

iOS 8 App Development



Essentials

iOS 8 App Development Essentials

iOS 8 App Development Essentials – Second Edition

© 2015 Neil Smyth/eBookFrenzy. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev 2.0



Table of Contents

1. Start Here	1
1.1 For New iOS Developers	1
1.2 For iOS 7 Developers	1
1.3 Source Code Download	2
1.4 Feedback.....	2
1.5 Errata	3
2. Joining the Apple iOS Developer Program	5
2.1 Registered Apple Developer	5
2.2 Downloading Xcode 6 and the iOS 8 SDK	5
2.3 iOS Developer Program	5
2.4 When to Enroll in the iOS Developer Program?	6
2.5 Enrolling in the iOS Developer Program	6
2.6 Summary.....	7
3. Installing Xcode 6 and the iOS 8 SDK.....	9
3.1 Identifying if you have an Intel or PowerPC based Mac	9
3.2 Installing Xcode 6 and the iOS 8 SDK	10
3.3 Starting Xcode.....	10
4. A Guided Tour of Xcode 6	11
4.1 Starting Xcode 6.....	11
4.2 Creating the iOS App User Interface	15
4.3 Changing Component Properties.....	17
4.4 Adding Objects to the User Interface	17
4.5 Building and Running an iOS 8 App in Xcode 6	20
4.6 Dealing with Build Errors	21
4.7 Monitoring Application Performance	21
4.8 An Exploded View of the User Interface Layout Hierarchy	22
4.9 Summary.....	23
5. Testing Apps on iOS 8 Devices with Xcode 6	25
5.1 Configuring Xcode with Apple IDs	25
5.2 Generating Signing Identities	26
5.3 Device Registration	28
5.4 Manually Adding Test Devices	29
5.5 Running an Application on a Registered Device	30
5.6 Summary.....	31
6. An Introduction to Swift Playgrounds	33

6.1 What is a Swift Playground?	33
6.2 Creating a New Swift Playground	33
6.3 A Basic Swift Playground Example	34
6.4 Playground Timelines	35
6.5 Working with UIKit in Playgrounds	37
6.6 When to Use Swift Playgrounds	38
6.7 Summary.....	38
7. Swift Data Types, Constants and Variables	41
7.1 Using a Swift Playground	41
7.2 Swift Data Types	42
7.2.1 Integer Data Types	43
7.2.2 Floating Point Data Types	43
7.2.3 Bool Data Type	43
7.2.4 Character Data Type.....	43
7.2.5 String Data Type.....	44
7.2.6 Special Characters/Escape Sequences.....	44
7.3 Swift Variables	45
7.4 Swift Constants	45
7.5 Declaring Constants and Variables	45
7.6 Type Annotations and Type Inference.....	46
7.7 The Swift Tuple	47
7.8 The Swift Optional Type	47
7.9 Type Casting and Type Checking	51
7.10 Summary.....	53
8. Swift Operators and Expressions	55
8.1 Expression Syntax in Swift	55
8.2 The Basic Assignment Operator.....	55
8.3 Swift Arithmetic Operators.....	55
8.4 Compound Assignment Operators	56
8.5 Increment and Decrement Operators	57
8.6 Comparison Operators	57
8.7 Boolean Logical Operators.....	58
8.8 Range Operators.....	59
8.9 The Ternary Operator	59
8.10 Bitwise Operators	60
8.10.1 Bitwise NOT.....	60
8.10.2 Bitwise AND.....	60
8.10.3 Bitwise OR	61
8.10.4 Bitwise XOR	61
8.10.5 Bitwise Left Shift.....	62

8.10.6 Bitwise Right Shift	62
8.11 Compound Bitwise Operators	63
8.12 Summary.....	63
9. Swift Flow Control	65
9.1 Looping Flow Control.....	65
9.2 The Swift for Statement.....	65
9.2.1 The Condition-Increment for Loop.....	65
9.2.2 The for-in Loop	66
9.2.3 The while Loop.....	67
9.3 The do ... while loop	68
9.4 Breaking from Loops.....	68
9.5 The continue Statement	69
9.6 Conditional Flow Control	69
9.7 Using the if Statement	69
9.8 Using if ... else ... Statements.....	70
9.9 Using if ... else if ... Statements.....	71
9.10 Summary.....	71
10. The Swift Switch Statement.....	73
10.1 Why Use a switch Statement?	73
10.2 Using the switch Statement Syntax	73
10.3 A Swift switch Statement Example	74
10.4 Combining case Statements	74
10.5 Range Matching in a switch Statement	75
10.6 Using the where statement	75
10.7 Fallthrough.....	76
10.8 Summary.....	77
11. An Overview of Swift Functions and Closures	79
11.1 What is a Function?	79
11.2 How to Declare a Swift Function	79
11.3 Calling a Swift Function.....	80
11.4 Declaring External Parameter Names.....	80
11.5 Declaring Default Function Parameters.....	81
11.6 Returning Multiple Results from a Function.....	82
11.7 Variable Numbers of Function Parameters	82
11.8 Parameters as Variables	83
11.9 Working with In-Out Parameters.....	83
11.10 Functions as Parameters.....	84
11.11 Closure Expressions	86
11.12 Closures in Swift	87

11.13 Summary.....	88
12. The Basics of Object Oriented Programming in Swift	89
12.1 What is an Object?.....	89
12.2 What is a Class?	89
12.3 Declaring a Swift Class	89
12.4 Adding Instance Properties to a Class.....	90
12.5 Defining Methods	90
12.6 Declaring and Initializing a Class Instance	91
12.7 Initializing and Deinitializing a Class Instance	92
12.8 Calling Methods and Accessing Properties	93
12.9 Stored and Computed Properties	94
12.10 Using self in Swift.....	95
12.11 Summary.....	96
13. An Introduction to Swift Inheritance	97
13.1 Inheritance, Classes and Subclasses	97
13.2 A Swift Inheritance Example	97
13.3 Extending the Functionality of a Subclass	98
13.4 Overriding Inherited Methods.....	99
13.5 Initializing the Subclass.....	100
13.6 Using the SavingsAccount Class.....	100
13.7 Summary.....	101
14. Working with Array and Dictionary Collections in Swift.....	103
14.1 Mutable and Immutable Collections	103
14.2 Swift Array Initialization.....	103
14.3 Working with Arrays in Swift	104
14.3.1 Array Item Count	104
14.3.2 Accessing Array Items	105
14.4 Appending Items to an Array.....	105
14.4.1 Inserting and Deleting Array Items.....	105
14.4.2 Array Iteration	105
14.5 Swift Dictionary Collections	106
14.6 Swift Dictionary Initialization.....	106
14.6.1 Dictionary Item Count	107
14.6.2 Accessing and Updating Dictionary Items.....	107
14.6.3 Adding and Removing Dictionary Entries	107
14.6.4 Dictionary Iteration	108
14.7 Summary.....	108
15. The iOS 8 Application and Development Architecture	109

15.1 An Overview of the iOS 8 Operating System Architecture	109
15.2 Model View Controller (MVC)	110
15.3 The Target-Action pattern, IBOutlets and IBActions	111
15.4 Subclassing.....	111
15.5 Delegation.....	111
15.6 Summary.....	112
16. Creating an Interactive iOS 8 App	113
16.1 Creating the New Project.....	113
16.2 Creating the User Interface	113
16.3 Building and Running the Sample Application.....	116
16.4 Adding Actions and Outlets	116
16.5 Building and Running the Finished Application	121
16.6 Hiding the Keyboard	121
16.7 Summary.....	122
17. Understanding iOS 8 Views, Windows and the View Hierarchy	123
17.1 An Overview of Views.....	123
17.2 The UIWindow Class	123
17.3 The View Hierarchy.....	124
17.4 View Types.....	125
17.4.1 The Window	125
17.4.2 Container Views.....	126
17.4.3 Controls	126
17.4.4 Display Views.....	126
17.4.5 Text and Web Views	126
17.4.6 Navigation Views and Tab Bars.....	126
17.4.7 Alert Views	126
17.5 Summary.....	126
18. An Introduction to Auto Layout in iOS 8	127
18.1 An Overview of Auto Layout.....	127
18.2 Alignment Rects.....	128
18.3 Intrinsic Content Size	129
18.4 Content Hugging and Compression Resistance Priorities.....	129
18.5 Three Ways to Create Constraints.....	129
18.6 Constraints in more Detail.....	129
18.7 Summary.....	130
19. Working with iOS 8 Auto Layout Constraints in Interface Builder	133
19.1 A Simple Example of Auto Layout in Action	133
19.2 Enabling and Disabling Auto Layout in Interface Builder.....	133

19.3 The Auto Layout Features of Interface Builder	139
19.3.1 Suggested Constraints	139
19.3.2 Visual Cues	140
19.3.3 Highlighting Constraint Problems	141
19.3.4 Viewing, Editing and Deleting Constraints	143
19.4 Creating New Constraints in Interface Builder	146
19.5 Adding Aspect Ratio Constraints	146
19.6 Resolving Auto Layout Problems	146
19.7 Summary.....	148
20. An iOS 8 Auto Layout Example.....	149
20.1 Preparing the Project.....	149
20.2 Designing the User Interface	149
20.3 Adding Auto Layout Constraints	151
20.4 Adjusting Constraint Priorities.....	152
20.5 Testing the Application	154
20.6 Summary.....	155
21. Implementing iOS 8 Auto Layout Constraints in Code	157
21.1 Creating Constraints in Code	157
21.2 Adding a Constraint to a View	158
21.3 Turning off Auto Resizing Translation.....	159
21.4 An Example Application	160
21.5 Creating the Views.....	160
21.6 Creating and Adding the Constraints.....	160
21.7 Removing Constraints.....	162
21.8 Summary.....	163
22. Implementing Cross-Hierarchy Auto Layout Constraints in iOS 8	165
22.1 The Example Application	165
22.2 Establishing Outlets	166
22.3 Writing the Code to Remove the Old Constraint.....	167
22.4 Adding the Cross Hierarchy Constraint.....	168
22.5 Testing the Application	168
22.6 Summary.....	168
23. Understanding the iOS 8 Auto Layout Visual Format Language	169
23.1 Introducing the Visual Format Language.....	169
23.2 Visual Language Format Examples	169
23.3 Using the constraintsWithVisualFormat Method	170
23.4 Summary.....	172
24. Using Size Classes to Design Universal iOS User Interfaces	173

24.1 Understanding Size Classes	173
24.2 Size Classes in Interface Builder.....	173
24.3 Setting “Any” Defaults	174
24.4 Working with Size Classes in Interface Builder	174
24.5 A Universal User Interface Tutorial.....	176
24.6 Designing the iPad Layout	176
24.7 Adding Universal Image Assets.....	179
24.8 Designing the iPhone Layout	179
24.9 Adding a Size Class Specific Image File	180
24.10 Removing Redundant Constraints	182
24.11 Previewing Layouts	183
24.12 Testing the Application	184
24.13 Summary.....	184
25. Using Storyboards in Xcode 6	185
25.1 Creating the Storyboard Example Project	185
25.2 Accessing the Storyboard	185
25.3 Adding Scenes to the Storyboard	187
25.4 Configuring Storyboard Segues	188
25.5 Configuring Storyboard Transitions	189
25.6 Associating a View Controller with a Scene.....	189
25.7 Passing Data Between Scenes	190
25.8 Unwinding Storyboard Segues.....	191
25.9 Triggering a Storyboard Segue Programmatically	192
25.10 Summary.....	193
26. Using Xcode 6 Storyboards to Create an iOS 8 Tab Bar Application	195
26.1 An Overview of the Tab Bar.....	195
26.2 Understanding View Controllers in a Multiview Application.....	195
26.3 Setting up the Tab Bar Example Application	196
26.4 Reviewing the Project Files.....	196
26.5 Adding the View Controllers for the Content Views.....	196
26.6 Adding the Tab Bar Controller to the Storyboard.....	197
26.7 Designing the View Controller User interfaces.....	199
26.8 Configuring the Tab Bar Items	199
26.9 Building and Running the Application	200
26.10 Summary.....	201
27. An Overview of iOS 8 Table Views and Xcode 6 Storyboards	203
27.1 An Overview of the Table View	203
27.2 Static vs. Dynamic Table Views.....	203
27.3 The Table View Delegate and dataSource	204

