr>
<tr>
<td>Notification-</td>
<td>CoreAccessibi</td>
<td>AddressBook</td>
<td>AdSupport</td>
<td>CFNetwork</td>
<td>AdSupport</td>
<td>AdSupport</td>
<td>CoreBluetooth</td>
</tr>
<tr>
<td>CenterPhotosUI</td>
<td>lity</td>
<td>AddressBook</td>
<td>AddressBook</td>
<td>CFNetwork</td>
<td>AddressBook</td>
<td>AddressBook</td>
<td>ExternalAccessory</td>
</tr>
<tr>
<td>Twitter</td>
<td>MediaAccessi</td>
<td>CloudKit</td>
<td>CFNetwork</td>
<td>CloudKit</td>
<td>CFNetwork</td>
<td>CFNetwork</td>
<td>LocalAuthentication</td>
</tr>
<tr>
<td>UIKit</td>
<td>MediaPlayer</td>
<td>CoreData</td>
<td>CoreData</td>
<td>CoreFoundation</td>
<td>CoreData</td>
<td>CoreFoundation</td>
<td>Security</td>
</tr>
<tr>
<td>iAd</td>
<td>Metal</td>
<td>CoreFoundation</td>
<td>CoreFoundation</td>
<td>CoreServices Layer</td>
<td>CoreServices Layer</td>
<td>CoreServices Layer</td>
<td>System</td>
</tr>
</tbody>
</table>

Fig. 3.2 | List of Cocoa Touch frameworks.
3.2.3 Using the UIKit and Foundation Frameworks in Swift Code

To use UIKit framework classes (or classes from any other existing framework), you must **import** the framework into each source-code file that uses it (as we do in Section 3.6.1). This exposes the framework’s capabilities so that you can access them in Swift code. In addition to UIKit framework UI components, this app also uses various classes from the Foundation framework, such as `NSDecimalNumber` and `NSNumberFormatter`. We do not **import** the Foundation framework—its features are available to your code because the UIKit framework indirectly imports the Foundation framework.

3.2.4 Creating Labels, a Text Field and a Slider with Interface Builder

You’ll again use Interface Builder and auto layout to design this app’s UI, which consists of **Labels** for displaying information, a **Slider** for selecting a custom tip percentage and a **Text Field** for receiving the user input. Several **Labels** are configured identically—we’ll show how to duplicate components in Interface Builder, so you can build UIs faster. **Labels** and the **Text Field** are objects of classes `UILabel`, `UISlider` and `UITextField`, respectively, and are part the UIKit framework that’s included with each app project you create.

3.2.5 View Controllers

Each **scene** you define is managed by a **view controller** object that determines what information is displayed. iPad apps sometimes use multiple view controllers in one scene to make better use of the larger screen size. Each scene represents a **view** that contains the UI components displayed on the screen. The view controller also specifies how user interactions with the scene are processed. Class `UIViewController` defines the basic view controller capabilities. Each view controller you create (or that’s created when you base a new app on one of Xcode’s app templates) inherits from `UIViewController` or one of its subclasses. In this app, Xcode creates the class `ViewController` to manage the app’s scene, and you’ll place additional code into that class to implement the **Tip Calculator**’s logic.

3.2.6 Linking UI Components to Your Swift Code

**Properties**

You’ll use Interface Builder to generate **properties** in your view controller for programmatically interacting with the app’s UI components. Swift classes may contain variable properties and constant properties. Variable properties are read/write and are declared with the `var` keyword. Constant properties, which cannot be modified after they’re initialized, are read-only and are declared with `let`. These keywords can also be used to declare local and global variables and constants. A variable property defines a **getter** and a **setter** that allow you to obtain and modify a property’s value, respectively. A constant property defines only a **getter** for obtaining its value.

**IBOutlet Properties**

Each property for programmatically interacting with a UI component is prefixed with `@IBOutlet`. This tells Interface Builder that the property is an **outlet**. You’ll use Interface Builder to **connect** a UI control to its corresponding outlet in the view controller using **drag-and-drop** techniques. Once connected, the view controller can manipulate the corresponding UI component programmatically. `@IBOutlet` properties are **variable** properties so they can be modified to refer to the UI controls when the storyboard creates them.
Action Methods
When you interact with a UI component (e.g., touching a Slider or entering text in a TextField), a user-interface event occurs. The view controller handles the event with an action—an event-handling method that specifies what to do when the event occurs. Each action is annotated with @IBAction in your view controller's class. @IBAction indicates to Interface Builder that a method can respond to user interactions with UI components. You'll use Interface Builder to visually connect an action to a specific user-interface event using drag-and-drop techniques.

3.2.7 Performing Tasks After a View Loads
When a user launches the Tip Calculator:
- Its main storyboard is loaded.
- The UI components are created.
- An object of the app's initial view controller class is instantiated.
- Using information stored in the storyboard, the view controller's @IBOutlet and @IBAction are connected to the appropriate UI components.

In this app, we have only one view-controller, because the app has only one scene. After all of the storyboard's objects are created, iOS calls the view controller's viewDidLoad method—here you perform view-specific tasks that can execute only after the scene's UI components exits. For example, in this app, you'll call the method becomeFirstResponder on the UITextField to make it the active component—as if the user touched it. You'll configure the UITextField such that when it's the active component, the numeric keypad is displayed in the screen's lower half. Calling becomeFirstResponder from viewDidLoad causes iOS to display the keypad immediately after the view loads. (Keypads are not displayed if a Bluetooth keyboard is connected to the device.) Calling this method also indicates that the UITextField is the first responder—the first component that will receive notification when an event occurs. iOS's responder chain defines the order in which components are notified that an event occurred. For the complete responder chain details, visit:


3.2.8 Financial Calculations with NSDecimalNumber
Financial calculations performed with Swift's Float and Double numeric types tend to be inaccurate due to rounding errors. For precise floating-point calculations, you should instead use objects of the Foundation framework class NSDecimalNumber. This class provides various methods for creating NSDecimalNumber objects and for performing arithmetic calculations with them. This app uses the class's methods to perform division, multiplication and addition.

Swift Numeric Types
Though this app's calculations use only NSDecimalNumbers, Swift has its own numeric types, which are defined in the Swift Standard Library. Figure 3.3 shows Swift's numeric and boolean types—each type name begins with a capital letter. For the integer types, each type's minimum and maximum values can be determined with its min and max properties—for example, Int.min and Int.max for type Int.
3.2 Technologies Overview

Swift also supports standard arithmetic operators for use with the numeric types in Fig. 3.3. The standard arithmetic operators are shown in Fig. 3.4.

---

### Type Description

#### Integer types
- **Int**
  - Default signed integer type—4 or 8 bytes depending on the platform.
- **Int8**
  - 8-bit (1-byte) signed integer. Values in the range –128 to 127.
- **Int16**
  - 16-bit (2-byte) signed integer. Values in the range –32,768 to 32,767.
- **Int32**
  - 32-bit (4-byte) signed integer. Values in the range –2,147,483,648 to 2,147,483,647.
- **Int64**
- **UInt8**
  - 8-bit (1-byte) unsigned integer. Values in the range 0 to 255.
- **UInt16**
  - 16-bit (2-byte) unsigned integer. Values in the range 0 to 65,535.
- **UInt32**
  - 32-bit (4-byte) unsigned integer. Values in the range 0 to 4,294,967,295.
- **UInt64**
  - 64-bit (8-byte) unsigned integer. Values in the range 0 to 18,446,744,073,709,551,615.

#### Floating-point types (conforms to IEEE 754)
- **Float**
  - 4-byte floating-point value.
  - **Negative range:** \(-3.4028234663852886e+38\) to \(-1.40129846432481707e–45\)
  - **Positive range:** \(1.40129846432481707e–45\) to \(3.4028234663852886e+38\)
- **Double**
  - 8-byte floating-point value.
  - **Negative range:** \(–1.7976931348623157e+308\) to \(–4.94065645841246544e–324\)
  - **Positive range:** \(4.94065645841246544e–324\) to \(1.7976931348623157e+308\)

#### Boolean type
- **Bool**
  - true or false values.

---

**Fig. 3.3** | Swift numeric and boolean types.

Swift also supports standard arithmetic operators for use with the numeric types in Fig. 3.3. The standard arithmetic operators are shown in Fig. 3.4.

---

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Algebraic expression</th>
<th>Swift expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>(f + 7)</td>
<td>(f + 7)</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>(p )</td>
<td>(p - c)</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>(b \cdot m)</td>
<td>(b * m)</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>(x / y) or (\frac{x}{y}) or (x + y)</td>
<td>(x / y)</td>
</tr>
<tr>
<td>Remainder</td>
<td>%</td>
<td>(r \mod s)</td>
<td>(r % s)</td>
</tr>
</tbody>
</table>

**Fig. 3.4** | Arithmetic operators in Swift.
3.2.9 Formatting Numbers as Locale-Specific Currency and Percentage Strings

You’ll use Foundation framework class `NSNumberFormatter`’s `localizedStringFromNumber` method to create locale-specific currency and percentage strings—an important part of internationalization. You could also add accessibility strings and internationalize the app using the techniques you learned in Sections 2.7–2.8.

3.2.10 Bridging Between Swift and Objective-C Types

You’ll often pass Swift objects into methods of classes written in Objective-C, such as those in the Cocoa Touch classes. Swift’s numeric types and its `String`, `Array`, and `Dictionary` types can all be used in contexts where their Objective-C equivalents are expected. Similarly, the Objective-C equivalents (`NSString`, `NSArray`, `NSMutableArray`, `NSDictionary` and `NSMutableDictionary`), when returned to your Swift code, are automatically treated as their Swift counterparts. In this app, for example, you’ll use class `NSNumberFormatter` to create locale-specific currency and percentage strings. These are returned from `NSNumberFormatter`’s methods as `NSString` objects, but are automatically treated by Swift as objects of Swift’s type `String`. This mechanism—known as bridging—is transparent to you. In fact, when you look at the Swift version of the Cocoa Touch documentation online or in Xcode, you’ll see the Swift types, not the Objective-C types for cases in which this bridging occurs.

3.2.11 Swift Operator Overloading

Swift allows operator overloading—you can define your own operators for use with existing types. In Section 3.6.7, we’ll define overloaded addition, multiplication and division operators to simplify the `NSDecimalNumber` arithmetic performed throughout the app’s logic. As you’ll see, you define an overloaded operator by creating a Swift function, but with an operator `symbol` as its name and a parameter list containing parameters that represent each operand. So, for example, you’d provide two parameters for an overloaded-operator function that defines an addition (`+`) binary operator—one for each operand.

3.2.12 Variable Initialization and Swift Optional Types

In Swift, every constant and variable you create (including a class’s properties) must be initialized (or for variables, assigned to) before it’s used in the code; otherwise, a compilation error occurs. A problem with this requirement occurs when you create `@IBOutlet` properties in a view controller using Interface Builder’s drag-and-drop techniques. Properties of such types refer to objects that are not created in your code. Rather, they’re created by the storyboard when the app executes, then the storyboard connects them to the view controller—that is, the storyboard assigns each UI component object to the appropriate property so that you can programmatically interact with that component.

For scenarios like this in which a variable receives its value at runtime, Swift provides optional types that can indicate the presence or absence of a value. A variable of an optional type can be initialized with the value `nil`, which indicates the absence of a value.

When you create an `@IBOutlet` with Interface Builder, it declares the property as an implicitly unwrapped optional type by following the type name with an exclamation point (`!`). Properties of such types are initialized by default to `nil`. Such properties must be declared
as variables (with var) so that they can eventually be assigned actual values of the specified type. Using optionals like this enables your code to compile because the @IBOutlet properties are, in fact, initialized—just not to the values they’ll have at runtime.

As you’ll see in later chapters, Swift has various language features for testing whether an optional has a value and, if so, unwrapping the value so that you can use it—known as explicit unwrapping. With implicitly unwrapped optionals (like the @IBOutlet properties), you can simply assume that they’re initialized and use them in your code. If an implicitly unwrapped optional is nil when you use it, a runtime error occurs. Also, an optional can be set to nil at any time to indicate that it no longer contains a value.

3.2.13 Value Types vs. Reference Types

Swift’s types are either value types or reference types. Swift’s numeric types, Bool type and String type are all values types.

**Value Types**
A value-type constant’s or variable’s value is copied when it’s passed to or returned from a function or method, when it’s assigned to another variable or when it’s used to initialize a constant. Note that Swift’s Strings are value types—in most other object-oriented languages (including Objective-C), Strings are reference types. Swift enables you to define your own value types as structs and enums (which we discuss in later chapters). Swift’s numeric types and String type are defined as structs. An enum is often used to define sets of named constants, but in Swift it’s much more powerful than in most C-based languages.

**Reference Types**
You’ll define a class and use several existing classes in this chapter. All class types (defined with the keyword class) are reference types—all other Swift types are value types. A constant or variable of a reference type (often called a reference) is said to refer to an object. Conceptually this means that the constant or variable stores the object’s location. Unlike Objective-C, C and C++, that location is not the actual memory address of the object, rather it’s a handle that enables you to locate the object so you can interact with it.

Both structs and enums in Swift provide many of the same capabilities as classes. In many contexts where you’d use classes in other languages, Swift idioms prefers structs or enums. We’ll say more about this later in the book.

**Reference-Type Objects That Are Assigned to Constants Are Not Constant Objects**
Initializing a constant (declared with let) with a reference-type object simply means that the constant always refers to the same object. You can still use a reference-type constant to access read/write properties and to call methods that modify the referenced object.

**Assigning References**
Reference-type objects are not copied. If you assign a reference-type variable to another variable or use it to initialize a constant, then both refer to the same object in memory.
Comparative Operators for Value Types

Conditions can be formed by using the comparative operators (==, !=, >, <, >= and <=) summarized in Fig. 3.5. These operators all have the same level of precedence and do not have associativity in Swift.

<table>
<thead>
<tr>
<th>Algebraic operator</th>
<th>Comparative operator</th>
<th>Sample condition</th>
<th>Meaning of condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>≠</td>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>≥</td>
<td>&gt;=</td>
<td>x &gt;= y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>≤</td>
<td>&lt;=</td>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
</tbody>
</table>

Fig. 3.5 | Comparative operators for value types.

Comparative Operators for Reference Types

One key difference between value types and reference types is comparing for equality and inequality. Only value-type constants and variables can be compared with the == (is equal to) and != (is not equal to) operators. In addition to the operators in Fig. 3.5, Swift also provides the === (identical to) and !== (not identical to) operators for comparing reference-type constants and variables to determine whether they refer to the same object.

3.2.14 Code Completion in the Source-Code Editor

As you type code in the source-code editor, Xcode displays code-completion suggestions (Fig. 3.6) for class names, method names, property names, and more. It provides one suggestion inline in the code (in gray) and below it displays a list of other suggestions (with the current inline one highlighted in blue). You can press Enter to select the highlighted suggestion or you can click an item from the displayed list to choose it. You can press the Esc key to close the suggestion list and press it again to reopen the list.

Fig. 3.6 | Code-completion suggestions in Xcode.
3.3 Building the App’s UI

In this section, you’ll build the Tip Calculator UI using the techniques you learned in Chapter 2. Here, we’ll show the detailed steps for building the UI—in later chapters, we’ll focus on new UI features.

3.3.1 Creating the Project

As you did in Section 2.3, begin by creating a new Single View Application iOS project. Specify the following settings in the Choose options for your new project sheet:

- **Product Name**: TipCalculator.
- **Organization Name**: Deitel and Associates, Inc.—or you can use your own organization name.
- **Company Identifier**: com.deitel—or you can use your own company identifier or use edu.self.
- **Language**: Swift.
- **Devices**: iPhone—This app is designed for iPhones and iPod touches. The app will run on iPads, but it will fill most of the screen and be centered, as in Fig. 3.7.

After specifying the settings, click **Next**, indicate where you’d like to save your project and click **Create** to create the project.

![Tip Calculator running in the iPad Air simulator.](image)

**Fig. 3.7** | Tip Calculator running in the iPad Air simulator.
Configuring the App to Support Only Portrait Orientation

In landscape orientation, the numeric keypad would obscure parts of the Tip Calculator’s UI. For this reason, this app will support only portrait orientation. In the project settings’ General tab that’s displayed in the Xcode Editor area, scroll to the Deployment Info section, then for Device Orientation ensure that only Portrait is selected. Recall from Section 2.5.1 that most iPhone apps should support portrait, landscape-left and landscape-right orientations, and most iPad apps should also support upside down orientation. You can learn more about Apple’s Human Interface Guidelines at:


3.3.2 Configuring the Size Classes for Designing a Portrait Orientation iPhone App

In Chapter 2, we designed a UI that supported both portrait and landscape orientations for any iOS device. For that purpose, we used the default size class Any for the design area’s width and height. In this section, you’ll configure the design area (also called the canvas) for a tall narrow device, such as an iPhone or iPod touch in portrait orientation. Select Main.storyboard to display the design area—a.k.a. the canvas. At the bottom of the canvas, click the Size Classes control to display the size classes tool, then click in the lower-left corner to specify the size classes Compact Width and Regular Height (Fig. 3.8).

3.3.3 Adding the UI Components

In this section, you’ll add and arrange the UI components to create the basic design. In Section 3.3.4, you’ll add auto layout constraints to complete the design.

Step 1: Adding the “Bill Amount” Label

First, you’ll add the “Bill Amount” Label to the UI:

1. Drag a Label from the Object library to the scene’s upper-left corner, using the blue guide lines to position the Label at the recommended distance from the
3.3 Building the App’s UI

scene’s top and left (Fig. 3.9). The symbol indicates that you’re adding a new component to the UI.

Step 2: Adding the Label That Displays the Formatted User Input

Next, you’ll add the blue Label that displays the formatted user input:

1. Drag another Label below the “Bill Amount” Label, such that the placement guides appear as shown in Fig. 3.10. This is where the user input will be displayed.

2. In the Attributes inspector, scroll to the View section and locate the Label’s Background attribute. Click the attribute’s value, then select Other... to display the Colors dialog. This dialog has five tabs at the top that allow you to select colors different ways. For this app, we used the Crayons tab. On the bottom row, select the Sky (blue) crayon as the color (Fig. 3.12), then set the Opacity to 50%—this allows the scene’s white background to blend with the Label’s color, resulting in a lighter blue color. The Label should now appear as shown in Fig. 3.13.
4. A Label’s default height is 21 points. We increased this Label’s height to add space above and below its text to make it more readable against the colored background. To do so, drag the bottom-center sizing handle down until the Label’s height is 30 (Fig. 3.14).

5. With the Label selected, delete the value for its Text property in the Attributes inspector. The Label should now be empty.

Step 3: Adding the “Custom Tip Percentage” Label and a Label to Display the Current Custom Tip Percentage

Next, you’ll add the Labels in the UI’s third row:
3.3 Building the App’s UI

1. Drag another Label onto the scene and position it below the blue Label as shown in Fig. 3.15.

![Fig. 3.15](image1) Adding the “Custom Tip Percentage:” Label to the scene.

2. Double click the Label and set its text to Custom Tip Percentage:.

3. Drag another Label onto the scene and position it to the right of the “Custom Tip Percentage:” Label (Fig. 3.16), then set its text to 18%—the initial custom tip percentage we chose in this app, which the app will update when the user moves the Slider’s thumb. The UI should now appear as shown in Fig. 3.17.

![Fig. 3.16](image2) Adding the Label that displays the current custom tip percentage.

![Fig. 3.17](image3) UI design so far.

**Step 4: Creating the Custom Tip Percentage Slider**

You’ll now create the Slider for selecting the custom tip percentage:

1. Drag a Slider from the Object library onto the scene so that it’s the recommended distance from the “Custom Tip Percentage:” Label, then size and position it as shown in Fig. 3.18.

![Fig. 3.18](image4)

2. Use the Attributes inspector to set the Slider’s Minimum value to 0 (the default), Maximum value to 30 and Current value to 18.
Chapter 3  Tip Calculator App

Step 5: Adding the “15%” and “18%” Labels

Next, you’ll add two more Labels containing the text 15% and 18% to serve as column headings for the calculation results. The app will update the “18%” Label when the user moves the Slider’s thumb. Initially, you’ll position these Labels approximately—later you’ll position them more precisely. Perform the following steps:

1. Drag another Label onto the scene and use the blue guides to position it the recommended distance below the Slider (Fig. 3.19), then set its Text to 15% and its Alignment to centered.

2. Next you’ll duplicate the “15%” Label, which copies all of its settings. Hold the option key and drag the “15%” Label to the right (Fig. 3.20). You can also duplicate a UI component by selecting it and typing `⌘ + D`, then moving the copy. Change the new Label’s text to 18%.
3.3 Building the App’s UI

Step 6: Creating the Labels That Display the Tips and Totals

Next, you’ll add four Labels in which the app will display the calculation results:

1. Drag a Label onto the UI until the blue guides appear as in Fig. 3.21.

Fig. 3.21 | Creating the first yellow Label.

2. Drag the Label’s bottom-center sizing handle until the Label’s Height is 30, and drag its left-center sizing handle until the Label’s Width is 156.

3. Use the Attributes inspector to clear the Text attribute, set the Alignment so the text is centered and set the Background color to Banana, which is located in the Color dialog’s Crayons tab in the second row from the bottom.

4. Set the Autoshrink property to Minimum Font Scale and change the value to .75— if the text becomes too wide to fit in the Label, this will allow the text to shrink to 75% of its original font size to accommodate more text. If you’d like the text to be able to shrink even more, you can choose a smaller value.

5. Next duplicate the yellow Label by holding the option key and dragging the Label to the left to create another Label below the “15%” Label.

6. Select both yellow Labels by holding the Shift key and clicking each Label. Hold the option key and drag any one of the selected Labels down until the blue guides appear as shown in Fig. 3.22.

Fig. 3.22 | Creating the second row of yellow Labels.
7. Now you can center the “15%” and “18%” *Labels* over their columns. Drag the “Tip” *Label* so that the blue guide lines appear as shown in Fig. 3.23. Repeat this for the “18%” *Label* to center it over the right column of yellow *Labels*.

---

**Fig. 3.23** | Repositioning the “15%” *Label*.

**Step 7: Creating the “Tip” and “Total” Labels to the Left of the Yellow Labels**

Next you’ll create the “Tip” and “Total” *Labels*:

1. Drag a *Label* onto the scene, change its *Text* to *Total*, set its *Alignment* to right aligned and position it to the left of the second row of yellow *Labels* as in Fig. 3.24.

---

**Fig. 3.24** | Positioning the “Total” *Label*.

2. Hold the *option* key and drag the “Total” *Label* up until the blue guides appear as shown in Fig. 3.25. Change the new *Label*’s text to Tip, then drag it to the right so that the right edges of the “Tip” and “Total” *Labels* align.

---

**Fig. 3.25** | Duplicating the “Total” *Label* so that you can create the “Tip” *Label*. 
3.3 Building the App’s UI

Step 8: Creating the Text Field for Receiving User Input
You’ll now create the Text Field that will receive the user input. Drag a Text Field from the Object library to the bottom edge of the scene, then use the Attributes inspector to set its Keyboard Type attribute to Number Pad and its Appearance to Dark. This Text Field will be hidden behind the numeric keypad when the app first loads. You’ll receive the user’s input through this Text Field, then format and display it in the blue Label at the top of the scene.

3.3.4 Adding the Auto Layout Constraints
You’ve now completed the Tip Calculator app’s basic UI design, but have not yet added any auto layout constraints. If you run the app in the simulator or on a device, however, you’ll notice that—depending on which simulator you use—some of the UI components extend beyond the trailing edge (Fig. 3.26). In this section, you’ll add auto layout constraints so that the UI components can adjust to display properly on devices of various sizes and resolutions.

Step 1: Adding the Missing Auto Layout Constraints
To add the missing auto layout constraints:

1. Click the white background in the design area or select View in the document outline window.

2. At the bottom of the canvas, click the Resolve Auto Layout Issues (keyboard) button and under All Views in View Controller select Add Missing Constraints.

Interface Builder analyzes the UI components in the design and based on their sizes, locations and alignment, then creates a set of auto layout constraints for you. In some cases, these constraints will be enough for your design, but you’ll often need to tweak the results. Figure 3.27 shows the UI in the iPhone 5s simulator after Interface Builder adds the missing

Fig. 3.26 | App in the iPhone 5s simulator without auto layout constraints added to the UI—some components flow off the trailing edge (the right side in this screen capture).

In Chapter 2, you manually added the required auto layout constraints. In this section, you’ll use Interface Builder to add missing constraints automatically, then run the app again to see the results. You’ll then create some additional constraints so that the app displays correctly in the simulator or on a device.

Step 1: Adding the Missing Auto Layout Constraints
To add the missing auto layout constraints:

1. Click the white background in the design area or select View in the document outline window.

2. At the bottom of the canvas, click the Resolve Auto Layout Issues (keyboard) button and under All Views in View Controller select Add Missing Constraints.