27.4 Table View Styles	204
27.5 Self-Sizing Table Cells.....	205
27.6 Dynamic Type	205
27.7 Table View Cell Styles	207
27.8 Table View Cell Reuse	207
27.9 Summary.....	208
28. Using Xcode 6 Storyboards to Build Dynamic TableViews with Prototype Table View Cells.....	209
28.1 Creating the Example Project	209
28.2 Adding the TableView Controller to the Storyboard	210
28.3 Creating the UITableViewController and UITableViewCell Subclasses.....	211
28.4 Declaring the Cell Reuse Identifier	211
28.5 Designing a Storyboard UITableView Prototype Cell.....	212
28.6 Modifying the AttractionTableViewCell Class.....	213
28.7 Creating the Table View Datasource	213
28.8 Downloading and Adding the Image Files	216
28.9 Compiling and Running the Application	216
28.10 Summary.....	217
29. Implementing iOS 8 TableView Navigation using Storyboards in Xcode 6.....	219
29.1 Understanding the Navigation Controller	219
29.2 Adding the New Scene to the Storyboard	219
29.3 Adding a Navigation Controller	220
29.4 Establishing the Storyboard Segue	221
29.5 Modifying the AttractionDetailViewController Class	222
29.6 Using prepareForSegue to Pass Data between Storyboard Scenes.....	223
29.7 Testing the Application	224
29.8 Summary.....	224
30. An iOS 8 Split View Master–Detail Example	225
30.1 An Overview of Split View and Popovers.....	225
30.2 About the Example Split View Project	226
30.3 Creating the Project.....	226
30.4 Reviewing the Project.....	226
30.5 Configuring Master View Items	227
30.6 Configuring the Detail View Controller.....	229
30.7 Connecting Master Selections to the Detail View	229
30.8 Modifying the DetailViewController Class.....	230
30.9 Testing the Application	231
30.10 Summary.....	231
31. Implementing a Page based iOS 8 Application using UIPageViewController	233

31.1 The UINavigationController Class.....	233
31.2 The UINavigationController DataSource	233
31.3 Navigation Orientation	234
31.4 Spine Location	234
31.5 The UINavigationController Delegate Protocol.....	235
31.6 Summary.....	235
32. An Example iOS 8 UINavigationController Application	237
32.1 The Xcode Page-based Application Template	237
32.2 Creating the Project.....	237
32.3 Adding the Content View Controller	237
32.4 Creating the Data Model	239
32.5 Initializing the UINavigationController.....	243
32.6 Running the UINavigationController Application.....	244
32.7 Summary.....	245
33. Working with Directories in Swift on iOS 8	247
33.1 The Application Documents Directory.....	247
33.2 The NSFileManager, NSFileHandle and NSData Classes	247
33.3 Understanding Pathnames in Swift	248
33.4 Obtaining a Reference to the Default NSFileManager Object.....	248
33.5 Identifying the Current Working Directory	248
33.6 Identifying the Documents Directory	249
33.7 Identifying the Temporary Directory	249
33.8 Changing Directory	250
33.9 Creating a New Directory	250
33.10 Deleting a Directory.....	251
33.11 Listing the Contents of a Directory	251
33.12 Getting the Attributes of a File or Directory.....	252
34. Working with Files in Swift on iOS 8	255
34.1 Creating an NSFileManager Instance.....	255
34.2 Checking for the Existence of a File	255
34.3 Comparing the Contents of Two Files.....	256
34.4 Checking if a File is Readable/Writable/Executable/Deletable	256
34.5 Moving/Renaming a File.....	256
34.6 Copying a File.....	257
34.7 Removing a File.....	257
34.8 Creating a Symbolic Link.....	257
34.9 Reading and Writing Files with NSFileManager.....	258
34.10 Working with Files using the NSFileHandle Class	258
34.11 Creating an NSFileHandle Object.....	258

34.12 NSFileHandle File Offsets and Seeking	259
34.13 Reading Data from a File.....	259
34.14 Writing Data to a File.....	260
34.15 Truncating a File	260
34.16 Summary.....	261
35. iOS 8 Directory Handling and File I/O in Swift – A Worked Example	263
35.1 The Example Application	263
35.2 Setting up the Application Project.....	263
35.3 Designing the User Interface	263
35.4 Checking the Data File on Application Startup	264
35.5 Implementing the Action Method	265
35.6 Building and Running the Example.....	266
36. Preparing an iOS 8 App to use iCloud Storage.....	267
36.1 iCloud Data Storage Services	267
36.2 Preparing an Application to Use iCloud Storage.....	268
36.3 Enabling iCloud Support for an iOS 8 Application.....	268
36.4 Reviewing the iCloud Entitlements File	268
36.5 Accessing Multiple Ubiquity Containers	269
36.6 Ubiquity Container URLs.....	269
36.7 Summary.....	270
37. Managing Files using the iOS 8 UIDocument Class	271
37.1 An Overview of the UIDocument Class.....	271
37.2 Subclassing the UIDocument Class	271
37.3 Conflict Resolution and Document States	271
37.4 The UIDocument Example Application	272
37.5 Creating a UIDocument Subclass	272
37.6 Designing the User Interface	272
37.7 Implementing the Application Data Structure.....	274
37.8 Implementing the contentsForType Method	274
37.9 Implementing the loadFromContents Method	275
37.10 Loading the Document at App Launch	275
37.11 Saving Content to the Document	278
37.12 Testing the Application.....	278
37.13 Summary.....	279
38. Using iCloud Storage in an iOS 8 Application	281
38.1 iCloud Usage Guidelines	281
38.2 Preparing the iCloudStore Application for iCloud Access	281
38.3 Configuring the View Controller	282

38.4 Implementing the viewDidLoad Method	283
38.5 Implementing the metadataQueryDidFinishGathering Method	284
38.6 Implementing the saveDocument Method	287
38.7 Enabling iCloud Document and Data Storage	287
38.8 Running the iCloud Application	288
38.9 Reviewing and Deleting iCloud Based Documents	288
38.10 Making a Local File Ubiquitous	289
38.11 Summary	289
39. Synchronizing iOS 8 Key-Value Data using iCloud	291
39.1 An Overview of iCloud Key-Value Data Storage	291
39.2 Sharing Data Between Applications	292
39.3 Data Storage Restrictions	292
39.4 Conflict Resolution	292
39.5 Receiving Notification of Key-Value Changes	292
39.6 An iCloud Key-Value Data Storage Example	293
39.7 Enabling the Application for iCloud Key Value Data Storage	293
39.8 Designing the User Interface	293
39.9 Implementing the View Controller	294
39.10 Modifying the viewDidLoad Method	294
39.11 Implementing the Notification Method	295
39.12 Implementing the saveData Method	296
39.13 Testing the Application	296
40. iOS 8 Data Persistence using Archiving	299
40.1 An Overview of Archiving	299
40.2 The Archiving Example Application	300
40.3 Designing the User Interface	300
40.4 Checking for the Existence of the Archive File on Startup	301
40.5 Archiving Object Data in the Action Method	302
40.6 Testing the Application	303
40.7 Summary	303
41. iOS 8 Database Implementation using SQLite	305
41.1 What is SQLite?	305
41.2 Structured Query Language (SQL)	305
41.3 Trying SQLite on MacOS X	306
41.4 Preparing an iOS Application Project for SQLite Integration	307
41.5 SQLite, Swift and Wrappers	307
41.6 Key FMDB Classes	308
41.7 Creating and Opening a Database	308
41.8 Creating a Database Table	308

41.9 Extracting Data from a Database Table	309
41.10 Closing a SQLite Database	309
41.11 Summary.....	309
42. An Example SQLite based iOS 8 Application using Swift and FMDB	311
42.1 About the Example SQLite Application	311
42.2 Creating and Preparing the SQLite Application Project	311
42.3 Checking Out the FMDB Source Code	311
42.4 Designing the User Interface	313
42.5 Creating the Database and Table	314
42.6 Implementing the Code to Save Data to the SQLite Database	316
42.7 Implementing Code to Extract Data from the SQLite Database	316
42.8 Building and Running the Application	317
42.9 Summary.....	318
43. Working with iOS 8 Databases using Core Data	319
43.1 The Core Data Stack.....	319
43.2 Managed Objects.....	320
43.3 Managed Object Context.....	320
43.4 Managed Object Model	320
43.5 Persistent Store Coordinator	321
43.6 Persistent Object Store	321
43.7 Defining an Entity Description	321
43.8 Obtaining the Managed Object Context.....	323
43.9 Getting an Entity Description	323
43.10 Generating a Managed Object Subclass	323
43.11 Setting the Attributes of a Managed Object.....	324
43.12 Saving a Managed Object	324
43.13 Fetching Managed Objects	324
43.14 Retrieving Managed Objects based on Criteria	324
43.15 Accessing the Data in a Retrieved Managed Object.....	325
43.16 Summary.....	325
44. An iOS 8 Core Data Tutorial	327
44.1 The Core Data Example Application	327
44.2 Creating a Core Data based Application	327
44.3 Creating the Entity Description.....	327
44.4 Generating the Managed Object Subclass.....	328
44.5 Modifying the Entity Class Name.....	329
44.6 Designing the User Interface	330
44.7 Accessing the Managed Object Context	330
44.8 Saving Data to the Persistent Store using Core Data.....	331

44.9 Retrieving Data from the Persistent Store using Core Data	332
44.10 Building and Running the Example Application	333
44.11 Summary.....	333
45. An Introduction to CloudKit Data Storage on iOS 8	335
45.1 An Overview of CloudKit.....	335
45.2 CloudKit Containers	335
45.3 CloudKit Public Database.....	336
45.4 CloudKit Private Databases.....	336
45.5 Data Storage and Transfer Quotas	336
45.6 CloudKit Records.....	336
45.7 CloudKit Record IDs	338
45.8 CloudKit References.....	338
45.9 CloudKit Assets	338
45.10 Record Zones	339
45.11 CloudKit Subscriptions	339
45.12 Obtaining iCloud User Information.....	340
45.13 CloudKit Dashboard	341
45.14 Summary.....	342
46. An iOS 8 CloudKit Example	343
46.1 About the Example CloudKit Project	343
46.2 Creating the CloudKit Example Project.....	343
46.3 Designing the User Interface	344
46.4 Establishing Outlets and Actions	345
46.5 Accessing the Public Database.....	346
46.6 Hiding the Keyboard	347
46.7 Implementing the selectPhoto method.....	347
46.8 Saving a Record to the Cloud Database.....	349
46.9 Implementing the notifyUser Method.....	350
46.10 Testing the Record Saving Method.....	350
46.11 Searching for Cloud Database Records.....	351
46.12 Updating Cloud Database Records	352
46.13 Deleting a Cloud Record	353
46.14 Testing the Application.....	354
46.15 Summary.....	354
47. An iOS 8 CloudKit Subscription Example	355
47.1 Push Notifications and CloudKit Subscriptions.....	355
47.2 Registering an App to Receive Push Notifications.....	355
47.3 Configuring a CloudKit Subscription	356
47.4 Handling Remote Notifications.....	358

47.5 Implementing the <code>didReceiveRemoteNotification</code> Method	358
47.6 Fetching a Record From a Cloud Database	359
47.7 Implementing the <code>didFinishLaunchingWithOptions</code> Method	360
47.8 Testing the Application	361
47.9 Summary.....	361
48. An Overview of iOS 8 Multitouch, Taps and Gestures	363
48.1 The Responder Chain.....	363
48.2 Forwarding an Event to the Next Responder	364
48.3 Gestures.....	364
48.4 Taps.....	364
48.5 Touches.....	364
48.6 Touch Notification Methods.....	364
48.6.1 <i>touchesBegan</i> method	364
48.6.2 <i>touchesMoved</i> method.....	365
48.6.3 <i>touchesEnded</i> method.....	365
48.6.4 <i>touchesCancelled</i> method	365
48.7 Summary.....	365
49. An Example iOS 8 Touch, Multitouch and Tap Application.....	367
49.1 The Example iOS 8 Tap and Touch Application.....	367
49.2 Creating the Example iOS Touch Project	367
49.3 Designing the User Interface	367
49.4 Enabling Multitouch on the View	368
49.5 Implementing the <code>touchesBegan</code> Method.....	369
49.6 Implementing the <code>touchesMoved</code> Method	369
49.7 Implementing the <code>touchesEnded</code> Method	370
49.8 Getting the Coordinates of a Touch.....	370
49.9 Building and Running the Touch Example Application	370
50. Detecting iOS 8 Touch Screen Gesture Motions	373
50.1 The Example iOS 8 Gesture Application	373
50.2 Creating the Example Project	373
50.3 Designing the Application User Interface	373
50.4 Implementing the <code>touchesBegan</code> Method.....	374
50.5 Implementing the <code>touchesMoved</code> Method	375
50.6 Implementing the <code>touchesEnded</code> Method	375
50.7 Building and Running the Gesture Example	376
50.8 Summary.....	376
51. Identifying Gestures using iOS 8 Gesture Recognizers	377
51.1 The <code>UIGestureRecognizer</code> Class	377

51.2 Recognizer Action Messages	378
51.3 Discrete and Continuous Gestures	378
51.4 Obtaining Data from a Gesture.....	378
51.5 Recognizing Tap Gestures.....	378
51.6 Recognizing Pinch Gestures	379
51.7 Detecting Rotation Gestures	379
51.8 Recognizing Pan and Dragging Gestures	379
51.9 Recognizing Swipe Gestures	379
51.10 Recognizing Long Touch (Touch and Hold) Gestures.....	380
51.11 Summary.....	380
52. An iOS 8 Gesture Recognition Tutorial	381
52.1 Creating the Gesture Recognition Project	381
52.2 Designing the User Interface	381
52.3 Implementing the Action Methods	383
52.4 Testing the Gesture Recognition Application	384
52.5 Summary.....	384
53. Implementing TouchID Authentication in iOS 8 Apps	385
53.1 The Local Authentication Framework.....	385
53.2 Checking for TouchID Availability	385
53.3 Evaluating TouchID Policy.....	386
53.4 A TouchID Example Project.....	386
53.5 Checking for TouchID Availability	387
53.6 Seeking TouchID Authentication	389
53.7 Testing the Application	390
53.8 Summary.....	391
54. An Overview of iOS 8 Collection View and Flow Layout	393
54.1 An Overview of Collection Views	393
54.2 The UICollectionView Class.....	394
54.3 The UICollectionViewCell Class.....	395
54.4 The UICollectionViewReusableView Class	395
54.5 The UICollectionViewFlowLayout Class	395
54.6 The UICollectionViewLayoutAttributes Class	396
54.7 The UICollectionViewDataSource Protocol	396
54.8 The UICollectionViewDelegate Protocol.....	397
54.9 The UICollectionViewDelegateFlowLayout Protocol	397
54.10 Cell and View Reuse.....	398
54.11 Summary.....	399
55. An iOS 8 Storyboard-based Collection View Tutorial	401

55.1	Creating the Collection View Example Project	401
55.2	Removing the Template View Controller	401
55.3	Adding a Collection View Controller to the Storyboard	401
55.4	Adding the Collection View Cell Class to the Project	403
55.5	Designing the Cell Prototype	403
55.6	Implementing the Data Model	405
55.7	Implementing the Data Source	406
55.8	Testing the Application	408
55.9	Setting Sizes for Cell Items.....	408
55.10	Changing Scroll Direction.....	409
55.11	Implementing a Supplementary View	410
55.12	Implementing the Supplementary View Protocol Methods	412
55.13	Summary.....	413
56.	Subclassing and Extending the iOS 8 Collection View Flow Layout.....	415
56.1	About the Example Layout Class.....	415
56.2	Subclassing the UICollectionViewFlowLayout Class	415
56.3	Extending the New Layout Class.....	415
56.4	Implementing the layoutAttributesForItemAtIndexPath Method	416
56.5	Implementing the layoutAttributesForElementsInRect Method.....	417
56.6	Implementing the modifyLayoutAttributes Method	418
56.7	Adding the New Layout and Pinch Gesture Recognizer	419
56.8	Implementing the Pinch Recognizer	420
56.9	Avoiding Image Clipping	421
56.10	Testing the Application	422
56.11	Summary.....	422
57.	Drawing iOS 8 2D Graphics with Core Graphics.....	423
57.1	Introducing Core Graphics and Quartz 2D.....	423
57.2	The drawRect Method	423
57.3	Points, Coordinates and Pixels.....	423
57.4	The Graphics Context	424
57.5	Working with Colors in Quartz 2D	424
57.6	Summary.....	425
58.	Interface Builder Live Views and iOS 8 Embedded Frameworks.....	427
58.1	Embedded Frameworks.....	427
58.2	Interface Builder Live Views	427
58.3	Creating the Example Project	428
58.4	Adding an Embedded Framework	429
58.5	Implementing the Drawing Code in the Framework	430
58.6	Making the View Designable	431

58.7 Making Variables Inspectable	432
58.8 Summary.....	433
59. An iOS 8 Graphics Tutorial using Core Graphics and Core Image	435
59.1 The iOS Drawing Example Application.....	435
59.2 Creating the New Project.....	435
59.3 Creating the UIView Subclass	435
59.4 Locating the drawRect Method in the UIView Subclass	436
59.5 Drawing a Line	436
59.6 Drawing Paths.....	438
59.7 Drawing a Rectangle	439
59.8 Drawing an Ellipse or Circle	440
59.9 Filling a Path with a Color	441
59.10 Drawing an Arc	443
59.11 Drawing a Cubic Bézier Curve	444
59.12 Drawing a Quadratic Bézier Curve	445
59.13 Dashed Line Drawing	446
59.14 Drawing Shadows	447
59.15 Drawing Gradients	448
59.16 Drawing an Image into a Graphics Context	452
59.17 Image Filtering with the Core Image Framework	454
59.18 Summary.....	455
60. Basic iOS 8 Animation using Core Animation	457
60.1 UIView Core Animation Blocks	457
60.2 Understanding Animation Curves.....	458
60.3 Receiving Notification of Animation Completion	458
60.4 Performing Affine Transformations.....	459
60.5 Combining Transformations	459
60.6 Creating the Animation Example Application.....	459
60.7 Implementing the Variables	460
60.8 Drawing in the UIView	460
60.9 Detecting Screen Touches and Performing the Animation	460
60.10 Building and Running the Animation Application.....	462
60.11 Summary.....	463
61. iOS 8 UIKit Dynamics – An Overview.....	465
61.1 Understanding UIKit Dynamics	465
61.2 The UIKit Dynamics Architecture	465
61.2.1 Dynamic Items.....	466
61.2.2 Dynamic Behaviors	466
61.2.3 The Reference View	466

61.2.4 <i>The Dynamic Animator</i>	466
61.3 Implementing UIKit Dynamics in an iOS 8 Application	467
61.4 Dynamic Animator Initialization	467
61.5 Configuring Gravity Behavior.....	468
61.6 Configuring Collision Behavior	468
61.7 Configuring Attachment Behavior	470
61.8 Configuring Snap Behavior	471
61.9 Configuring Push Behavior.....	471
61.10 The UIDynamicItemBehavior Class	473
61.11 Combining Behaviors to Create a Custom Behavior	473
61.12 Summary.....	474
62. An iOS 8 UIKit Dynamics Tutorial	475
62.1 Creating the UIKit Dynamics Example Project	475
62.2 Adding the Dynamic Items.....	475
62.3 Creating the Dynamic Animator Instance.....	476
62.4 Adding Gravity to the Views	477
62.5 Implementing Collision Behavior	478
62.6 Attaching a View to an Anchor Point.....	479
62.7 Implementing a Spring Attachment Between two Views.....	481
62.8 Summary.....	483
63. An Introduction to iOS 8 Sprite Kit Programming.....	485
63.1 What is Sprite Kit?	485
63.2 The Key Components of a Sprite Kit Game	485
63.2.1 <i>Sprite Kit View</i>	486
63.2.2 <i>Scenes</i>	486
63.2.3 <i>Nodes</i>	486
63.2.4 <i>Physics Bodies</i>	486
63.2.5 <i>Physics World</i>	487
63.2.6 <i>Actions</i>	487
63.2.7 <i>Transitions</i>	487
63.2.8 <i>Texture Atlas</i>	487
63.2.9 <i>Constraints</i>	488
63.3 An Example Sprite Kit Game Hierarchy.....	488
63.4 The Sprite Kit Game Rendering Loop.....	488
63.5 The Sprite Kit Level Editor.....	489
63.6 Summary.....	489
64. An iOS 8 Sprite Kit Level Editor Game Tutorial	491
64.1 About the Sprite Kit Demo Game	491
64.2 Creating the SpriteKitDemo Project	492

64.3	Reviewing the SpriteKit Game Template Project	492
64.4	Restricting Interface Orientation	494
64.5	Modifying the GameScene SpriteKit Scene File	494
64.6	Creating the Archery Scene	496
64.7	Transitioning to the Archery Scene	497
64.8	Adding the Texture Atlas	498
64.9	Designing the Archery Scene	499
64.10	Preparing the Archery Scene	501
64.11	Preparing the Animation Texture Atlas	502
64.12	Animating the Archer Sprite Node	503
64.13	Creating the Arrow Sprite Node	503
64.14	Shooting the Arrow.....	504
64.15	Adding the Ball Sprite Node.....	505
64.16	Summary.....	507
65.	An iOS 8 Sprite Kit Collision Handling Tutorial	509
65.1	Defining the Category Bit Masks	509
65.2	Assigning the Category Masks to the Sprite Nodes	509
65.3	Configuring the Collision and Contact Masks	510
65.4	Implementing the Contact Delegate	511
65.5	Implementing a Physics Joint Between Nodes	512
65.6	Game Over	514
65.7	Summary.....	515
66.	An iOS 8 Sprite Kit Particle Emitter Tutorial	517
66.1	What is the Particle Emitter?.....	517
66.2	The Particle Emitter Editor	517
66.3	The SKEmitterNode Class.....	518
66.4	Using the Particle Emitter Editor	518
66.5	Particle Emitter Node Properties.....	520
66.5.1	<i>Background</i>	520
66.5.2	<i>Particle Texture</i>	520
66.5.3	<i>Particle Birthrate</i>	520
66.5.4	<i>Particle Life Cycle</i>	520
66.5.5	<i>Particle Position Range</i>	520
66.5.6	<i>Angle</i>	520
66.5.7	<i>Particle Speed</i>	520
66.5.8	<i>Particle Acceleration</i>	521
66.5.9	<i>Particle Scale</i>	521
66.5.10	<i>Particle Rotation</i>	521
66.5.11	<i>Particle Color</i>	521
66.5.12	<i>Particle Blend Mode</i>	521