Interface Builder analyzes the UI components in the design and based on their sizes, locations and alignment, then creates a set of auto layout constraints for you. In some cases, these constraints will be enough for your design, but you’ll often need to tweak the results. Figure 3.27 shows the UI in the iPhone 5s simulator after Interface Builder adds the missing

Fig. 3.26 | App in the iPhone 5s simulator without auto layout constraints added to the UI—some components flow off the trailing edge (the right side in this screen capture).

In Chapter 2, you manually added the required auto layout constraints. In this section, you’ll use Interface Builder to add missing constraints automatically, then run the app again to see the results. You’ll then create some additional constraints so that the app displays correctly in the simulator or on a device.

Step 1: Adding the Missing Auto Layout Constraints
To add the missing auto layout constraints:

1. Click the white background in the design area or select View in the document outline window.

2. At the bottom of the canvas, click the Resolve Auto Layout Issues (keyboard) button and under All Views in View Controller select Add Missing Constraints.

Interface Builder analyzes the UI components in the design and based on their sizes, locations and alignment, then creates a set of auto layout constraints for you. In some cases, these constraints will be enough for your design, but you’ll often need to tweak the results. Figure 3.27 shows the UI in the iPhone 5s simulator after Interface Builder adds the missing

Fig. 3.26 | App in the iPhone 5s simulator without auto layout constraints added to the UI—some components flow off the trailing edge (the right side in this screen capture).

In Chapter 2, you manually added the required auto layout constraints. In this section, you’ll use Interface Builder to add missing constraints automatically, then run the app again to see the results. You’ll then create some additional constraints so that the app displays correctly in the simulator or on a device.

Step 1: Adding the Missing Auto Layout Constraints
To add the missing auto layout constraints:

1. Click the white background in the design area or select View in the document outline window.

2. At the bottom of the canvas, click the Resolve Auto Layout Issues (keyboard) button and under All Views in View Controller select Add Missing Constraints.

Interface Builder analyzes the UI components in the design and based on their sizes, locations and alignment, then creates a set of auto layout constraints for you. In some cases, these constraints will be enough for your design, but you’ll often need to tweak the results. Figure 3.27 shows the UI in the iPhone 5s simulator after Interface Builder adds the missing
constraints. Now, all of the UI components are completely visible, but some of them are not sized and positioned correctly. In particular, the yellow Labels should all be the same width.

---

**Step 2: Setting the Yellow Labels to Have Equal Widths**

To set the yellow Labels to have equal widths:

1. Select all four yellow Labels by holding the *shift* key and clicking each one.

2. In the auto layout tools at the bottom of the canvas, click the Pin tools icon ( ). Ensure that *Equal Widths* is checked and click the *Add 3 Constraints* button, as shown in Fig. 3.28. Only three constraints are added, because three of the Labels will be set to have the same width as the fourth.
3.3 Building the App’s UI

Figure 3.29 shows the UI in the simulator. Setting the yellow Labels to Equal Widths caused the 18% Label over the right column to disappear and the “Tip” and “Total” Labels to become too narrow to display.

![Fig. 3.29](image)

**Fig. 3.29** | App in the simulator after setting the yellow Labels to equal widths.

**Step 3: Debugging the Missing “18%” Label**

Based on the initial design, the missing “18%” Label should be centered over the right column of yellow Labels. If you select that Label in the canvas and select the Size inspector in the Utilities area, you can see the missing Label’s complete set of constraints (Fig. 3.30).

![Fig. 3.30](image)

**Fig. 3.30** | “18%” Label’s constraints.

There are two constraints on the “18%” Label’s horizontal positioning:

- The **Trailing Space to: Superview** constraint specifies that this Label should be 60 points from the scene’s trailing edge.
- The **Align Center X to: Label** constraint specifies that this Label should be centered horizontally over the specified Label.

These two constraints conflict with one another—depending on the yellow Label’s width, the “18%” Label could appear different distances from the scene’s trailing edge. By removing the **Trailing Space to: Superview** constraint, we can eliminate the conflict. To do so, simply click that constraint in the Size inspector and press the delete key. Figure 3.31
shows the final UI in the iPhone 5s simulator, but you can test the UI in other simulators to confirm that it works correctly in each.

3.4 Creating Outlets with Interface Builder

You’ll now use Interface Builder to create the outlets for the UI components that the app interacts with programmatically. Figure 3.32 shows the outlet names that we specified when creating this app. A common naming convention is to use the UI component’s class name without the UI class prefix at the end of an outlet property’s name—for example,
3.4 Creating Outlets with Interface Builder

billAmountLabel rather than billAmountUILabel. (At the time of this writing, Apple had not yet published their Swift coding guidelines.) Interface Builder makes it easy for you to create outlets for UI components by control dragging from the component into your source code. To do this, you’ll take advantage of the Xcode Assistant editor.

Opening the Assistant Editor
To create outlets, ensure that your scene’s storyboard is displayed by selecting it in the Project navigator. Next, select the Assistant editor button ( ) on the Xcode toolbar (or select View > Assistant Editor > Show Assistant Editor). Xcode’s Editor area splits and the file ViewController.swift (Fig. 3.33) is displayed to the right of the storyboard. By default, when viewing a storyboard, the Assistant editor shows the corresponding view controller’s source code. However, by clicking Automatic in the jump bar at the top of the Assistant editor, you can select from options for previewing the UI for different device sizes and orientations, previewing localized versions of the UI or viewing other files that you’d like to view side-by-side with the content currently displayed in the editor. The comments in lines 1–7 are autogenerated by Xcode—later, we delete these comments and replace them with our own. Delete the method didReceiveMemoryWarning in lines 18–21 as we will not use it in this app. We’ll discuss the details of ViewController.swift and add code to it in Section 3.6.

Creating an Outlet
You’ll now create an outlet for the blue Label that displays the user’s input. You need this outlet to programmatically change the Label’s text to display the input in currency format. Outlets are declared as properties of a view controller class. To create the outlet:

1. Control drag from the blue Label to below line 11 in ViewController.swift (Fig. 3.34) and release. This displays a popover for configuring the outlet (Fig. 3.35).

Fig. 3.33 | ViewController.swift displayed in the Assistant editor.

Creating an Outlet
You’ll now create an outlet for the blue Label that displays the user’s input. You need this outlet to programmatically change the Label’s text to display the input in currency format. Outlets are declared as properties of a view controller class. To create the outlet:

1. Control drag from the blue Label to below line 11 in ViewController.swift (Fig. 3.34) and release. This displays a popover for configuring the outlet (Fig. 3.35).
2. In the popover, ensure that Outlet is selected for the Connection type, specify the name billAmountLabel for the outlet’s Name and click Connect.

Xcode inserts the following property declaration in class ViewController:

```swift
@IBOutlet weak var billAmountLabel: UILabel!
```

We’ll explain this code in Section 3.6.3. You can now use this property to programmatically modify the Label’s text.

**Creating the Other Outlets**

Repeat the steps above to create outlets for the other labeled UI components in Fig. 3.32. Your code should now appear as shown in Fig. 3.36. In the gray margin to the left of each outlet property is a small bullseye (○) symbol indicating that the outlet is connected to a UI component. Hovering the mouse over that symbol highlights the connected UI component in the scene. You can use this to confirm that each outlet is connected properly.
3.5 Creating Actions with Interface Builder

Now that you've created the outlets, you need to create actions (i.e., event handlers) that can respond to the user-interface events. A Text Field's Editing Changed event occurs every time the user changes the Text Field's contents. If you connect an action to the Text Field for this event, the Text Field will send a message to the view-controller object to execute the action each time the event occurs. Similarly, the Value Changed event repeatedly occurs for a Slider as the user moves the thumb. If you connect an action method to the Slider for this event, the Slider will send a message to the view controller to execute the action each time the event occurs.

In this app, you'll create one action method that's called for each of these events. You'll connect the Text Field and the Slider to this action using the Assistant editor. To do so, perform the following steps:

1. Control drag from the Text Field in the scene to ViewController.swift between the right braces (}) at lines 25 and 26 (Fig. 3.37), then release. This displays a popover for configuring an outlet. From the Connection list in the popover, select Action to display the options for configuring an action (Fig. 3.38).
2. In the popover, specify `calculateTip` for the action’s **Name**, select **Editing Changed** for the **Event** and click **Connect**.

Xcode inserts the following empty method definition in the code:

```swift
@IBAction func calculateTip(sender: AnyObject) {
}
```

and displays a small bullseye (○) symbol (Fig. 3.39) in the gray margin to the left of the method indicating that the action is connected to a UI component. Now, when the user edits the **Text Field**, a message will be sent to the **ViewController** object to execute `calculateTip`. You’ll define the logic for this method in Section 3.6.6.

**Connecting the Slider to Method calculateTip**

Recall that `calculateTip` should also be called as the user changes the custom tip percentage. You can simply connect the **Slider** to this existing action to handle the **Slider’s Value Changed** event. To do so, select the **Slider** in the scene, then hold the `control` key and drag from the **Slider** to the **calculateTip:** method (Fig. 3.39) and release. This connects the **Slider’s Value Changed** event to the action. You’re now ready to implement the app’s logic.

### 3.6 Class ViewController

Sections 3.6.1–3.6.7 present **ViewController.swift**, which contains class **ViewController** and several global utility functions that are used throughout the class to format **NSDecimalNumber**s as currency and to perform calculations using **NSDecimalNumber** objects. We modified the autogenerated comments that Xcode inserted at the beginning of the source code file.
3.6 Class ViewController

3.6.1 import Declarations
Recall that to use features from the iOS 8 frameworks, you must import them into your Swift code. Throughout this app, we use the UIKit framework’s UI component classes. In Fig. 3.46, line 3 is an import declaration indicating that the program uses features from the UIKit framework. All import declarations must appear before any other Swift code (except comments) in your source-code files.

```swift
1 // ViewController.swift
2 // Implements the tip calculator's logic
3 import UIKit
```

Fig. 3.40 | import declaration in ViewController.swift.

3.6.2 ViewController Class Definition
In Fig. 3.41, line 5—which was generated by the IDE when you created the project—begins a class definition for class ViewController.

```swift
5 class ViewController: UIViewController {
```

Fig. 3.41 | ViewController class definition and properties.

**Keyword class and Class Names**
The class keyword introduces a class definition and is immediately followed by the class name (ViewController). Class name identifiers use camel-case naming in which each word in the identifier begins with a capital letter. Class names (and other type names) begin with an initial uppercase letter and other identifiers begin with lowercase letters. Each new class you create becomes a new type that can be used to declare variables and create objects.

**Class Body**
A left brace (at the end of line 5), {, begins the body of every class definition. A corresponding right brace (at line 82 in Fig. 3.45), }, ends each class definition. By convention, the contents of a class’s body are indented.

**Error-Prevention Tip 3.1**
A class must be defined before you use it in a given source-code file. In an Xcode project, if you define a class in one .swift file, you can use it in the project’s other source-code files—which is typical of other object-oriented languages, such as Objective-C, Java, C# and C++.

**Inheriting from Class UIViewController**
The notation : UIViewController in line 5 indicates that class ViewController inherits from class UIViewController—the UIKit framework superclass of all view controllers. Inheritance is a form of software reuse in which a new class is created by absorbing an existing class’s members and enhancing them with new or modified capabilities. This relationship indicates that a ViewController is a UIViewController. It also ensures that ViewController has the basic capabilities that iOS expects in all view controllers, including methods like
Chapter 3  Tip Calculator App

viewDidLoad (Section 3.6.5) that help iOS manage a view controller’s lifecycle. The class on
the left of the : in line 5 is the subclass (derived class) and one on the right is the superclass
(base class). Every scene has its own UIViewController subclass that defines the scene’s event
handlers and other logic. Unlike some object-oriented programming languages, Swift classes
are not required to directly or indirectly inherit from a common superclass.

3.6.3 ViewController’s @IBOutlet Properties

Figure 3.42 shows class ViewController’s nine @IBOutlet property declarations that were
created by Interface Builder when you created the outlets in Section 3.4. Typically, you’ll
define a class’s properties first followed by the class’s methods, but this is not required.

// properties for programmatically interacting with UI components
IBOutlet weak var billAmountLabel: UILabel!
IBOutlet weak var customTipPercentLabel1: UILabel!
IBOutlet weak var customTipPercentageSlider: UISlider!
IBOutlet weak var customTipPercentLabel2: UILabel!
IBOutlet weak var tip15Label: UILabel!
IBOutlet weak var total15Label: UILabel!
IBOutlet weak var tipCustomLabel: UILabel!
IBOutlet weak var totalCustomLabel: UILabel!
IBOutlet weak var inputTextField: UITextField!

Fig. 3.42  ViewController’s @IBOutlet properties.