66.6 Experimenting with the Particle Emitter Editor	522
66.7 Bursting a Ball using Particle Emitter Effects	523
66.8 Adding the Burst Particle Emitter Effect	524
66.9 Summary	526
67. Integrating iAds into an iOS 8 App	527
67.1 Preparing to Run iAds within an Application	527
67.2 iAd Advertisement Formats	527
67.2.1 Banner Ads	527
67.2.2 Interstitial Ads	528
67.2.3 Medium Rectangle Ads	529
67.2.4 Pre-Roll Video Ads	530
67.3 Creating an Example iAds Application	531
67.4 Adding the iAds Framework to the Xcode Project	531
67.5 Enabling Banner Ads	531
67.6 Adding a Medium Rectangle Ad	532
67.7 Implementing an Interstitial Ad	533
67.8 Configuring iAds Test Settings	535
67.9 Summary	536
68. iOS 8 Multitasking, Background Transfer Service and Fetching	537
68.1 Understanding iOS Application States	537
68.2 A Brief Overview of the Multitasking Application Lifecycle	538
68.3 Checking for Multitasking Support	539
68.4 Enabling Multitasking for an iOS Application	539
68.5 Supported Forms of Background Execution	540
68.6 An Overview of Background Fetch	540
68.7 An Overview of Remote Notifications	542
68.8 An Overview of Local Notifications	542
68.9 An Overview of Background Transfer Service	542
68.10 The Rules of Background Execution	543
68.11 Summary	543
69. Scheduling iOS 8 Local Notifications	545
69.1 Creating the Local Notification App Project	545
69.2 Adding a Sound File to the Project	545
69.3 Requesting Permission to Trigger Alerts	546
69.4 Locating the Application Delegate Method	546
69.5 Scheduling the Local Notification	547
69.6 Testing the Application	547
69.7 Cancelling Scheduled Notifications	548
69.8 Immediate Triggering of a Local Notification	548

69.9 Summary.....	549
70. An Overview of iOS 8 Application State Preservation and Restoration	551
70.1 The Preservation and Restoration Process	551
70.2 Opting In to Preservation and Restoration	552
70.3 Assigning Restoration Identifiers	552
70.4 Default Preservation Features of UIKit	553
70.5 Saving and Restoring Additional State Information.....	553
70.6 Understanding the Restoration Process	554
70.7 Saving General Application State.....	556
70.8 Summary.....	556
71. An iOS 8 State Preservation and Restoration Tutorial.....	557
71.1 Creating the Example Application	557
71.2 Trying the Application without State Preservation	557
71.3 Opting-in to State Preservation	557
71.4 Setting Restoration Identifiers.....	558
71.5 Encoding and Decoding View Controller State	559
71.6 Adding a Navigation Controller to the Storyboard	561
71.7 Adding the Third View Controller	561
71.8 Creating the Restoration Class.....	563
71.9 Summary.....	564
72. Integrating Maps into iOS 8 Applications using MKMapItem	565
72.1 MKMapItem and MKPlacemark Classes	565
72.2 An Introduction to Forward and Reverse Geocoding	566
72.3 Creating MKPlacemark Instances	568
72.4 Working with MKMapItem	568
72.5 MKMapItem Options and Enabling Turn-by-Turn Directions	569
72.6 Adding Item Details to an MKMapItem	570
72.7 Summary.....	571
73. An Example iOS 8 MKMapItem Application	573
73.1 Creating the MapItem Project	573
73.2 Designing the User Interface	573
73.3 Converting the Destination using Forward Geocoding.....	574
73.4 Launching the Map	576
73.5 Building and Running the Application	576
73.6 Summary.....	577
74. Getting Location Information using the iOS 8 Core Location Framework	579
74.1 The Core Location Manager	579

74.2 Requesting Location Access Authorization	579
74.3 Configuring the Desired Location Accuracy	580
74.4 Configuring the Distance Filter	580
74.5 The Location Manager Delegate	581
74.6 Starting Location Updates	581
74.7 Obtaining Location Information from CLLocation Objects	582
74.7.1 Longitude and Latitude	582
74.7.2 Accuracy	582
74.7.3 Altitude	582
74.8 Calculating Distances	582
74.9 Location Information and Multitasking	583
74.10 Summary	583
75. An Example iOS 8 Location Application	585
75.1 Creating the Example iOS 8 Location Project	585
75.2 Designing the User Interface	585
75.3 Configuring the CLLocationManager Object	587
75.4 Setting up the Usage Description Key	587
75.5 Implementing the Action Method	587
75.6 Implementing the Application Delegate Methods	588
75.7 Building and Running the Location Application	589
76. Working with Maps on iOS 8 with MapKit and the MKMapView Class	591
76.1 About the MapKit Framework	591
76.2 Understanding Map Regions	591
76.3 About the MKMapView Tutorial	592
76.4 Creating the Map Project	592
76.5 Adding the MapKit Framework to the Xcode Project	592
76.6 Adding the Navigation Controller	592
76.7 Creating the MKMapView Instance and Toolbar	593
76.8 Obtaining Location Information Permission	595
76.9 Setting up the Usage Description Key	596
76.10 Configuring the Map View	596
76.11 Changing the MapView Region	596
76.12 Changing the Map Type	597
76.13 Testing the MapView Application	597
76.14 Updating the Map View based on User Movement	598
76.15 Summary	599
77. Working with MapKit Local Search in iOS 8	601
77.1 An Overview of iOS 8 Local Search	601
77.2 Adding Local Search to the MapSample Application	603

77.3 Adding the Local Search Text Field	603
77.4 Performing the Local Search.....	604
77.5 Testing the Application	606
77.6 Summary.....	607
78. Using MKDirections to get iOS 8 Map Directions and Routes.....	609
78.1 An Overview of MKDirections.....	609
78.2 Adding Directions and Routes to the MapSample Application	611
78.3 Adding the New Classes to the Project.....	611
78.4 Configuring the Results Table View	611
78.5 Implementing the Result Table View Segue	613
78.6 Adding the Route Scene	614
78.7 Getting the Route and Directions	615
78.8 Establishing the Route Segue	617
78.9 Testing the Application	618
78.10 Summary.....	618
79. An Introduction to Extensions in iOS 8.....	619
79.1 iOS Extensions – An Overview	619
79.2 Extension Types	619
79.2.1 Today Extension	620
79.2.2 Share Extension	620
79.2.3 Action Extension.....	621
79.2.4 Photo Editing Extension.....	622
79.2.5 Document Provider Extension	623
79.2.6 Custom Keyboard Extension	623
79.3 Creating Extensions	623
79.4 Summary.....	624
80. An iOS 8 Today Extension Widget Tutorial	625
80.1 About the Example Extension Widget	625
80.2 Creating the Example Project	625
80.3 Adding the Extension to the Project.....	625
80.4 Reviewing the Extension Files.....	628
80.5 Designing the Widget User Interface.....	628
80.6 Setting the Preferred Content Size in Code	630
80.7 Modifying the Widget View Controller	630
80.8 Testing the Extension	632
80.9 Opening the Containing App from the Extension.....	632
80.10 Summary.....	634
81. Creating an iOS 8 Photo Editing Extension	637

81.1 Creating a Photo Editing Extension.....	637
81.2 Accessing the Photo Editing Extension	638
81.3 Configuring the Info.plist File.....	640
81.4 Designing the User Interface	641
81.5 The PHContentEditingController Protocol.....	642
81.6 Photo Extensions and Adjustment Data	642
81.7 Receiving the Content	642
81.8 Implementing the Filter Actions	644
81.9 Returning the Image to the Photos App	646
81.10 Testing the Application	648
81.11 Summary.....	649
82. Creating an iOS 8 Action Extension	651
82.1 An Overview of Action Extensions	651
82.2 About the Action Extension Example	652
82.3 Creating the Action Extension Project	652
82.4 Adding the Action Extension Target	652
82.5 Configuring the Action Extension	653
82.6 Designing the Action Extension User Interface	654
82.7 Receiving the Content	655
82.8 Returning the Modified Data to the Host App	658
82.9 Testing the Extension	658
82.10 Summary.....	660
83. Receiving Data from an iOS 8 Action Extension	663
83.1 Creating the Example Project	663
83.2 Designing the User Interface	663
83.3 Importing the Mobile Core Services Framework.....	664
83.4 Adding a Share Button to the Application	664
83.5 Receiving Data from an Extension	666
83.6 Testing the Application	667
83.7 Summary.....	667
84. Using iOS 8 Event Kit to Create Date and Location Based Reminders.....	669
84.1 An Overview of the Event Kit Framework	669
84.2 The EKEventStore Class	669
84.3 Accessing Calendars in the Database	671
84.4 Creating Reminders	671
84.5 Creating Alarms	672
84.6 Creating the Example Project	672
84.7 Designing the User Interface for the Date/Time Based Reminder Screen	672
84.8 Implementing the Reminder Code	674

84.9 Hiding the Keyboard	676
84.10 Designing the Location-based Reminder Screen	676
84.11 Creating a Location-based Reminder.....	677
84.12 Setting up the Usage Description Key.....	680
84.13 Testing the Application	680
84.14 Summary.....	681
85. Accessing the iOS 8 Camera and Photo Library	683
85.1 The UIImagePickerController Class.....	683
85.2 Creating and Configuring a UIImagePickerController Instance	683
85.3 Configuring the UIImagePickerController Delegate	684
85.4 Detecting Device Capabilities	685
85.5 Saving Movies and Images.....	686
85.6 Summary.....	687
86. An Example iOS 8 Camera Application	689
86.1 An Overview of the Application	689
86.2 Creating the Camera Project	689
86.3 Designing the User Interface	689
86.4 Implementing the Action Methods	691
86.5 Writing the Delegate Methods	692
86.6 Building and Running the Application	693
87. iOS 8 Video Playback using AVPlayer and AVPlayerViewController	695
87.1 The AVPlayer and AVPlayerViewController Classes	695
87.2 The iOS Movie Player Example Application	695
87.3 Adding the AVKit Framework to the Xcode Project.....	695
87.4 Designing the User Interface	696
87.5 Initializing Video Playback	696
87.6 Build and Run the Application	697
87.7 Creating AVPlayerViewController Instance from Code	698
87.8 Summary.....	698
88. Playing Audio on iOS 8 using AVAudioPlayer	699
88.1 Supported Audio Formats.....	699
88.2 Receiving Playback Notifications	699
88.3 Controlling and Monitoring Playback	700
88.4 Creating the Audio Example Application	700
88.5 Adding an Audio File to the Project Resources.....	700
88.6 Designing the User Interface	701
88.7 Implementing the Action Methods	702
88.8 Creating and Initializing the AVAudioPlayer Object	703

88.9 Implementing the AVAudioPlayerDelegate Protocol Methods	703
88.10 Building and Running the Application	704
88.11 Summary.....	704
89. Recording Audio on iOS 8 with AVAudioRecorder	705
89.1 An Overview of the AVAudioRecorder Tutorial	705
89.2 Creating the Recorder Project	705
89.3 Designing the User Interface	705
89.4 Creating the AVAudioRecorder Instance	707
89.5 Implementing the Action Methods	708
89.6 Implementing the Delegate Methods	709
89.7 Testing the Application	710
90. Integrating Twitter and Facebook into iOS 8 Applications	711
90.1 The UIActivityViewController class.....	711
90.2 The Social Framework.....	711
90.3 Accounts Framework.....	712
90.4 Using the UIActivityViewController Class	713
90.5 Using the SLComposeViewController Class	714
90.6 Summary.....	715
91. An iOS 8 Facebook Integration Tutorial using UIActivityViewController	717
91.1 Creating the Facebook Social App	717
91.2 Designing the User Interface	717
91.3 Creating Outlets and Actions.....	718
91.4 Implementing the selectImage and Delegate Methods	719
91.5 Hiding the Keyboard	720
91.6 Posting the Message to Facebook	720
91.7 Running the Social Application	721
91.8 Summary.....	722
92. iOS 8 Facebook and Twitter Integration using SLRequest	723
92.1 Using SLRequest and the Account Framework.....	723
92.2 Twitter Integration using SLRequest	724
92.3 Facebook Integration using SLRequest.....	727
92.4 Summary.....	729
93. An iOS 8 Twitter Integration Tutorial using SLRequest.....	731
93.1 Creating the TwitterApp Project.....	731
93.2 Designing the User Interface	731
93.3 Modifying the View Controller Class	732
93.4 Accessing the Twitter API	733