@IBOutlet Property Declarations
The notation @IBOutlet indicates to Xcode that the property references a UI component in
the app’s storyboard. When a scene loads, the UI component objects are created, an object
of the corresponding view-controller class is created and the connections between the view
controller’s outlet properties and the UI components are established. The connection informa-
tion is stored in the storyboard. @IBOutlet properties are declared as variables using the
var keyword, so that the storyboard can assign each UI component object’s reference to the
appropriate outlet once the UI components and view controller object are created.

Automatic Reference Counting (ARC) and Property Attributes
Swift manages the memory for your app’s reference-type objects using automatic reference
counting (ARC), which keeps track of how many references there are to a given object. The
runtime can remove an object from memory only when its reference count becomes 0.

Property attributes can specify whether a class maintains an ownership or nonown-
ership relationship with the referenced object. By default, properties in Swift create strong
references to objects, indicating an ownership relationship. Every strong reference incre-
ments an object’s reference count by 1. When a strong reference no longer refers to an
object, its reference count decrements by 1. The code that manages incrementing and decre-
menting the reference counts is inserted by the Swift compiler.

The @IBOutlet properties are declared as weak references, because the view controller
does not own the UI components—the view defined by the storyboard that created them
does. A weak reference does not affect the object’s reference count. A view controller does,
however, have a strong reference to its view.
3.6 Class ViewController

Type Annotations and Implicitly Unwrapped Optional Types

A type annotation specifies a variable’s or constant’s type. Type annotations are specified by following the variable’s or constant’s identifier with a colon (:) and a type name. For example, line 7 (Fig. 3.42) indicates that billAmountLabel is a UILabel!. Recall from Section 3.2.12 that the exclamation point indicates an implicitly unwrapped optional type and that variables of such types are initialized to nil by default. This allows the class to compile, because these @IBOutlet properties are initialized—they’ll be assigned actual UI component objects once the UI is created at runtime.

3.6.4 Other ViewController Properties

Figure 3.43 shows class ViewController’s other properties, which you should add below the @IBOutlet properties. Line 18 defines the constant decimal100 that’s initialized with an NSDecimalNumber object. Identifiers for Swift constants follow the same camel-case naming conventions as variables. Class NSDecimalNumber provides many initializers—this one receives a String parameter containing the initial value ("100.0"), then returns an NSDecimalNumber representing the corresponding numeric value. We’ll use decimal100 to calculate the custom tip percentage by dividing the slider’s value by 100.0. We’ll also use it to divide the user’s input by 100.0 for placing a decimal point in the bill amount that’s displayed at the top of the app. Initializers are commonly called constructors in many other object-oriented programming languages. Line 19 defines the constant decimal15Percent that’s initialized with an NSDecimalNumber object representing the value 0.15. We’ll use this to calculate the 15% tip.

```swift
let decimal100 = NSDecimalNumber(string: "100.0")
let decimal15Percent = NSDecimalNumber(string: "0.15")
```

Fig. 3.43 | ViewController class definition and properties.

Initializer Parameter Names Are Required

When initializing an object in Swift, you must specify each parameter’s name, followed by a colon (:) and the argument value. As you type your code, Xcode displays the parameter names for initializers and methods to help you write code quickly and correctly. Required parameter names in Swift are known as external parameter names.

Type Inference

Neither constant in Fig. 3.43 was declared with a type annotation. Like many popular languages, Swift has powerful type inference capabilities and can determine a constant’s or variable’s type from its initializer value. In lines 18–19, Swift infers from the initializers that both constants are NSDecimalNumbers.

3.6.5 Overridden UIViewController method viewDidLoad

Method viewDidLoad (Fig. 3.44)—which Xcode generated when it created class ViewController—is inherited from superclass UIViewController. You typically override it to define tasks that can be performed only after the view has been initialized. You should add lines 25–26 to the method.
A method definition begins with the keyword `func` (line 22) followed by the function’s name and parameter list enclosed in required parentheses, then the function’s body enclosed in braces (`{` and `}`). The parameter list optionally contains a comma-separated list of parameters with type annotations. This function does not receive any parameters, so its parameter list is empty—you’ll see a method with parameters in Section 3.6.6. This method does not return a value, so it does not specify a return type—you’ll see how to specify return types in Section 3.6.7.

When overriding a superclass method, you declare it with keyword `override` preceding the keyword `func`, and the first statement in the method’s body typically uses the `super` keyword to invoke the superclass’s version of the method (line 23). The keyword `super` references the object of the class in which the method appears, but is used to access members inherited from the superclass.

**Displaying the Numeric Keypad When the App Begins Executing**

In this app, we want `inputTextField` to be the selected object when the app begins executing so that the numeric keypad is displayed immediately. To do this, we use property `inputTextField` to invoke the `UITextField` method `becomeFirstResponder`, which programmatically makes `inputTextField` the active component on the screen—as if the user touched it. You configured `inputTextField` such that when it’s selected, the numeric keypad is displayed, so line 26 displays this keypad when the view loads.

**3.6.6 ViewController Action Method calculateTip**

Method `calculateTip` (Fig. 3.45) is the `action` (as specified by `@IBAction` on line 31) that responds to the `TextField's Editing Changed` event and the `Slider's Value Changed` event. Add the code in lines 32–81 to the body of `calculateTip`. (If you’re entering the Swift code as you read this section, you’ll get errors on several statements that perform `NSDecimalNumber` calculations using overloaded operators that you’ll define in Section 3.6.7.)

The method takes one parameter. Each parameter’s name must be declared with a type annotation specifying the parameter’s type. When a view-controller object receives a message from a UI component, it also receives as an argument a reference to that component—the event’s `sender`. Parameter `sender`’s type—the Swift type `AnyObject`—represents any type of object and does not provide any information about the object. For this reason, the object’s type must be determined at runtime. This dynamic typing is used for actions (i.e., event handlers), because many different types of objects can generate events. In action methods that respond to events from multiple UI components, the send-
is often used to determine which UI component the user interacted with (as we do in lines 42 and 57).

```swift
@IBAction func calculateTip(sender: AnyObject)
{
    let inputString = inputTextField.text // get user input

    let sliderValue =
    NSDecimalNumber(integer: Int(customTipPercentageSlider.value))

    let customPercent = sliderValue / decimal100

    if sender is UISlider {
        // thumb moved so update the Labels with new custom percent
        customTipPercentLabel1.text =
        NSNumberFormatter.localizedStringFromNumber(customPercent,
        numberStyle: NSNumberFormatterStyle.PercentStyle)

        customTipPercentLabel2.text = customTipPercentLabel1.text
    }

    if inputString.isEmpty {
        let billAmount =
        NSDecimalNumber(string: inputString) / decimal100

        if sender is UITextField {
            // update billAmountLabel with currency-formatted total
            billAmountLabel.text = " " + formatAsCurrency(billAmount)

            // calculate and display the 15% tip and total
            let fifteenTip = billAmount * decimal15Percent
            tip15Label.text = formatAsCurrency(fifteenTip)
            total15Label.text =
            formatAsCurrency(billAmount + fifteenTip)
        }

        // calculate custom tip and display custom tip and total
        let customTip = billAmount * customPercent
        tipCustomLabel.text = formatAsCurrency(customTip)
        totalCustomLabel.text =
        formatAsCurrency(billAmount + customTip)
    }
    else {
        billAmountLabel.text = ""
        tip15Label.text = ""
    }
}

Fig. 3.45 | ViewController action method calculateTip. (Part 1 of 2.)
Chapter 3  Tip Calculator App

Getting the Current Values of `inputTextField` and `customTipPercentageSlider`

Line 32 stores the value of `inputTextField`'s `text` property—which contains the user's input—in the local `String` variable `inputString`—Swift infers type `String` because `UITextField`'s `text` property is a `String`.

Lines 35–36 get the `customTipPercentageSlider`'s `value` property, which contains a `Float` value representing the `Slider`'s thumb position (a value from 0 to 30, as specified in Section 3.3.3). The value is a `Float`, so we could get tip percentages like, 3.1, 15.245, etc. This app uses only whole-number tip percentages, so we convert the value to an `Int` before using it to initialize the `NSDecimalNumber` object that's assigned to local variable `sliderValue`. In this case, we use the `NSDecimalNumber` initializer that takes an `Int` value named `integer`.

Line 39 uses the overloaded division operator function that we define in Section 3.6.7 to divide `sliderValue` by 100 (`decimal100`). This creates an `NSDecimalNumber` representing the custom tip percentage that we'll use in later calculations and that will be displayed as a `locale-specific` percentage String showing the current custom tip percentage.

Updating the Custom Tip Percentage Labels When the Slider Value Changes

Lines 42–48 update `customTipPercentLabel1` and `customTipPercentLabel2` when the `Slider` value changes. Line 42 determines whether the `sender` is a `UISlider` object, meaning that the user interacted with the `customTipPercentageSlider`. The `is` operator returns true if an object's class is the same as, or has an `is a` (inheritance) relationship with, the class in the right operand.

We perform a similar test at line 57 to determine whether the user interacted with the `inputTextField`. Testing the `sender` argument like this enables you to perform different tasks, based on the component that caused the event.

Lines 44–46 set the `customTipPercentLabel1`'s `text` property to a `locale-specific` percentage String based on the device's current locale. `NSNumberFormatter` class method `localizedStringFromNumber` returns a `String` representation of a formatted number. The method receives two arguments:

- The first is the `NSNumber` to format. Class `NSDecimalNumber` is a subclass of `NSNumber`, so you can use an `NSDecimalNumber` anywhere that an `NSNumber` is expected.
- The second argument (which has the external parameter name `numberStyle`) is a constant from the enumeration `NSNumberFormatterStyle` that represents the formatting to apply to the number—the `PercentStyle` constant indicates that the number should be formatted as a percentage. Because the second argument must be of type `NSNumberFormatterStyle`, Swift can infer information about the
3.6 Class ViewController

method’s argument. As such, it’s possible to write the expression `NSNumberFormatterStyle.PercentStyle` with the shorthand notation:

```
NSNumberFormatterStyle.PercentStyle
```

Line 47 assigns the same String to `customTipPercentLabel2`’s text property.

**Updating the Tip and Total Labels**

Lines 51–80 update the tip and total Labels that display the calculation results. Line 51 uses the Swift String type’s `isEmpty` property to ensure that `inputString` is not empty—that is, the user entered a bill amount. If so, lines 53–72 perform the tip and total calculations and update the corresponding Labels; otherwise, the `inputTextField` is empty and lines 75–79 clear all the tip and total Labels and the `billAmountLabel` by assigning the empty String literal ("") to their text properties.

Lines 53–54 use `inputString` to initialize an `NSDecimalNumber`, then divide it by 100 to place the decimal point in the bill amount—for example, if the user enters 5632, the amount used for calculating tips and totals is 56.32.

Lines 57–66 execute only if the event’s `sender` was a `UITextField`—that is, the user tapped keypad buttons to enter or remove a digit in this app’s `inputTextField`. Line 59 displays the currency-formatted bill amount in `billAmountLabel` by calling the `formatAsCurrency` method (defined in Section 3.6.7). Line 62 calculates the 15% tip amount by using an overloaded multiplication operator function for `NSDecimalNumbers` (defined in Section 3.6.7). Then line 63 displays the currency-formatted value in the `tip15Label`. Next, lines 64–65 calculates and displays the total amount for a 15% tip by using an overloaded addition operator function for `NSDecimalNumbers` (defined in Section 3.6.7) to perform the calculation, then passing the result to the `formatAsCurrency` function. Lines 69–72 calculate and display the custom tip and total amounts based on the custom tip percentage.

**Why an External Name Is Not Required for a Method’s First Argument**

You might be wondering why we did not provide a parameter name for the first argument in the method call at lines 45–46. For method calls, Swift requires external parameter names for all parameters after the first parameter. Apple’s reasoning for this is that they want method calls to read like sentences. A method’s name should refer to the first parameter, and each subsequent parameter should have a name that’s specified as part of the method call.

### 3.6.7 Global Utility Functions Defined in ViewController.swift

Figure 3.46 contains several global utility functions used throughout class ViewController. Add lines 84–103 after the closing right brace of class ViewController.

```
84 // convert a numeric value to localized currency string
85 func formatAsCurrency(number: NSNumber) -> String {
86    return NSNumberFormatter.localizedStringFromNumber(
87        number, numberStyle: NSNumberFormatterStyle.CurrencyStyle)
88 }
```

**Fig. 3.46** ViewController.swift global utility and overloaded operator functions. (Part 1 of 2.)
Defining a Function—formatAsCurrency

Lines 85–88 define the function formatAsCurrency. Like a method definition, a function definition begins with the keyword func (line 85) followed by the function’s name and parameter list enclosed in required parentheses, then the function’s body enclosed in braces ({ and }). The primary difference between a method and a function is that a method is defined in the body of a class definition (or struct or enum definition). Function formatAsCurrency receives one parameter (number) of type NSNumber (from the Foundation framework).

A function may also specify a return type by following the parameter list with -> and the type the function returns—this function returns a String. A function that does not specify a return type does not return a value—if you prefer to be explicit, you can specify the return type Void. A function with a return type uses a return statement (line 86) to pass a result back to its caller.

We use formatAsCurrency throughout class ViewController to format NSDecimalNumbers as locale-specific currency Strings. NSDecimalNumber is a subclass of NSNumber, so any NSDecimalNumber can be passed as an argument to this function. An NSNumber parameter can also receive as an argument any Swift numeric type value—such types are automatically bridged by the runtime to type NSNumber.

Lines 86–87 invoke NSNumberFormatter class method localizedStringFromNumber, which returns a locale-specific String representation of a number. This method receives as arguments the NSNumber to format—formatAsCurrency’s number parameter—and a constant from the NSNumberFormatterStyle enum that specifies the formatting style—the constant CurrencyStyle specifies that a locale-specific currency format should be used. Once again, we could have specified the second argument as .CurrencyStyle, because Swift knows that the numberStyle parameter must be a constant from the NSNumberFormatterStyle enumeration and thus can infer the constant’s type.

Defining Overloaded Operator Functions for Adding, Subtracting and Multiplying NSDecimalNumbers

Lines 91–93, 96–98 and 101–103 create global functions that overload the addition (+), multiplication (*) and division (/) operators, respectively. Global functions (also called
free functions or just functions) are defined outside a type definition (such as a class). These functions enable us to:

- add two 
  NSDecimalNumbers
  with the + operator (lines 65 and 72 of Fig. 3.45)
- multiply two 
  NSDecimalNumbers
  with the * operator (lines 62 and 69 of Fig. 3.45)
- divide two
  NSDecimalNumbers
  with the / operator (lines 39 and 54 of Fig. 3.45)

Overloaded operator functions are defined like other global functions, but the function name is the symbol of the operator being overloaded (Fig. 3.46lines 91, 96 and 101). Each of these functions receives two 
NSDecimalNumbers
representing the operator’s left and right operands.

The addition (+) operator function (lines 91–93) returns the result of invoking

NSDecimalNumber
instance method
decimalNumberByAdding
on the left operand with the right operand as the method’s argument—this adds the operands. The multiplication (*) operator function (lines 96–98) returns the result of invoking

NSDecimalNumber
instance method
decimalNumberByMultiplyingBy
on the left operand with the right operand as the method’s argument—this multiplies the operands. The division (/) operator function (lines 101–103) returns the result of invoking

NSDecimalNumber
instance method
decimalNumberByDividingBy
on the left operand with the right operand as the method’s argument—this divides the left operand by the right operand. Since each of these
NSDecimalNumber
instance methods receives only one parameter, the parameter’s name is not required in the method call. Unlike initializers and methods, a global function’s parameter names are not external parameter names and are not required in function calls unless they’re are explicitly defined as external parameter names in the function’s definition.

### 3.7 Wrap-Up

This chapter presented the **Tip Calculator** app that calculates and displays 15% and custom tip percentage tips and totals for a restaurant bill. The app uses **Text Field** and **Slider** UI components to receive user input and update suggested tips and bill totals in response to each user interaction.

We introduced Swift—Apple’s programming language of the future—and several of its object-oriented programming capabilities, including objects, classes, inheritance, methods and properties. As you saw, the app’s code required various Swift data types, operators, control statements and keywords.

You learned about strong and weak references and that only strong references affect an object’s reference count. You also learned that iOS’s automatic reference counting (ARC) removes an object from memory only when the object’s reference count becomes 0.

You used Interface Builder to design the app’s UI visually. We showed how to build your UI faster by duplicating UI components that had similar attribute settings. You learned that **Labels** (UILabel), **Sliders** (UISlider) and **Text Fields** (UITextField) are part of iOS’s UIKit framework that’s automatically included with each app you create.

We showed how to use **import** to give your code access to features in preexisting frameworks. You learned that a scene is managed by a view-controller object that determines what information is displayed and how user interactions with the scene’s UI are processed. Our view-controller class inherited from class **UIViewController**, which defines the base capabilities required by view controllers in iOS.
You used Interface Builder to generate @IBOutlet properties (outlets) in your view controller for programmatically interacting with the app’s UI components. You used visual tools in Interface Builder to connect a UI control to a corresponding outlet in the view controller. Once a connection was made, the view controller was able to manipulate the corresponding UI component programmatically.

You saw that interacting with a UI component caused a user-interface event and sent a message from the UI component to an action (event-handling method) in the view controller. You learned that an action is declared in Swift code as an @IBAction. You used visual tools in Interface Builder to connect the action to specific user-interface events.

Next, you learned that after all the objects in a storyboard are created, iOS sends a viewDidLoad message to the corresponding view controller so that it can perform view-specific tasks that can be executed only after the UI components in the view exist. You also called the UITextField’s becomeFirstResponder method in viewDidLoad so that iOS would display this keypad immediately after the view loaded.

You used NSDecimalNumbers for precise financial calculations. You also used class NSNumberFormatter to create locale-specific currency and percentage string representations of NSDecimalNumbers. You used Swift’s operator overloading capabilities to simplify NSDecimalNumber calculations.

In the next chapter, we present the Twitter Searches app, which allows you to save your favorite (possibly lengthy) Twitter search strings with easy-to-remember short tag names. You’ll store the search strings and their short tag names in Foundation framework collections. You’ll also use iCloud key–value pair storage so that you can sync your query between all your iOS devices that have the Twitter Searches app installed.
Symbols

! for explicitly unwrapping an optional 149
!= (not equals) operator 84
!== (not identical to) operator 84
? for unwrapping a non-nil optional 149
?? (nil coalescing operator) 296
... (closed range) operator 168
..< (half-open range) operator 168, 169
{, left brace 101
}, right brace 101
* (multiplication) operator 81
/ (division) operator 81
% (remainder) operator 81
- (subtraction) operator 81
+ (addition) operator 81
< (less than) operator 84
<= (less than or equal) operator 84
== (is equal to) operator 84
=== (identical to) operator 84
> (greater than) operator 84
>= (greater than or equal to) operator 84

Numerics

100 Destinations 29

A
A8 64-bit chip 7
Accelerate framework 29
accelerometer 5, 9
accelerometer sensor 243, 250
access modifier 132
        internal 132
        private 132
        public 132
Accessibility 6, 12, 40, 46, 67
Accessibility Programming Guide for iOS 40

Accessibility (cont.)
accessibility strings 44
Accessibility Inspector 68
Large Text 7
UIAccessibility protocol 68
VoiceOver 6
White on Black 7
Zoom 7
accessories 29
Accounts framework 28
action 80
        create 99
action (event handler) 99
activity 17
Ad 315
Ad Hoc distribution 315
Ad Hoc provisioning profile 317
adaptive design 31
Add Missing Constraints 93

addition 81
addLineToPoint method of class
UIBezierPath 265
addObserver method of class
NSNotificationCenter 135
AddressBook framework 28
AddressBookUI framework 26
addTextFieldWithConfigurationHandler method of class
UIAlertController 148
admin 314, 315, 316
adopt a protocol 124
AdSupport framework 28
advertising networks

advertising networks (cont.)
mMedia 335
Mobclix 335
Nexage 335
advertising networks (cont.)
Smaato 335
Tapjoy 335
advertising revenue 325
Agent (for a development team) 315
AirDrop 15, 117
AirPrint 243, 248
AirPrint 11
Alignment attribute of a Label 91
allObjects property of class
NSSet 233
allowsRotation property of
class SKPhysicsBody 216
alpha property of a UIView 167
altimeter sensor 250
Amazon Mobile app 323
Ambient light sensor 6
Android for Programmers website xix
animated transition 205
animateWithDuration method of class
UIView 189, 190
animation xxiii, 163, 201, 281
animation frame 204
anonymous function 20, 127
AnyObject generic object type 104
AnyObject type (Swift) 122
API 25
app approval process 313
App Bundle 326
app extension 16
app icons 54
app ID 314, 315, 317
app name 49
app platforms
        Amazon Kindle 336
app platforms (cont.)
Android 336
BlackBerry 336
iPhone 336
Windows Mobile 336
App Preview 320
app record 330
app review 333
app review and recommendation sites 333
App Store xxviii, 313, 314, 316, 322, 324, 333
Books category 19
Business category 19
Catalogs category 19
Education category 19
Entertainment category 19
Finance category 19
Food and Drink category 19
Games category 19
Health and Fitness category 19
Kids category 19
Lifestyle category 19
Medical category 19
Music category 19
Navigation category 19
News category 19
Newstand category 19
Photo and Video category 19
Productivity category 19
Reference category 19
Social Networking category 19
Sports category 19
Travel category 19
Utilities category 19
Weather category 19
App Store approval 318
App Store distribution 315
App Store Marketing Guidelines 331
App Store Resource Center 314
App Store Review Guidelines 318
app templates 48
Game 48
Master-Detail Application 48, 120, 128, 129, 282
Page-Based Application 48
Single View Application 48
Tabbed Application 48
AppDelegate class 138, 288, 309
app-driven approach xxii, 2
AppKit 26
Apple Developer Program roles admin 314, 315
team member 314, 315
Apple Pay 8, 18, 28
Apple Push Notification 10
Apple Watch 8, 18
Apple World Wide Developer Conference (WWDC) 16
Application Loader 331
applicationDidEnterBackground method of the UIApplicationDelegate protocol 139
application-level events 138
applyImpulse method of class SKPhysicsBody 217
arc4random UNIX function 167
arc4random_uniform UNIX function 167
ARGB color scheme 245
arithmetic operators 81
array bounds checking 21
Array Swift Standard Library type 120
Array type (Swift) 23, 82, 120, 121, 122
element type 121
empty literal 133
filter method 140, 183, 196
removeAll method 263
removeAtIndex method 137
removeLast method 263
values property 196
arrayForKey method of class NSUserDefaults 135
as operator 135
aspect ratio 59