93.5 Calling the getTimeLine Method	736
93.6 The Table View Delegate Methods	736
93.7 Building and Running the Application	737
93.8 Summary.....	737
94. Making Store Purchases with the SKStoreProductViewController Class	739
94.1 The SKStoreProductViewController Class.....	739
94.2 Creating the Example Project	740
94.3 Creating the User Interface	740
94.4 Displaying the Store Kit Product View Controller	741
94.5 Implementing the Delegate Method	742
94.6 Testing the Application	743
94.7 Summary.....	743
95. Building In-App Purchasing into iOS 8 Applications.....	745
95.1 In-App Purchase Options	745
95.2 Uploading App Store Hosted Content	746
95.3 Configuring In-App Purchase Items	746
95.4 Sending a Product Request.....	746
95.5 Accessing the Payment Queue	747
95.6 The Transaction Observer Object	747
95.7 Initiating the Purchase.....	748
95.8 The Transaction Process	748
95.9 Transaction Restoration Process	750
95.10 Testing In-App Purchases.....	750
95.11 Summary.....	750
96. Preparing an iOS 8 Application for In-App Purchases.....	751
96.1 About the Example Application	751
96.2 Creating the Xcode Project	751
96.3 Registering and Enabling the App ID for In App Purchasing	751
96.4 Configuring the Application in iTunes Connect	752
96.5 Creating an In-App Purchase Item	752
96.6 Summary.....	753
97. An iOS 8 In-App Purchase Tutorial	755
97.1 The Application User Interface	755
97.2 Designing the Storyboard	756
97.3 Creating the Purchase View Controller Class.....	757
97.4 Storing the Home View Controller in the App Delegate Class	759
97.5 Completing the ViewController Class	759
97.6 Completing the PurchaseViewController Class	760

97.7 Testing the Application	763
97.8 Troubleshooting.....	763
97.9 Summary.....	764
98. Configuring and Creating App Store Hosted Content for iOS 8 In-App Purchases.....	765
98.1 Configuring an Application for In-App Purchase Hosted Content	765
98.2 The Anatomy of an In-App Purchase Hosted Content Package.....	766
98.3 Creating an In-App Purchase Hosted Content Package	766
98.4 Archiving the Hosted Content Package	767
98.5 Validating the Hosted Content Package	767
98.6 Uploading the Hosted Content Package.....	768
98.7 Summary.....	768
99. Preparing and Submitting an iOS 8 Application to the App Store.....	769
99.1 Verifying the iOS Distribution Certificate	769
99.2 Adding App Icons.....	770
99.3 Designing the Launch Screen.....	771
99.4 Assign the Project to a Team	771
99.5 Archiving the Application for Distribution	772
99.6 Configuring the Application in iTunes Connect	772
99.7 Validating and Submitting the Application	773
99.8 Configuring and Submitting the App for Review	774
Index	777

1. Start Here

The goal of this book is to teach the skills necessary to create iOS applications using the iOS 8 SDK, Xcode 6 and the Swift programming language.

How you make use of this book will depend to a large extent on whether you are new to iOS development, or have worked with iOS 7 and need to get up to speed on the features of iOS 8 and the Swift programming language. Rest assured, however, that the book is intended to address both category of reader.

1.1 For New iOS Developers

If you are entirely new to iOS development then the entire contents of the book will be relevant to you.

Beginning with the basics, this book provides an outline of the steps necessary to set up an iOS development environment. An introduction to the architecture of iOS 8 and programming in Swift is provided, followed by an in-depth look at the design of iOS applications and user interfaces. More advanced topics such as file handling, database management, in-app purchases, graphics drawing and animation are also covered, as are touch screen handling, gesture recognition, multitasking, iAds integration, location management, local notifications, camera access and video and audio playback support. Other features are also covered including Auto Layout, Twitter and Facebook integration, App Store hosted in-app purchase content, collection views, Sprite Kit-based game development, local map search and user interface animation using UIKit dynamics.

The key new features of the iOS 8 SDK and Xcode 6 are also covered, including Swift playgrounds, universal user interface design using size classes, app extensions, Interface Builder Live Views, embedded frameworks, CloudKit data storage and TouchID authentication.

The aim of this book, therefore, is to teach you the skills necessary to build your own apps for iOS 8. Assuming you are ready to download the iOS 8 SDK and Xcode, have an Intel-based Mac and ideas for some apps to develop, you are ready to get started.

1.2 For iOS 7 Developers

If you have already read the iOS 7 edition of this book, or have experience with the iOS 7 SDK then you might prefer to go directly to the new chapters in this iOS 8 edition of the book.

All chapters have been updated to reflect the changes and features introduced as part of iOS 8 and Xcode 6. Chapters included in this edition that were not contained in the previous edition, or have been significantly rewritten for iOS 8 and Xcode 6 are as follows:

Start Here

- *An Introduction to Swift Playgrounds*
- *Swift Data Types, Constants and Variables*
- *Swift Operators and Expressions*
- *Swift Flow Control*
- *The Swift Switch Statement*
- *An Overview of Swift Functions and Closures*
- *The Basics of Object Oriented Programming in Swift*
- *An Introduction to Swift Inheritance*
- *Working with Array and Dictionary Collections in Swift*
- *Using Size Classes to Design Universal iOS User Interfaces*
- *An Introduction to CloudKit Data Storage on iOS 8*
- *An iOS 8 CloudKit Example*
- *An iOS 8 CloudKit Subscription Example*
- *Implementing TouchID Authentication in iOS 8 Apps*
- *Interface Builder Live Views and iOS 8 Embedded Frameworks*
- *An Introduction to Extensions in iOS 8*
- *An iOS 8 Today Extension Widget Tutorial*
- *Creating an iOS 8 Photo Editing Extension*
- *Creating an iOS 8 Action Extension*
- *Receiving Data from an iOS 8 Action Extension*

In addition the following changes have also been made:

- All chapters have been updated where necessary to reflect the changes made to Xcode 6.
- All chapters and examples have been rewritten to use Swift instead of Objective-C.
- TableView examples have been updated to cover self-sizing table cells and dynamic type.
- The SpriteKit chapters have been updated to use the new Xcode 6 SpriteKit Level Editor environment.
- The SQLite chapters have been updated to make use of the FMDB wrapper classes.
- The video playback chapter has been rewritten to use the AVPlayerViewControler and AVPlayer classes.

1.3 Source Code Download

The source code and Xcode project files for the examples contained in this book are available for download at:

<http://www.ebookfrenzy.com/direct/ios8/>

1.4 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book, or have any comments, questions or concerns please contact us at feedback@ebookfrenzy.com.

1.5 Errata

Whilst we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions at the following URL:

<http://www.ebookfrenzy.com/errata/ios8.html>

In the event that you find an error not listed in the errata, please let us know by emailing our technical support team at *feedback@ebookfrenzy.com*.

2. Joining the Apple iOS Developer Program

The first step in the process of learning to develop iOS 8 based applications involves gaining an understanding of the differences between *Registered Apple Developers* and *iOS Developer Program Members*. Having gained such an understanding, the next choice is to decide the point at which it makes sense for you to pay to join the iOS Developer Program. With these goals in mind, this chapter will cover the differences between the two categories of developer, outline the costs and benefits of joining the developer program and, finally, walk through the steps involved in obtaining each membership level.

2.1 Registered Apple Developer

There is no fee associated with becoming a registered Apple developer. Simply visit the following web page to begin the registration process:

<http://developer.apple.com/programs/register/>

An existing Apple ID (used for making iTunes or App Store purchases) is usually adequate to complete the registration process.

Once the registration process is complete, access is provided to developer resources such as online documentation and tutorials. Registered developers are also able to download older versions of the iOS SDK and Xcode development environment.

2.2 Downloading Xcode 6 and the iOS 8 SDK

The latest versions of both the iOS SDK and Xcode can be downloaded free of charge from the Mac App Store. Since the tools are free, this raises the question of whether to upgrade to the iOS Developer Program, or to remain as a Registered Apple Developer. It is important, therefore, to understand the key benefits of the iOS Developer Program.

2.3 iOS Developer Program

Membership in the iOS Developer Program currently costs \$99 per year. As previously mentioned, membership includes access to the latest versions of the iOS SDK and Xcode development environment. The benefits of membership, however, go far beyond those offered at the Registered Apple Developer level.

One of the key advantages of the developer program is that it permits the creation of certificates and provisioning profiles to test applications on physical devices. Although Xcode includes device simulators which allow for a significant amount of testing to be performed, there are certain areas of functionality, such as location tracking, TouchID authentication and device motion, which can only fully be tested on a physical device. Of particular

Joining the Apple iOS Developer Program

significance is the fact that some aspects of iCloud access, Reminders and In-App Purchasing can only be tested when applications are running on physical devices.

Of further significance is the fact that iOS Developer Program members have unrestricted access to the full range of guides and tutorials relating to the latest iOS SDK and, more importantly, have access to technical support from Apple's iOS technical support engineers (though the annual fee covers the submission of only two support incident reports).

By far the most important aspect of the iOS Developer Program is that membership is a mandatory requirement in order to publish an application for sale or download in the App Store.

Clearly, developer program membership is going to be required at some point before your application reaches the App Store. The only question remaining is when exactly to sign up.

2.4 When to Enroll in the iOS Developer Program?

Clearly, there are many benefits to iOS Developer Program membership and, eventually, membership will be necessary to begin selling applications. As to whether or not to pay the enrollment fee now or later will depend on individual circumstances. If you are still in the early stages of learning to develop iOS applications or have yet to come up with a compelling idea for an application to develop then much of what you need is provided in the Registered Apple Developer package. As your skill level increases and your ideas for applications to develop take shape you can, after all, always enroll in the developer program at a later date.

If, on the other hand, you are confident that you will reach the stage of having an application ready to publish or know that you will need to test the functionality of the application on a physical device as opposed to a simulator then it is worth joining the developer program sooner rather than later.

2.5 Enrolling in the iOS Developer Program

If your goal is to develop iOS applications for your employer then it is first worth checking whether the company already has membership. That being the case, contact the program administrator in your company and ask them to send you an invitation from within the iOS Developer Program Member Center to join the team. Once they have done so, Apple will send you an email entitled *You Have Been Invited to Join an Apple Developer Program* containing a link to activate your membership. If you or your company is not already a program member, you can enroll online at:

<http://developer.apple.com/programs/ios/>

Apple provides enrollment options for businesses and individuals. To enroll as an individual you will need to provide credit card information in order to verify your identity. To enroll as a company you must have legal signature authority (or access to someone who does) and be able to provide documentation such as Articles of Incorporation and a Business License.

Acceptance into the developer program as an individual member typically takes less than 24 hours with notification arriving in the form of an activation email from Apple. Enrollment as a company can take considerably longer (sometimes weeks or even months) due to the burden of the additional verification requirements.