asset catalog 46, 54, 166, 210
AssetsLibrary framework 27
Assistant editor (Xcode) 32, 51, 97, 99
AssistiveTouch 7
attribute of a class 33
Attributes inspector 59
audio xxiii
AudioToolbox framework 27
AudioUnit framework 27
authentication 40
authorization 40
auto layout 13, 31, 45, 57, 62, 93
auto layout constraints adding 93
Equal Widths 94
equality constraint 175
missing 93
auto-image stabilization 8
automatic reference counting (ARC) 102
auto-renewable subscription 325
Autoshrink 61, 91
AV Foundation framework 27
AVAudioPlayer class 204, 211, 212
play method 204
AVFoundation framework 27
awakeFromNib method of class UIView 288
B
back button 172
Background attribute of a GUI component 87
backgroundColor property of a UIView 167
barometer sensor 8, 250
base class 102
base internationalization 70
base language (internationalization) 70, 71
becomeFirstResponder method of a GUI component 80, 104
behavior of a class 34
Beta App Review 316
beta testing 316
beta testing an app 316
blood pressure monitor 17
Bluetooth 18, 29
body of a class definition 101
Bool type 81
bounds property of a UIView 167
brand awareness 322
Index

branding apps
Amazon Mobile 323
Bank of America 323
Best Buy 323
Epicurious Recipe 323
ESPN ScoreCenter 323
ING Direct ATM Finder 323
NFL Mobile 323
Nike Training Club 323
NYTimes 323
Pocket Agent 323
Progressive Insurance 323
UPS Mobile 323
USA Today 323
Wells Fargo Mobile 323

Breakpoint navigator 50
bridging 23
bridging between Swift and Objective-C types 82, 82, 108, 121
Apple’s Using Swift with Cocoa and Objective-C guide 122
downcast 122
bullseye symbol for an outlet or action 98
bundle ID 49, 329
bundle ID search string 317
bundle seed ID 317

C
C Standard Library 167
C# xx
C++ xx
CALayer class 250
renderInContext method 266
camera 8
Camera app 11
Cannon Game app 27
CarPlay 16
categoryBitMask property of class SKPhysicsBody 207, 216, 219
center property of a UIView 167
CFNetwork framework 28
CFTimeInterval 205, 218
CGFloat struct 207
CGGeometry 207
CGFloat 207
CGPoint 207
CGPointMake 207
CGRectMake 208
CGSizeMake 208
CGPath Reference 261
CGPoint struct 207
CGPointMake function 207
CGRect struct 249
CGRectMake function 208
CGSize struct 208
CGSizeMake function 208
CGVector struct 208

Classes (cont.)
NSManagedObjectContext 280, 288, 292, 294, 296, 310
NSManagedObjectModel 280, 310
NSArray 23, 82, 121
NSMutableArray 23, 82, 121
NSMutableDictionary 23, 82, 121
NSMutableString 23
NotificationCenter 122, 306
NSDate 122
NSDecimalNumber 78, 79, 80
NSDictionary 23, 82, 121, 122
NSManagedObject 280, 291, 301
NSManagedObjectModel 280, 310
NSMutableArray 23, 82, 121
NSMutableDictionary 23, 82, 121
NSMutableString 23
NSNotificationCenter 122, 306
NSNumber 106, 108, 122
NSString 23, 82, 122
NSUserDefaultsl 122
SKAction 205
SKConstraint 206
SKLabelNode 204
SKNode 204, 215, 221
SKPhysicsBody 204, 206, 207, 215, 217
SKPhysicsWorld 204
SKScene 204, 205, 211, 212
SKShapeNode 205, 221, 222
SKSpriteNode 204, 214, 219
SKTexture 205
SKTransition 205
SKView 204, 211
UIActivityViewController 123, 124, 243
UIAlertAction 125, 146
UIAlertController 125
UIActionButton 243, 257, 272
UIBezierPath 250, 250, 261
UIDevice 143
UIImage 45, 58
UILabel 45, 79
UILongPressGestureRecognizer 125
UNavigationController 130, 165
Classes (cont.)
- UIResponder 208, 257, 260
- UISegmentedControl 165, 193
- UISlider 79
- UISplitViewController 130, 289
- UISwitch 193, 194
- UITableView 120, 151, 152
- UITableViewCellEditingStyle 154
- UITableViewController 130
- UITextField 79
- UIToolbar 243, 257, 272
- UITouch 243, 250
- UIViewController 79, 103
- UIWebView 116, 120
- Click-through rate (CTR) 325
- Closure 20, 189
- Accessing an enclosing class’s members 149
- Trailing closure 141
- Closure (anonymous function) 127, 140
- Empty parameter list 127
- Expression 127
- Fully typed 127
- Inferred types 127
- Inferred types and implicit return 128
- Operator function 128
- Shorthand argument names 128
- Cloud Kit 17
- Cloud Kit dashboard 17
- CloudKit framework 28
- Cocoa xix
- Cocoa frameworks 23, 25
- Cocoa Touch xxii, 45, 57, 78, 78
- Cocoa Touch frameworks 3, 23, 26, 78
- Accelerate 29
- Accounts 28
- AddressBook 28
- AddressBookUI 26
- AdSupport 28
- AssetsLibrary 27
- AudioToolbox 27
- Cocoa Touch frameworks (cont.)
  - AudioUnit 27
  - AVFoundation 27
  - CFNetwork 28
  - CloudKit 28
  - CoreAudio 27
  - CoreBluetooth 29
  - CoreData 28
  - CoreFoundation 28
  - CoreGraphics 27
  - CoreLocation 28
  - CoreMedia 28
  - CoreMidi 27
  - CoreMotion 28
  - CoreTelephony 28
  - CoreText 27
  - CoreVideo 27
  - EventKit 28
  - EventKitUI 26
  - ExternalAccessory 29
  - GameController 27
  - GameKit 26
  - GLKit 27
  - HealthKit 28
  - HomeKit 28
  - iAd 26
  - ImageIO 27
  - JavaScriptCore 28
  - LocalAuthentication 29
  - MapKit 26
  - MediaAccessibility 27
  - MediaPlayer 27
  - MessageUI 26
  - Metal 27
  - MobileCoreServices 28
  - MultipeerConnectivity 28
  - NewsstandKit 28
  - NotificationCenter 26
  - OpenAL 27
  - OpenGLES 27
  - PassKit 28
  - PhotosUI 26
  - PushKit 28
  - QuartzCore 27
  - QuickLook 28
  - SceneKit 27
  - Security 29
  - Social 28
  - SpriteKit 27
  - Code Snippet library 57
  - Code walkthrough 3
  - Collections
    - NSArray 121
    - NSDictionary 121
  - Collision detection 206
  - Precise 206
  - CollisionBitMask of an SKPhysicsBody 207
  - Color
    - Opacity 87
  - Company identifier 49, 49, 85, 128, 170, 209, 251, 282
  - Comparative operators 84
  - Compass 6
  - Component 33
  - ComponentsSeparatedByString method of class NSString 181
  - Computed property 136, 169, 182
  - Get accessor 182
  - Set accessor 182
  - Syntax 182
  - Conform to (implement) a protocol 35
  - Conform to a protocol 124
  - Connect a GUI control to a corresponding 79
  - Connection type 98
  - Connections inspector 59
  - Constant property 79
  - Consumables 325
  - ContactTestBitMask property of class SKPhysicsBody 207, 216, 219
context-sensitive help 51
contract information 320
convenience initializer (Swift) 216
copy and paste 9
copying an image to the clipboard 243
copy-on-write 83
Core Animation framework 249, 250
Core Animation Programming Guide 250
Core Data framework @NSManaged attribute 284
274, 280, 282, 294
Core Data Programming Guide 288
data model 280
Data Model editor 280, 282
entity 280
managed object 292
unmanaged object 292
Core Data support
Master-Detail Application template 279, 280
Single View Application template 279
Core Graphics Framework 207
Core Motion framework 250
Core Motion Framework Reference 250
CoreAudio framework 27
CoreBluetooth framework 29
CoreData framework 28
CoreFoundation framework 28
CoreGraphics framework 27
CoreLocation framework 28
CoreMedia framework 28
CoreMidi framework 27
CoreMotion framework 9, 28
CoreTelephony framework 28
CoreText framework 27
CoreVideo framework 27
cos function 224
countElements global Swift function 170, 188
CPU xxiii
crash report 328
create an action in Interface Builder 99
create an outlet in Interface Builder 97
Creating an iTunes Connect Record for an App 330
cross fade transition 205
cross-platform mobile-development tools 336
Adobe Air 337
Appcelerator 337
PhoneGap 337
QT 337
RhoMobile 337
Sencha Touch 337
cryptographic services 40
CTR (click-through rate) 325
currency format 97
currency formatting 38
CurrencyStyle constant of the NSNumberFormatterStyle enumeration 108
custom keyboard 16
cut text 9
D
Darwin module 167
data model in Core Data 275, 280, 284, 294
.xcdatamodeld filename extension 280
Data Model editor (Xcode) 280, 282
data store 280
Debug area (Xcode) 49, 51
Debug navigator 50
debugger 32
decimalNumberByAdding method of class NSDecimalNumber 109
decimalNumberByDividingBy: method of class NSDecimalNumber 109
decimalNumberByMultiplyingBy: method of class NSDecimalNumber 109
declaration
import 101
default constructor 133
defaultCenter method of class NSNotificationCenter 135
defaults system 122
defaultStore method of class NSUbiquitousKeyValueStore 135
definition
class 101
definit keyword 304
deinitializer (Swift) 304
Deitel Facebook page 332
Deitel® Buzz Online Newsletter 337
Deitel® Training 337
Delegation design pattern 125
deleteRowsAtIndexPaths method of class UITableView 154
Deployment Info 53
dequeueReusableCellWithIdentifier method of class UITableViewCell 152
derived class 102
design pattern xxiii, 126
Delegation 125
Observer 126
target-Action 125
designated initializer (Swift) 216
designing a storyboard from scratch 165
details view 120
Development Certificate 314, 315, 316
development team 314
Device Orientation 53
Devices project setting 49
Dictionary type (Swift) 120, 121, 122
empty literal 133
removeValueForKey method 137
subscripting notation 136
updateValue method 141
values property 263
Dictionary type in Swift 23, 82
dictionaryForKey method of class NSURLSession 134
didApplyConstraints method of class SKScene 206
didBeginContact method of the protocol
SKPhysicsContactDelegate 207
Index

didEndContact method of the protocol
SKPhysicsContactDelegate 207
didEvaluateActions method of class SKScene 205
didFinishUpdate method of class SKScene 206
didMoveToView method of class SKScene 213, 228
didSimulatePhysics method of class SKScene 206
digital certificate 316
Digital Crown 18
Digital Touch 18
disabilities 46, 67
dispatch_after function from the Grand Central Dispatch library 166, 190
dispatch_get_main_queue 190
dispatch_queue_t 190
DISPATCH_TIME_NOW 190
dispatch_time_t 190
distribution certificate 316
doors closing transition 205
doors opening transition 205
doorway transition 205
double tap gesture 33
drag gesture 5, 33
drawRect method of class UIView 249
drive sales 322
duplicate existing GUI components 79
dynamic prototypes (table cells) 281
dynamically typed 104
earnings 322
eedge-based physics bodies 206
eEditing Changed event for a Text Field 99, 104
equality operator 35
equality operator in Core Data 280
enum keyword 83
Equal Widths constraint 94
event handler 99
event-handling method 80
EventKit framework 9, 28
EventKitUI framework 26
Events
Editing Changed event for a Text Field 99, 104
Value Changed event for a Slider 99, 104
explicit app ID 317
explicitly unwrap an optional with ! 149
explicitly unwrapping an optional 83
extension keyword 170, 193
external parameter name 137
# to use local parameter name 126
for a function parameter 126
external parameter names 103
ExternalAccessory framework 29
Eyes Free 14

F
Facebook 13, 332
Deitel page 332, 333
Facebook integration 13
FaceTime 10, 14
factory settings 41
fade transition 205
fee-based app 19
file in the Project navigator 52
File inspector 51
File System Programming Guide 248
File Template library 57
filter method of Array 140, 183, 196
financial calculations 80
financial transaction 324
Find My iPhone 41
Find navigator 50
### Finder window
- 37

### fingerprint authentication
- 6

### first responder
- 80, 307

### Fisher-Yates shuffle
- 193

### fitness tracker
- 17

### Fix-it
- 32

### flick gesture
- 5

### flip transition
- 205

### Float type
- 80, 81

### for...in loop statement
- 168

### Foundation
- 25, 78

### Foundation Framework
- 121

### fourth-generation iPad
- 4

### frame property of a UIView
- 167, 190

### frames-per-second (FPS)
- 201, 205

### Frameworks
- Core Data 280
- Core Graphics 207
- Foundation 79
- Metal 203
- OpenGL ES 204
- SceneKit 203
- SpriteKit 203
- Store Kit 324
- UIKit 57, 79, 101

### free app
- 19, 321, 322

### Free Applications contract
- 320

### free function
- 109

### free subscription
- 325

### freemium app monetization
- model 323

### friction property of class
- SKPhysicsBody 215

### fully qualified name
- 284

### fully typed closure expression
- 127

### func keyword
- 104, 108

### function
- 109

### external parameter name
- 126

### free
- 109

### global
- 108

### with multiple return values
- 21

### Functions
- countElements 188
- join 188
- stride 169, 169
- swap 193

### Game Center
- 12, 15, 315, 317

### Game Center app
- 11

### Game Center Programming
- Guide 41

### game loop
- 201, 204, 205, 234

### Game project
- 209

### game technologies
- 203

### Game template
- 48, 203

### GameKit framework
- 12, 26

### local-player authentication
- 12

### matchmaking
- 12

### player display name
- 12