Whilst awaiting activation you may log into the Member Center with restricted access using your Apple ID and password at the following URL:

<http://developer.apple.com/membercenter>

Once logged in, clicking on the *Your Account* tab at the top of the page will display the prevailing status of your application to join the developer program as *Enrollment Pending*:



Figure 2-1

Once the activation email has arrived, log into the Member Center again and note that access is now available to a wide range of options and resources as illustrated in Figure 2-2:

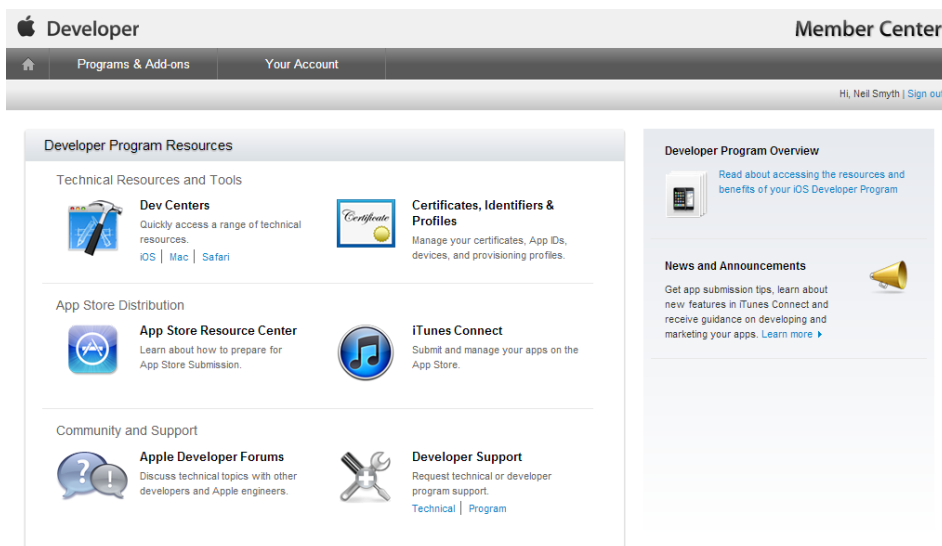


Figure 2-2

2.6 Summary

An important early step in the iOS 8 application development process involves registering as an Apple Developer and identifying the best time to upgrade to iOS Developer Program membership. This chapter has outlined the differences between the two programs, provided some guidance to keep in mind when considering developer program membership and walked briefly through the enrollment process. The next step is to download and install the iOS 8 SDK and Xcode 6 development environment.

3. Installing Xcode 6 and the iOS 8 SDK

iOS apps are developed using the iOS SDK in conjunction with Apple’s Xcode 6.x development environment. Xcode 6 is an integrated development environment (IDE) within which you will code, compile, test and debug your iOS applications. The Xcode 6 environment also includes a feature called Interface Builder which enables you to graphically design the user interface of your application using the components provided by the UIKit Framework.

In this chapter we will cover the steps involved in installing both Xcode 6 and the iOS 8 SDK on Mac OS X.

3.1 Identifying if you have an Intel or PowerPC based Mac

Only Intel based Mac OS X systems can be used to develop applications for iOS. If you have an older, PowerPC based Mac then you will need to purchase a new system before you can begin your iOS app development project. If you are unsure of the processor type inside your Mac, you can find this information by clicking on the Apple menu in the top left hand corner of the screen and selecting the *About This Mac* option from the menu. In the resulting dialog check the *Processor* line. Figure 3-1 illustrates the results obtained on an Intel based system.

If the dialog on your Mac does not reflect the presence of an Intel based processor then your current system is, sadly, unsuitable as a platform for iOS app development.

In addition, the iOS 8 SDK with Xcode 6 environment requires that the version of Mac OS X running on the system be version 10.9.4 or later. If the “About This Mac” dialog does not indicate that Mac OS X 10.9.4 or later is running, click on the *Software Update...* button to download and install the appropriate operating system upgrades.



Figure 3-1

3.2 Installing Xcode 6 and the iOS 8 SDK

The best way to obtain the latest versions of Xcode and the iOS SDK is to download them from the Apple Mac App Store. Launch the App Store on your Mac OS X system, enter Xcode into the search box and click on the *Free* button to initiate the installation.

The download is several Gigabytes in size and may take a number of hours to complete depending on the speed of your internet connection.

3.3 Starting Xcode

Having successfully installed the SDK and Xcode, the next step is to launch it so that we can create a sample iOS 8 application. To start up Xcode, open the Finder and search for *Xcode*. Since you will be making frequent use of this tool take this opportunity to drag and drop it into your dock for easier access in the future. Click on the Xcode icon in the dock to launch the tool. The first time Xcode runs you may be prompted to install additional components. Follow these steps, entering your username and password when prompted to do so.

Once Xcode has loaded, and assuming this is the first time you have used Xcode on this system, you will be presented with the *Welcome* screen from which you are ready to proceed:



Figure 3-2

Having installed the iOS 8 SDK and successfully launched Xcode 6 we can now look at Xcode 6 in more detail.

4. A Guided Tour of Xcode 6

Just about every activity related to developing and testing iOS applications involves the use of the Xcode environment. This chapter is intended to serve two purposes. Primarily it is intended to provide an overview of many of the key areas that comprise the Xcode development environment. In the course of providing this overview, the chapter will also work through the creation of a very simple iOS application project designed to display a label which reads “Hello World” on a colored background.

By the end of this chapter you will have a basic familiarity with Xcode and your first running iOS application.

4.1 Starting Xcode 6

As with all iOS examples in this book, the development of our example will take place within the Xcode 6 development environment. If you have not already installed this tool together with the latest iOS SDK refer first to the *Installing Xcode 6 and the iOS 8 SDK* chapter of this book. Assuming that the installation is complete, launch Xcode either by clicking on the icon on the dock (assuming you created one) or use the Mac OS X Finder to locate Xcode in the Applications folder of your system.

When launched for the first time, and until you turn off the *Show this window when Xcode launches* toggle, the screen illustrated in Figure 4-1 will appear by default:

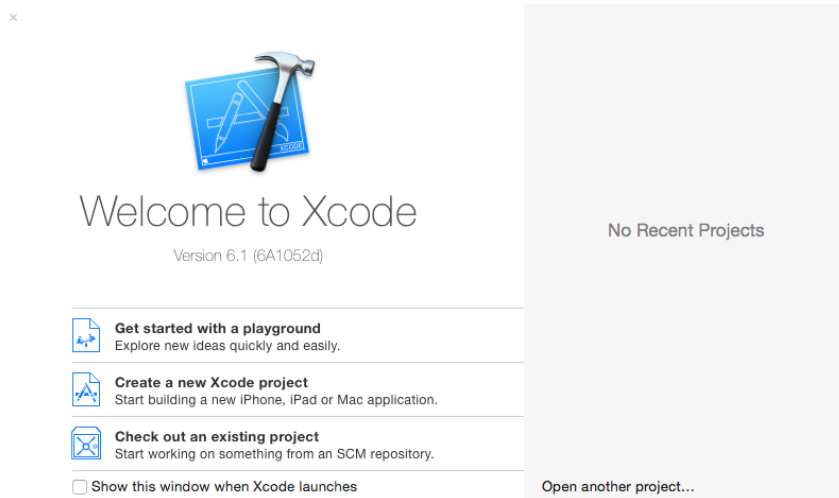


Figure 4-1

A Guided Tour of Xcode 6

If you do not see this window, simply select the *Window -> Welcome to Xcode* menu option to display it. From within this window, click on the option to *Create a new Xcode project*. This will display the main Xcode 6 project window together with the *project template* panel where we are able to select a template matching the type of project we want to develop:

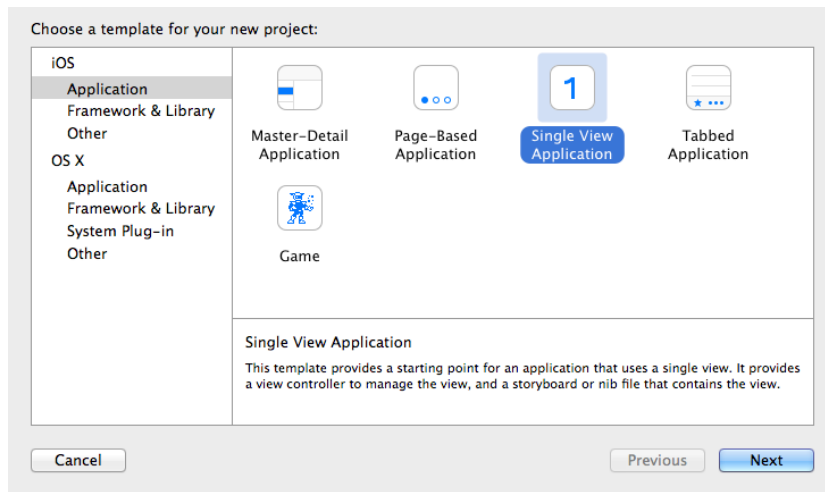


Figure 4-2

The panel located on the left hand side of the window allows for the selection of the target platform, providing options to develop an application either for iOS based devices or Mac OS X.

Begin by making sure that the *Application* option located beneath *iOS* is selected. The main panel contains a list of templates available to use as the basis for an application. The options available are as follows:

- **Master-Detail Application** – Used to create a list based application. Selecting an item from a master list displays a detail view corresponding to the selection. The template then provides a *Back* button to return to the list. You may have seen a similar technique used for news based applications, whereby selecting an item from a list of headlines displays the content of the corresponding news article. When used for an iPad based application this template implements a basic split-view configuration.
- **Page-based Application** – Creates a template project using the page view controller designed to allow views to be transitioned by turning pages on the screen.
- **Tabbed Application** – Creates a template application with a tab bar. The tab bar typically appears across the bottom of the device display and can be programmed to contain items that, when selected, change the main display to different views. The iPhone's built-in *Phone* user interface, for example, uses a tab bar to allow the user to move between favorites, contacts, keypad and voicemail.
- **Single View Application** – Creates a basic template for an application containing a single view and corresponding view controller.
- **Game** – Creates a project configured to take advantage of Sprite Kit, Scene Kit, OpenGL ES and Metal for the development of 2D and 3D games.

For the purposes of our simple example, we are going to use the *Single View Application* template so select this option from the new project window and click *Next* to configure some more project options:

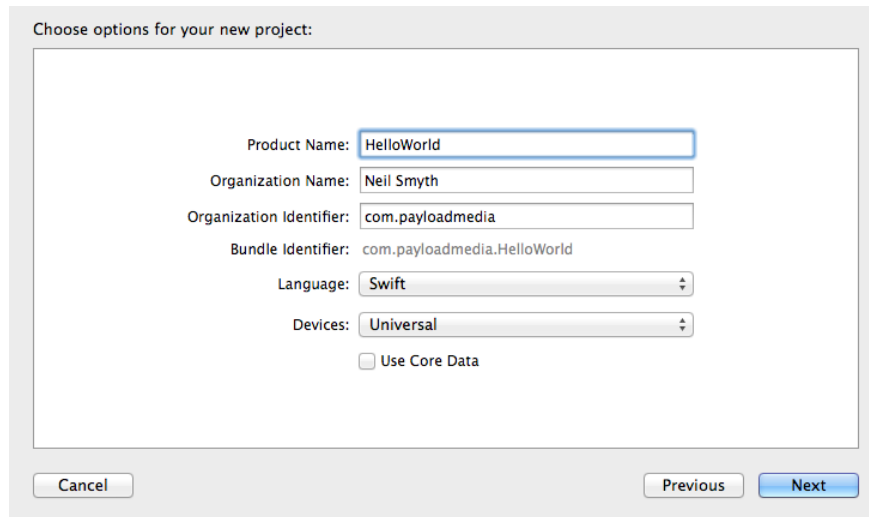


Figure 4-3

On this screen, enter a Product name for the application that is going to be created, in this case “HelloWorld”. The company identifier is typically the reversed URL of your company’s website, for example “com.mycompany”. This will be used when creating provisioning profiles and certificates to enable applications to be tested on a physical iPhone or iPad device (covered in more detail in *Testing Apps on iOS 8 Devices with Xcode 6*).

The iOS ecosystem now includes a variety of devices and screen sizes. When creating a new project it is possible to indicate that the project is intended to target either the iPhone or iPad family of devices. With the gap between iPad and iPhone screen sizes now reduced by the introduction of the iPad Mini and iPhone 6 Plus it no longer makes sense to create a project that targets just one device family. A much more sensible approach is to create a single project that addresses all device types and screen sizes. In fact, as will be shown in later chapters, Xcode 6 and iOS 8 include a number of features designed specifically to make the goal of *universal* application projects easy to achieve. With this in mind, make sure that the *Devices* menu is set to *Universal*.

Along with iOS 8 and Xcode 6, Apple has also introduced a new programming language named *Swift*. Whilst it is still possible to program using the older Objective-C language, Apple considers Swift to be the future of iOS development. All the code examples in this book are written in Swift, so make sure that the *Language* menu is set accordingly before clicking on the *Next* button.

On the final screen, choose a location on the file system for the new project to be created and click on *Create*.

Once the new project has been created, the main Xcode window will appear as illustrated in Figure 4-4:

A Guided Tour of Xcode 6

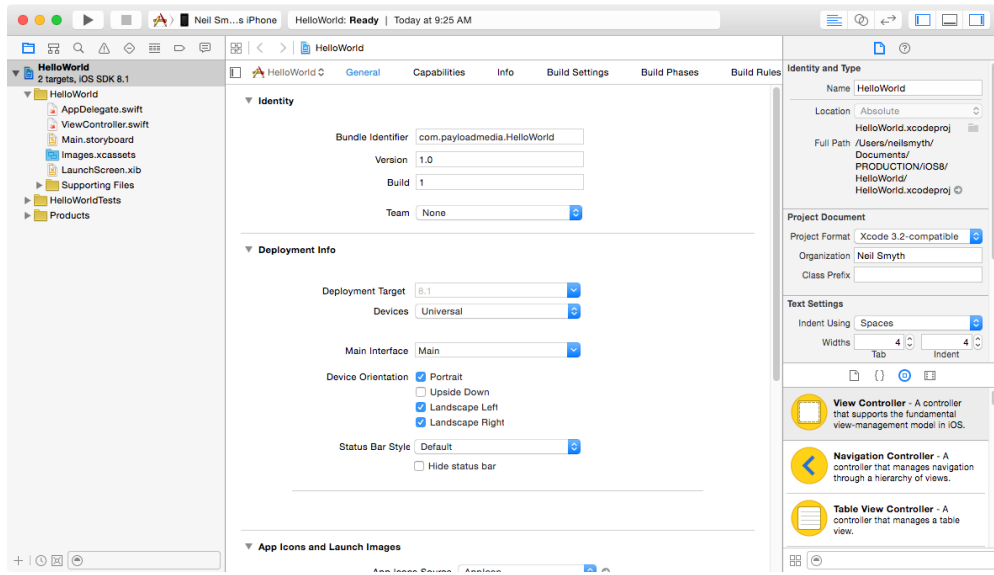


Figure 4-4

Before proceeding we should take some time to look at what Xcode has done for us. Firstly it has created a group of files that we will need to create our application. Some of these are Swift source code files (with a `.swift` extension) where we will enter the code to make our application work.

In addition, the `Main.storyboard` file is the save file used by the Interface Builder tool to hold the user interface design we will create. A second Interface builder file named `LaunchScreen.xib` will also have been added to the project. This contains the user interface layout design for the screen which appears on the device while the application is loading.

Also present will be one or more files with a `.plist` file extension. These are *Property List* files which contain key/value pair information. For example, the `Info.plist` file in the *Supporting Files* folder contains resource settings relating to items such as the language, executable name and app identifier. The list of files is displayed in the *Project Navigator* located in the left hand panel of the main Xcode project window. A toolbar at the top of this panel contains options to display other information such as build and run history, breakpoints and compilation errors.

By default, the center panel of the window shows a general summary of the settings for the application project. This includes the identifier specified during the project creation process and the target device. Options are also provided to configure the orientations of the device that are to be supported by the application together with options to upload icons (the small images the user selects on the device screen to launch the application) and launch screen images (displayed to the user while the application loads) for the application.

In addition to the General screen, tabs are provided to view and modify additional settings consisting of Capabilities, Info, Build Settings, Build Phases and Build Rules. As we progress through subsequent chapters of this book we will explore some of these other configuration options in greater detail. To return to the project settings panel at any future point in time, make sure the *Project Navigator* is selected in the left hand panel and select the top item (the application name) in the navigator list.

When a source file is selected from the list in the navigator panel, the contents of that file will appear in the center panel where it may then be edited. To open the file in a separate editing window, simply double click on the file in the list.

4.2 Creating the iOS App User Interface

Simply by the very nature of the environment in which they run, iOS apps are typically visually oriented. As such, a key component of just about any app involves a user interface through which the user will interact with the application and, in turn, receive feedback. Whilst it is possible to develop user interfaces by writing code to create and position items on the screen, this is a complex and error prone process. In recognition of this, Apple provides a tool called Interface Builder which allows a user interface to be visually constructed by dragging and dropping components onto a canvas and setting properties to configure the appearance and behavior of those components. Interface Builder was originally developed some time ago for creating Mac OS X applications, but has now been updated to allow for the design of iOS app user interfaces.

As mentioned in the preceding section, Xcode pre-created a number of files for our project, one of which has a .storyboard filename extension. This is an Interface Builder storyboard save file and the file we are interested in for our HelloWorld project is named *Main.storyboard*. To load this file into Interface Builder simply select the file name in the list in the left hand panel. Interface Builder will subsequently appear in the center panel as shown in Figure 4-5:

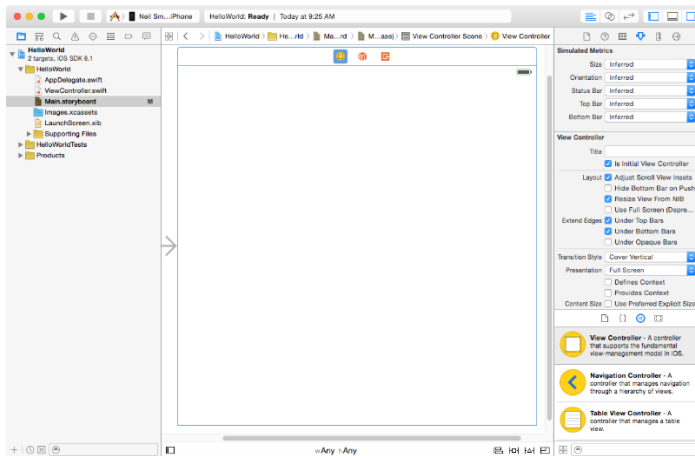


Figure 4-5

In the center panel a visual representation of the user interface of the application is displayed. Initially this consists solely of the *UIView* object. This *UIView* object was added to our design by Xcode when we selected the Single View Application option during the project creation phase. We will construct the user interface for our HelloWorld app by dragging and dropping user interface objects onto this *UIView* object. Designing a user interface consists primarily of dragging and dropping visual components onto the canvas and setting a range of properties and settings. In order to access objects and property settings it is necessary to display the Xcode right hand panel (if it is not already displayed). This panel is referred to as the *Utilities panel* and can be displayed by selecting the right hand button in the right hand section of the Xcode toolbar:



Figure 4-6

The Utilities panel, once displayed, will appear as illustrated in Figure 4-7:

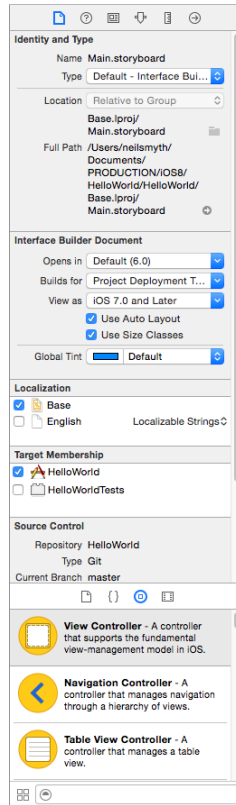


Figure 4-7

Along the top edge of the panel is a row of buttons which change the settings displayed in the upper half of the panel. By default the *File Inspector* is displayed. Options are also provided to display quick help, the *Identity Inspector*, *Attributes Inspector*, *Size Inspector* and *Connections Inspector*. Before proceeding, take some time to review each of these selections to gain some familiarity with the configuration options each provides. Throughout the remainder of this book extensive use of these inspectors will be made.

The lower section of the panel may default to displaying the file template library. Above this panel is another toolbar containing buttons to display other categories. Options include frequently used code snippets to save on typing when writing code, the Object Library and the Media Library. For the purposes of this tutorial we need to display the Object Library so click on the appropriate toolbar button (represented by the circle with a small square in the center). This will display the UI components that can be used to construct our user interface. Move the cursor to the line above the lower toolbar and click and drag to increase the amount of space available for the library if required. The layout of the items in the library may also be switched from a single column of objects with descriptions to multiple

columns without descriptions by clicking on the button located in the bottom left hand corner of the panel and to the left of the search box.

4.3 Changing Component Properties

With the property panel for the View selected in the main panel, we will begin our design work by changing the background color of this view. Start by making sure the View is selected and that the Attributes Inspector (*View -> Utilities -> Show Attributes Inspector*) is displayed in the Utilities panel. Click on the white rectangle next to the *Background* label to invoke the *Colors* dialog. Using the color selection tool, choose a visually pleasing color and close the dialog. You will now notice that the view window has changed from white to the new color selection.

4.4 Adding Objects to the User Interface

The next step is to add a Label object to our view. To achieve this, either scroll down the list of objects in the Object Library panel to locate the Label object or, as illustrated in Figure 4-8, enter *Label* into the search box beneath the panel:

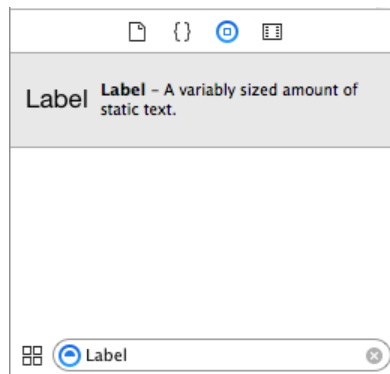


Figure 4-8

Having located the Label object, click on it and drag it to the center of the view so that the vertical and horizontal center guidelines appear. Once it is in position release the mouse button to drop it at that location. Cancel the Object Library search by clicking on the “x” button on the right hand edge of the search field. Select the newly added label and stretch it horizontally so that it is approximately three times the current width. With the Label still selected, click on the centered alignment button in the Attributes Inspector (*View -> Utilities -> Show Attributes Inspector*) to center the text in the middle of the label view.