### player timeout
- 12

### games
- 39

### generic type
- 121

### generics
- 21

### gesture
- 4, 33

### double tap
- 5

### drag
- 5

### flick
- 5

### pinch
- 5

### swipe
- 5

### tap
- 5

### touch and hold
- 5

### gestures
- shake 257

### get accessor of a computed property
- 182

### getRed method of class UIColor
- 268

### Git
- 49

### Glances
- 18

### GLKit framework
- 27

### global function
- 108

### countElements 188

### join 188

### swap 193

### Google Maps
- 29

### GPS sensor
- 6, 250

### Grand Central Dispatch (GCD)
- 9, 166, 190

### dispatch_after function
- 166, 190

### dispatch_get_main_queue
- 190

### dispatch_queue_t
- 190

### dispatch_time
- 190

### DISPATCH_TIME_NOW
- 190

### DISPATCH_TIME_NOW
- 190

### Grand Central Dispatch (GCD)
- Grand Central Dispatch Reference 166

### graphics
- xiii

### graphics context
- 250, 251

### greater than or equal to
- constraint 175

### group in the Project navigator
- 52

### GUI Components

### Web View
- 120

### guide lines
- 58

### Guided Access
- 7

### gyroscope
- 9

### gyroscopic sensor
- 250

### H

### half-open range operator
- 168, 169

### Handoff
- 17

### hashtag
- 333

### HDR (High Dynamic Range)
- Photos 10

### HealthKit Framework
- 17

### HealthKit framework
- 28

### hearing impaired
- 6

### height or a GUI component
- 88

### hide status bar
- 210

### HIG (Human Interface Guidelines)
- 58, 86

### High Dynamic Range (HDR)
- Photos 10

### HomeKit framework
- 17, 28

### Human Interface Guidelines (HIG)
- 58, 86
import declaration 101
#import preprocessor directive 79
in-app advertising 321, 325
In-App Purchase 313, 315, 317, 321, 324, 325
In-App Purchase Configuration Guide for iTunes Connect 325
in keyword introduce a closure’s body 127
In-App Purchase 13
information hiding 35
inheritance 35, 101, 103
inherits 101
init keyword 134
initial 171
initial view controller 171
initializer 133, 217
convenience 216
designated 216
required 216
initializers 103
inout parameters 193
insertRowsAtIndexPaths method of class UITableView 149
insertSegmentWithTitle method of class UISegmentedControl 188
inspector 51, 58
Attributes 59
Connections 59
File 51
Identity 59
Quick Help 51
Size 59
instance 34
Instruments 32
Instruments tool xxiii
Int type 81
Int16 type 81
Int32 type 81
Int64 type 81
Int8 type 81
integerForKey method of class NSUserDefaults 180
inter-app audio 15
Interface Builder 20, 31, 44, 45
duplicate existing GUI components 79
Pin tools 94
internal access modifier 132
international App Stores 319
internationalization 46, 64, 69, 209, 237
base language 70, 71
lock your components for localization 70
Internationalization and Localization Guide 70
Internet public relations resources
ClickPress 334
i-Newswire 334
Marketwire 334
Mobility PR 335
openPR 334
PR Leap 334
Press Release Writing 335
PRLog 334
PRWeb 334
ion-strengthened glass 7
iOS 9
iOS 4.x 11
iOS 6
Social Framework 123
iOS 8 xix, 6, 16, 18
iOS 8 for Programmers website xix
iOS app templates 48
iOS defaults system 122
NSUserDefaults 122
iOS Dev Center 331
iOS Developer Enterprise Program xxviii
iOS Developer Forums 41
iOS Developer Library Reference 26, 78
iOS Developer Program xxviii, 33, 64, 313, 314, 315
iOS Developer University Program xxviii
iOS Distribution Certificate 315
iOS game technologies 203
Index

iOS Human Interface Guidelines 39, 313, 317
iOS Paid Applications contract 320
iOS Simulator 32, 44, 46, 64, 313, 314
iOS Team Provisioning Profile 315
iOS wildcard app ID 315
iPad xx, 4
iPad 2 4
iPad Air xx, 4
iPad Mini 4
iPad, first generation 4
iPad, The New 4
iPhone 3G 3
iPhone 3GS 3
iPhone 4 3, 9
iPhone 4S 3
iPhone 5 3
iPhone 5c xix, 3
iPhone 5s xix, 3
iPhone 6 xix, 4
iPhone 6 Plus xix, 4
iPhone OS 2 9
iPhone OS 3 9
iPhone sales 3
iPod touch 2
is operator 106
isEmpty property of type String 107
iSight camera 8
Issue navigator 50
iTunes 9, 325
iTunes Connect 313, 314, 327
Agreements, Tax & Banking Information 327
iAd 327
My Apps 327
Payments and Financial Reports 327
record for your app 330
Resources and Help 327
Sales and Trend Reports 327
TestFlight beta testing 316
Users and Roles 327
iTunes Connect Developer Guide 313, 316, 330
iTunes Connect Modules 327
J
Java xx
JavaScriptCore framework 28
join global Swift function 170, 188
jump bar (Assistant editor) 97
K
kCGLineCapRound 261
kCGLineJoinRound 261
key type 121
keyboard
  how to display 80, 104
keyboard shortcuts 52
Keyboard Type attribute of a TextField 93
key–value pairs 21
Keywords 318, 319
class 83, 101
definit 304
enum 83
extension 170, 193
func 104, 108
import 101
init 134
internal 132
let 79
mutating 193
nil 82
override 104
private 132
protocol 132
public 132
required 216
return 108
self 134
static 209, 226
struct 83
super 104
var 79, 102
L
Label 45, 60, 88
Alignment attribute 61, 91
Font attribute 61
Lines attribute 61
Text attribute 61
lambda 20
landscape keyboard 5, 9
landscape orientation 53, 56
language support 9
Large Text accessibility feature 7
launch image 320
launch images 54
launch screen 320
leaderboard 12
leading edge of a view 64
left brace, { 101
let keyword 79
Library window 58
light sensor 250
linearDamping property of class SKPhysicsBody 216
lineCapStyle property of class UIBezierPath 261
lineJoinStyle property of class UIBezierPath 261
LineWidth property of class UIBezierPath 261
LLVM Compiler 32
local variable 134
LocalAuthentication framework 29
locale–specific currency string 38, 108
locale–specific percentage string 106
localizable String 209
localization 69, 209, 237, 329
lock GUI components 70
localize 44
LocalizedStringFromNumber method of class NSNumberFormatter 82, 106, 108
local–player authentication 12
locate your iPhone 41
location simulation 32
locationInView method of class UITouch 265
lock your components for localization 70
All Properties 70
total storyboard 70
Localizable Properties 70
Non–localizable Properties 70
Nothing 70
loop statement
  for…in 168
  while 168
### Index

#### M
- Mac xxi
- magnetometer sensor 6, 9, 250
- Mail app 12
- main bundle 166, 181, 197
- `mainBundle` method of class `NSBundle` 181
- managed object 292
- MapKit framework 13, 26
- Maps app 13
- Marketwire 334
- mashup 29
- Master-Detail Application
  - template 48, 120, 128, 129, 275, 282
  - Core Data support 279, 280
- master-list view 120
- max property of an integer type 80
- Media library 57
- MediaAccessibility framework 15, 27
- MediaPlayer framework 27
- memory leak xiii
- memory leaks 52
- message 99
- MessageUI framework 15, 26
- Metal 48
- Metal framework 27, 203
  - method 34, 102
  - call 35
  - camel case naming 101
  - local variable 134
- micro blogging 332, 333
- microphone 9
- min property of an integer type 80
- Minimum Font Scale 61, 91
- missing auto layout constraints 93
- mobile advertising networks 325, 335
  - AdMob 335
  - Conversant 335
  - Flurry 335
  - InMobi 335
  - Inneractive 335
  - Leadbolt 335
  - Millennial Media 335
- mobileCoreServices framework 28
- Mode attribute 59
- Model-View-Controller (MVC) design pattern xiii, 36, 123
- moisture sensor 250
- monetary values 38
- monetizing apps 313, 322, 324, 325
- motion data
  - `motionEnded` method of class `UIResponder` 250, 257, 260
  - `moveInTransition` method of class `UIBezierPath` 265
- multimedia xiii
- Multimedia Programming Guide 204
- MultipeerConnectivity framework 15, 28
- multipleTouchEnabled
  - property of class `UIView` 262
- multiplication 81
- multitasking 9
- multi-touch gestures 7
- multi-touch GUI components 26
- mutating keyword 193
- MVC (Model-View-Controller) 36
- MVC (Model-view-controller) xiii

#### N
- namespace 284
- naming convention
  - GUI components 96
- Navigation Controller 171
- navigation controller
  - back button 172
  - root view controller 171
- Navigator area (Xcode) 49, 50, 51
- Navigators 50
  - Breakpoint 50
  - Debug 50
  - Issue 50
  - Log 50
  - Project 50, 52
  - Search 50
- Symbol 50
- near-field communication (NFC) 8
- nested functions 22
- nested types 22
- network activity xiii
- networkActivityIndicatorVisible property of class `UIApplication` 156
- Newsstand app 11
- Newsstand Kit 11
- NewsstandKit framework 28
- NeXTSTEP operating system 20, 78
- NFC sensor 6
- nil 296
- `nil` coalescing operator ?? 296
- `nil` keyword 82
- nonconsumables 325
- non-deterministic random numbers 167
- not identical to (!==) operator 84
-NotificationCenter 11
- NotificationCenter framework 26
- notifications 122
- `NSNotificationCenter` 122
- register to receive 135
- NSArray class 23, 82, 121, 122
- `NSBundle` class 166, 181
- `mainBundle` method 181
- `pathsForResource` method 212
- `pathsForResource ofType` method 181
- `NSData` class 122
- `NSDate` class 122
- `NSDecimalNumber` class 78, 79, 80
  - `decimalNumberByAdding` method 109
overloaded division operator 106
override keyword 104
Overview of iTunes Connect 330

P
Page-Based Application template 48
paid app 321
parameter inout 193
type annotation 134
Pass Kit 13
Passbook app 8, 13, 14
pass-by-reference 268
passes 13
PassKit framework 28
paste text 9
pathForResource method of class NSBundle 212
pathsForResourceOfType method of class NSBundle 181
payment 325
PC free device activation and iOS updates 11
peer-to-peer games 9
PercentStyle constant 106
performance issues 52
persistent data store 280
PhoneGap 336
photo sharing 14, 332
Photos framework 17
PhotosUI framework 17, 26
physics 216
physics attributes 204
physics engine 201
physics simulation 205, 206
Pin tools in Interface Builder 94
pinch gesture 5, 33
pixel density 54
placeholder property of a UITextField 148
play method of class AVAudioPlayer 204
playground 31
PNG image 166
portrait orientation 53, 56
PR Leap 334
precise collision detection 206
Preferences and Settings Programming Guide 122
preferredDisplayMode property of class UISplitViewController 309
prefersStatusBarHidden method of class UIViewController 213
prepareForSegue method of class UIViewController 150, 192, 258
presentScene method of class SKScene 213
press release writing 335
price 19, 322
price tier 329
Pricing Matrix 329
pricing your app 321
principle of least privilege 40
printing with AirPrint 243
privacy 41
private access modifier 132
PRLog 334
processor 23
Programmableweb 29
programmatically select a component 80, 104
programming languages Objective-C 23
project 47
project name 49
Project navigator 50, 52
Project Structure group 52
promotional code 327
property 35, 102
computed 79, 169
constant 79
variable 79
property attribute 102
weak 102
property declaration 98
Protocol UISplitViewControllerDelegate 309
protocol 35, 124, 178
adopt 124
conform to 35, 124
similar to an interface in other programming languages 35, 124
protocol keyword 132
Protocols
    NSFetchedResultsControllerDelegate 280, 288, 297
    SKPhysicsContactDelegate 207, 219, 226
    SKSceneDelegate 206
    UITableViewDataSource 151
    UITableViewDelegate 281, 294
    UIWebViewDelegate 156
    protocols
        UIApplicationDelegate 139, 309
    prototype cell
        Reuse Identifier 130
    prototype cell of a UITableView 130
    provision 315
    Provisioning Profile 314, 315
    proximity sensor 6, 250
public access modifier 132
public relations 334
purchasing interface 325
Push Notification 10
push notifications 315, 317
push transition 205
PushKit framework 28

Q
Quartz 249
Quartz Core framework 27, 250
Quick Help inspector 51
QuickLook framework 28

R
random numbers
    arc4random UNIX function 167
    arc4random_uniform UNIX function 167
rating apps 328
rawValue of an enum constant 218
Read-Eval-Print-Loop (REPL) 31
Index 351

Receipt Validation Programming Guide 325
recent projects 47
record for your app in iTunes Connect 330
refer to an object 83
reference 83
reference count 102
reference type 83
references type 209
register to receive notifications 135
reinventing the wheel 78
relational database 280
relationship segue 171
release date 328
remainder operator, % 81
Reminders app 11, 13
Remote Wipe 41
removeAll method of Array 263
removeAllSegments method of class UISegmentedControl 186
removeAtIndexPath method of type Array 137
removeLast method of Array 263
removeObjectForKey method of class NSUbiquitousKeyValueStore 137
removeValueForKey method of type Dictionary 137
render a sprite 205
renderInContext method of class CALayer 266
rendering loop 201
REPL (Read-Eval-Print-Loop) 31
Report navigator 50
required initializer (Swift) 217
required keyword 216
Resolve Auto Layout Issues 93
responder chain 80, 307
restitution property of class SKPhysicsBody 215
Retina HD display 7
return keyword 108
return type of a method or function 108
reusable software components 33
reuse 34, 78
reuse identifier for a UITableView cell 152
Reuse Identifier for a UITableView prototype cell 130
reveal transition 205
review and recommendation sites 333
RGB 245
Rhapsody 25
right brace, } 101
risk assessment 40
root SKNode 204
root view controller 165, 171
rotate 33
rotateToAngle method of class SKAction 224
routing app 13
row property of an NSIndexPath 152
Run button (Xcode) 38
runAction method of class SKNode 224
S
Safari app 11
sandboxing 40
save method of class NSManagedObjectContext 293
saving an image 243
scene 56
SceneKit 48
SceneKit framework 17, 27, 203
Scheme selector (Xcode) 38
SCM (source-code management) repository 47
scope 284
screen 318, 320, 329
search operators (Twitter) 112
security 8, 40
Security framework 29
security system 17
segue (storyboard) 150, 165, 172, 193
segue in a storyboard 131
segue popover 172, 254, 256
select a component programmatically 80, 104
selectedSegmentIndex property of class UISegmentedControl 194
selecting multiple GUI components 91
selector 125, 135
@selector attribute 135
Selector type 135
selectRowAtIndexPath method of class UITableView 188
self keyword 134
send a message to an object 35
sender of an event 104
sensor 5
accelerometer 5, 243
Ambient light sensor 6
compass 6
GPS 6
magnetic sensor 6
proximity sensor 6
three-axis gyro 5
sensors
accelerometer 250
altimeter 250
barometer 250
GPS 250
gyroscope 250
light 250
magnetometer 250
moisture 250
proximity 250
set accessor of a computed property 182
setObject method of class NSUbiquitousKeyValueStore 141
setObject method of class NSUserDefaults 138
setStroke method of class UIColor 262
shadow a property 134
shake gesture 5, 257
sharing an image 243
sharing options 118
sheet 47, 49
shouldAutorotate method of class UIViewController 213
shuffle
Fisher-Yates 193
simple touch events 208
simulator 44, 46, 64
sin function 224
Sina Weibo 13
Single View Application template 48, 85, 170, 251
Core Data support 279
Siri 12, 18
Eyes Free 14
size class 56
Any 86
Compact Width 86
Regular Height 86
Size inspector 59, 95
SKAction class 205
rotateToAngle method 224
SKColor class 214
SKConstraint class 206
SKLabelNode class 204
SKNode class 204, 215, 221
runAction method 224
SKPhysicsBody class 204, 206, 207, 215, 217
allowsRotation property 216
applyImpulse method 217
categoryBitMask property 207, 216, 219
collisionBitMask 207
contactTestBitMask property 207, 216, 219
friction property 215
linearDamping property 216
restitution property 215
usePreciseCollisionDetection property 216
SKPhysicsContactDelegate protocol 207, 219, 226
didBeginContact method 207
didEndContact method 207
SKPhysicsWorld class 204
SKScene class 204, 205, 211, 212
didApplyConstraints method 206
didEvaluateActions method 205
SKScene class (cont.)
didFinishUpdate method 206
didMoveToView method 213
didSimulatePhysics method 206
ignoresSiblingOrder property 213
presentScene method 213
touchesBegan method 237
update method 205
SKSceneDelegate protocol 206
SKSceneScaleMode 212
SKShapeNode class 205, 221, 222
SKSpriteNode class 204, 214, 219
initializer 214
SKTexture class 205, 214
SKTransition class 205
SKView class 204, 211
Slider 37, 76
thumb 37, 76, 89, 90
thumb position 106
Value Changed event 99, 104
SMS 117
Social Framework 117, 123
Social framework 13, 28
Social Framework Reference 41, 123
social media sites 332
social networking 332
sound 210
source code 3
source-code control system 49
source-code management (SCM) repository 47
sprite 201, 210
SpriteKit 17, 48, 210
SKAction 205
SKConstraint 206
SKLabelNode 204
SKNode 204, 215, 221
SKPhysicsBody 204, 206, 207, 215, 217
SKPhysicsContactDelegate 207, 219, 226
SKPhysicsWorld 204
SKScene 204, 205, 211, 212
SKSceneDelegate 206
SKShapeNode 205, 221, 222
SpriteKit (cont.)
SKSpriteNode 204, 214, 219
SKTexture 205
SKTransition 205
SKView 204, 211
SpriteKit framework 15, 27, 199, 203, 204
SQLite 280, 310
Standard editor (Xcode) 50
standardUserDefaults method of class NSUserDefaults 134
State Preservation 13
Statements
for...in 168
while 168
static cells in a
UITableView 281, 286
static keyword 209, 226
status bar 210
hide 210
StepStone 20
stopLoading method of class
UIWebView 156
Store Kit framework 324, 325
Store Kit Framework Reference 325
Store Kit Programming Guide 325
stored property 136, 169
StoreKit framework 11, 15, 29
storyboard 31, 56
design from scratch 165
segue 165, 193
storyboarding 45
stride global function 169, 169
closed-range 169
half-open range 169
String interpolation 22
String type 107
isEmpty 107
String type (Swift) 122
String type in Swift 23, 82
stringByAddingPercentEncodingWithAllowedCharacters method of class NSString 151
stroke method of class
UIBezierPath 250, 261, 262, 263
strong reference 102
struct keyword 83
subclass 35, 102, 217
subject object 126
submitting apps for approval 330
Submitting Your App 331
subscription 325
super keyword 104
subclass 35, 102
subject object 126
submitting apps for approval 330
Submitting Your App 331
subscription 325
superclass 35, 102
supportedInterfaceOrientations method of class
UIViewController 213
swap global function 169, 193
Swift xxi, 3, 20, 23, 76
   AnyObject type 122
   Apple publications 24
   Array type 120, 121, 122
   as operator 135
   Blog 24
   convenience initializer 216
deinitializer 304
designated initializer 216
Dictionary type 120, 121, 122
numeric types 122
operator overloading 82
pass-by-reference 268
required initializer 216, 217
sample code 24
static keyword 209, 226
String type 122
Swift Programming Language
iBook 24
Swift Standard Library 23
Swift Standard Library Reference 23
type constant 209, 226
type variable 209, 213
Swift for Programmers
(www.deitel.com/books/swiftfp/) xxi
Swift global function
countElements 188
join 188
swap 193
Swift programming language
xii
Swift Programming Language
book (Apple) 209
Swift Standard Library 80
countElements global function 170
join global function 170
swap global function 169
Swift types
   Array 23, 82
dictionary 23, 82
numeric 23, 80, 82
string 23, 82
swap gesture 5, 33
Symbol navigator 50
synchronize method of class
NSSubiquitousKeyValueStore 136
synchronize method of class
NSUserDefaults 138
syntax coloring 3
System framework 29
SystemConfiguration framework 29
T
   tab bar 48
   Tabbed Application template 48
tableView method of protocol
UITableViewDataSource for getting the cell at a given
index 152
tableView method of protocol
UITableViewDataSource for responding to an edit 154
tableView method of protocol
UITableViewDataSource for the number of rows in a
section 152
tableView method of protocol
UITableViewDataSource that determines whether a cell is
editable 153
tableView method of protocol
UITableViewDataSource that determines whether a cell is
movable 154
tap gesture 5, 33
Taptic Engine 18
Target-Action design pattern 125
target-language attribute
(XLIFF) 72, 238
Team Agent 320
team member 314, 315, 316
Technical Support Incident
(TSI) 314
template 48
Test navigator 50
TestFlight beta testing 316
TestFlight FAQ 316
Text Field 86, 87
   Editing Changed event 99, 104
   Keyboard Type attribute 93
Text property 88
   Text property of a Label 88
text property of a UILabel 106, 107
text property of a UITextField 106
textFieldShouldReturn method of
protocol UITextFieldDelegate 307
thermostat 17
thread safe UI 166
threat modeling 40
three-axis gyro 5
thumb of a Slider 37, 76, 89, 90
thumb position of a Slider 106
titleForSegmentAtIndex method of class
UISegmentedControl 188
touch and hold gesture 5, 33
touch event 250
touch events
   simple 208
Touch ID Authentication 17
Touch ID sensor 6
touchesBegan method of an
SKScene 237
touchesBegan method of class
UIResponder 208, 250, 265
touchesCannelled method of class
UIResponder 250, 266
touchesEnded method of class
UIResponder 250, 265
touchesMoved method of class
UIResponder 250, 265
TouchID 8
   trailing closure syntax 141
   trailing edge of a scene 93
   trailing edge of a view 64
transform property of a UIView
transition
door opening 237
TSI (Technical Support Incident) 314
tuple 21
tweet 333
Twitter 13, 29, 333
@deitel 333
hashtag 333
tweet 333
Twitter account API 11
Twitter framework 13, 26
Twitter integration 11
Twitter search 112
operators 114
Twitter Searches app 13, 28
two-finger drag 33
type annotation 103, 121
parameter 134
type constant 209, 226
type inference 20, 103
type safe 121
type variable 209, 213
Types
Bool 81
Double 81
Float 81
Int 81
Int16 81
Int32 81
Int64 81
Int8 81
max property of each integer type 80
min property of each integer type 80
UInt16 81
UInt32 81
UInt64 81
UInt8 81
types
Double 80
Float 80
Void 108

U
UI components are not thread safe 166
UIAccessibility protocol 68
UIActivityViewController class 123, 124, 243
UIAlertAction class 125, 146
UIAlertController class 125
addTextFieldWithConfigurationHandler method 148
UIApplication class
networkActivityIndicatorVisible property 156
UIAutomation framework 29
UITapGestureRecognizer class
addLineToPoint method 265
tiangleCapStyle property 261
iangleJoinStyle property 261
iangleWidth property 261
moveToPoint method 250, 261, 262, 263
UIBezierPath class
setStroke method 262
getRed method 268
UIControl class
userInterfaceIdiom property 143
UIKit framework 26, 26, 57, 79, 101
UILabel 79
UISlider 79
UITextField 79
UIViewController 79
UIKit functions 251
UIGraphicsBeginImageContextWithOptions function 266
UIGraphicsGetCurrentContext function 267
UIGraphicsGetImageFromCurrentContext function 267
UIKit graphics system 249
UILabel class 45, 79
text property 106, 107
UILongPressGestureRecognizer class 125
INavigationController class 130, 165, 171
root view controller 165
UInt16 type 81
UInt32 type 81
UInt64 type 81
UInt8 type 81
UIResponder class 208, 257, 260
motionEnded method 250, 257, 260
touchesBegan method 208, 250, 265
touchesCancelled method 250, 266
touchesEnded method 250, 265
touchesMoved method 250, 265
UISegmentedControl class 165, 193
insertSegmentWithTitle method 188
removeAllSegments method 186
selectedSegmentIndex property 194
titleForSegmentAtIndex method 188
UITextField class 79
value property 106
Index

video 320
video sharing 332
view animation 159, 163, 167, 189, 190
view controller 79
    initial 171
view debugger 32
viewDidAppear method of class UIViewController 156
viewDidLoad message 80, 143, 156
viewWillAppear method of class UIViewController 289
viewWillDisappear method of class UIViewController 156, 304
viral marketing 332
viral video 333
virtual goods 324, 324
vision impaired 6
VoiceOver 6, 46, 67, 69
    enable/disable 67
Void type 108
Volume Purchase Program (VPP) 326
volume-based physics bodies 206
VPP (Volume Purchase Program) 326

W
WatchKit 18
weak property attribute 102
web services 29
    Amazon eCommerce 30
eBay 30
Facebook 30
Flickr 30
Foursquare 30
Google Maps 30
Groupon 30
Instagram 30
Last.fm 30
LinkedIn 30
Microsoft Bing 30
Netflix 30
PayPal 30
Salesforce.com 30
Skype 30
Twitter 30

X
    .xcdatamodeld filename extension 280
Xcode xix, 3, 37, 45
    Assistant editor 51, 97, 99
code-completion suggestions 84
    Data Model editor 280
    Debug area 49, 51
    Editor area 49, 50
    Game project 209
    Navigator area 49, 50, 51
    Single View Application project 85, 170, 251
    Standard editor 50
    Utilities area 49, 51
    Version editor 51
Xcode 6 xxiii, 31
Xcode Groups
    project structure 52
Xcode IDE 44

Xcode Libraries
    Code Snippet 57
    File Template 57
    Media 57
    Object 57
Xcode navigators
    Breakpoint 50
    Debug 50
    Find 50
    Issue 50
    Project 50, 52
    Report 50
    Symbol 50
    Test 50
Xcode templates
    Game 203
Xcode toolbar 51
Xcode Windows Library 58
    Welcome to Xcode 47
XCTest 32
XLIFF
    XML Localization Interchange File Format 70, 71, 238
XML Localization Interchange File Format (XLIFF) 70, 71, 238

Y
    Yellow Box API 25

Z
    Zoom accessibility feature 7