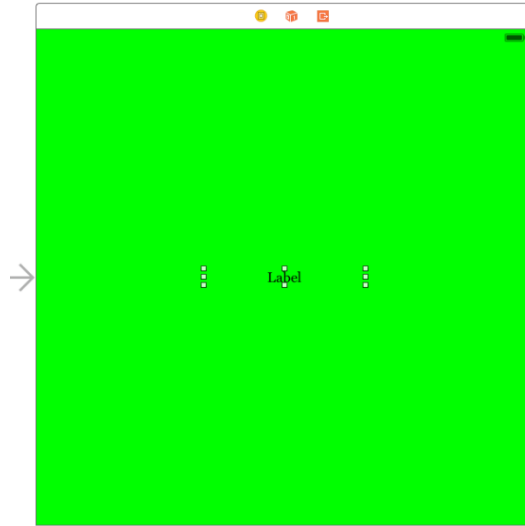


Figure 4-9

Double click on the text in the label that currently reads “Label” and type in “Hello World”. Locate the font setting property in the Attributes Inspector panel and click on the “T” button next to the font name to display the font selection menu. Change the Font setting from *System – System* to *Custom* and choose a larger font setting, for example a Georgia bold typeface with a size of 24 as shown in Figure 4-10:

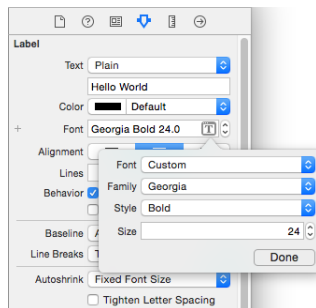


Figure 4-10

The final step is to add some layout constraints to ensure that the label remains centered within the containing view regardless of the size of screen on which the application ultimately runs. This involves the use of the Auto Layout capabilities of iOS, a topic which will be covered extensively in later chapters. For this example, simply select the Label object, display the Align menu as shown in Figure 4-11 and enable both the *Horizontal Center in Container* and *Vertical Center in Container* options before clicking on the *Add 2 Constraints* button.

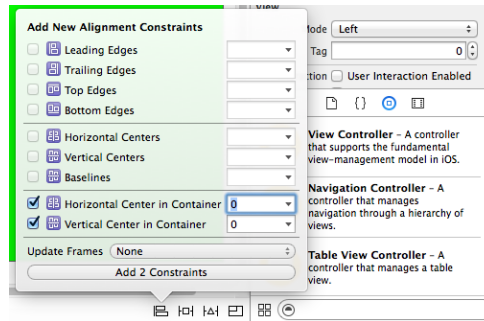


Figure 4-11

With the label still selected, display the *Pin* menu, enable the *Width* constraint and set the *Update Frames* menu to *All Frames in Container* before clicking on the *Add 1 Constraint* button. This will set the label to a specific width and update the storyboard view so that these new constraints are reflected in the layout.

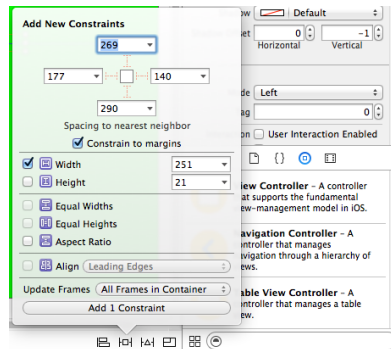


Figure 4-12

At this point, your View window will hopefully appear as outlined in Figure 4-13 (allowing, of course, for differences in your color and font choices).

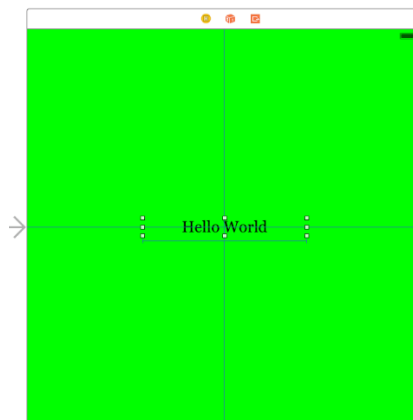


Figure 4-13

A Guided Tour of Xcode 6

Before building and running the project it is worth taking a short detour to look at the Xcode *Document Outline* panel. This panel appears by default to the left of the Interface Builder panel and is controlled by the small button in the bottom left hand corner (indicated by the arrow in Figure 4-14) of the Interface Builder panel.

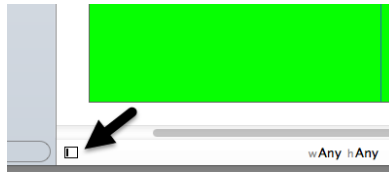


Figure 4-14

When displayed, the document outline shows a hierarchical overview of the elements that make up a user interface layout together with any constraints that have been applied to views in the layout.

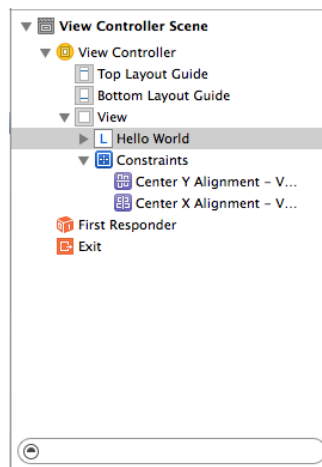


Figure 4-15

4.5 Building and Running an iOS 8 App in Xcode 6

Before an app can be run it must first be compiled. Once successfully compiled it may be run either within a simulator or on a physical iPhone, iPad or iPod Touch device. The process for testing an app on a physical device requires some additional steps to be performed involving developer certificates and provisioning profiles and will be covered in detail in *Testing Apps on iOS 8 Devices with Xcode 6*. For the purposes of this chapter, however, it is sufficient to run the app in the simulator.

Within the main Xcode 6 project window, make sure that the menu located in the top left hand corner of the window (marked C in Figure 4-16) has the *iPhone 6* simulator option selected:

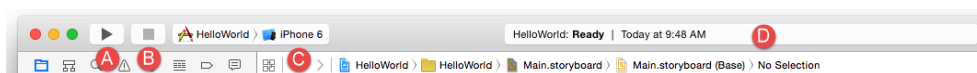


Figure 4-16

Click on the *Run* toolbar button (A) to compile the code and run the app in the simulator. The small panel in the center of the Xcode toolbar (D) will report the progress of the build process together with any problems or errors that cause the build process to fail. Once the app is built, the simulator will start and the HelloWorld app will run:



Figure 4-17

Note that the user interface appears as designed in the Interface Builder tool. Click on the stop button (B), change the target menu from iPhone 6 to iPad Air 2 and run the application again. Once again, the label will appear centered in the screen even with the larger screen size. Finally, verify that the layout is correct in landscape orientation by using the *Hardware -> Rotate Left* menu option. This indicates that the Auto Layout constraints are working and that we have designed a *universal* user interface for the project.

4.6 Dealing with Build Errors

As we have not actually written or modified any code in this chapter it is unlikely that any errors will be detected during the build and run process. In the unlikely event that something did get inadvertently changed thereby causing the build to fail it is worth taking a few minutes to talk about build errors within the context of the Xcode environment.

If for any reason a build fails, the status window in the Xcode toolbar will report that an error has been detected by displaying “Build” together with the number of errors detected and any warnings. In addition, the left hand panel of the Xcode window will update with a list of the errors. Selecting an error from this list will take you to the location in the code where corrective action needs to be taken.

4.7 Monitoring Application Performance

Another useful feature of Xcode is the ability to monitor the performance of an application while it is running. This information is accessed by displaying the *Debug Navigator*.

When Xcode is launched, the project navigator is displayed in the left hand panel by default. Along the top of this panel is a bar with a range of other options. The sixth option from the left displays the debug navigator when selected

A Guided Tour of Xcode 6

as illustrated in Figure 4-18. When displayed, this panel shows a number of real-time statistics relating to the performance of the currently running application such as memory, CPU usage, disk access, network activity and iCloud storage access.

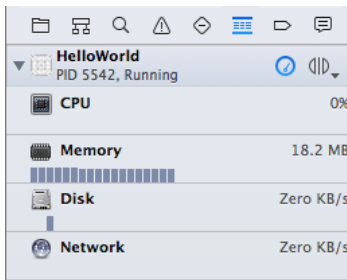


Figure 4-18

When one of these categories is selected, the main panel (Figure 4-19) updates to provide additional information about that particular aspect of the application's performance:

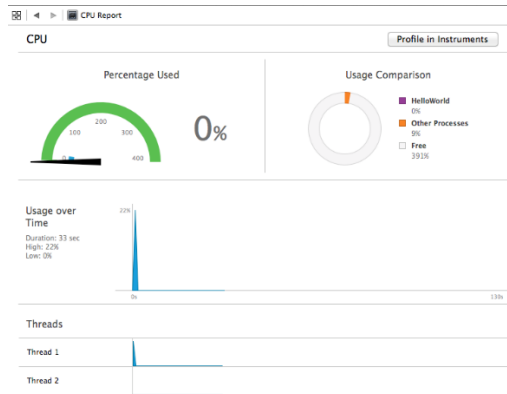


Figure 4-19

Yet more information can be obtained by clicking on the *Profile in Instruments* button in the top right hand corner of the panel.

4.8 An Exploded View of the User Interface Layout Hierarchy

Xcode 6 also provides an option to break the user interface layout out into a rotatable 3D view that shows how the view hierarchy for a user interface is constructed. This can be particularly useful for identifying situations where one view object is obscured by another appearing on top of it or a layout is not appearing as intended. To access the View Hierarchy in this mode, run the application and click on the *Debug View Hierarchy* button highlighted in Figure 4-20:



Figure 4-20

Once activated, a 3D “exploded” view of the layout will appear and may be used for design and debugging work. Figure 4-21 shows an example layout in this mode for a slightly more complex user interface than that created in this chapter:

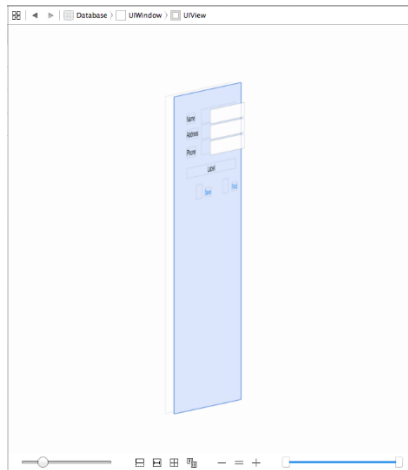


Figure 4-21

4.9 Summary

Applications are primarily created within the Xcode development environment. This chapter has served to provide a basic overview of the Xcode environment and to work through the creation of a very simple example application. Finally, a brief overview was provided of some of the performance monitoring features in Xcode 6. Many more features and capabilities of Xcode and Interface Builder will be covered in subsequent chapters of the book.

5. Testing Apps on iOS 8 Devices with Xcode 6

In the chapter entitled *A Guided Tour of Xcode 6* we were able to run an application in the iOS Simulator environment bundled with the iOS 8 SDK. Whilst this is fine for most cases, in practice there are a number of areas that cannot be comprehensively tested in the simulator. For example, no matter how hard you shake your computer (not something we actually recommend) or where in the world you move it to, neither the accelerometer nor GPS features will provide real world results within the simulator (though the simulator does have the option to perform a basic virtual shake gesture and to simulate location data). If we really want to test an iOS application thoroughly in the real world, therefore, we need to install the app onto a physical iOS device.

Many new features have been added to Xcode 6 to make the task of the developer easier. One of these features makes it considerably easier to obtain the signing certificates and provisioning profiles that are necessary to perform testing of applications on physical iOS devices.

A previous edition of this book, which was based on Xcode 4, dedicated no less than 11 pages to the process of obtaining a developer certificate, App ID and provisioning profile to test an application on a physical iOS device. Much of this work is now performed automatically by Xcode resulting in a much simpler path to testing applications on iOS devices.

5.1 Configuring Xcode with Apple IDs

The first step in setting up a fully configured development environment involves entering the Apple ID associated with your Apple Developer Program membership.

To enter this information, start Xcode and select the *Xcode -> Preferences...* menu option. From within the preferences window, select the *Accounts* tab as illustrated in Figure 5-1:

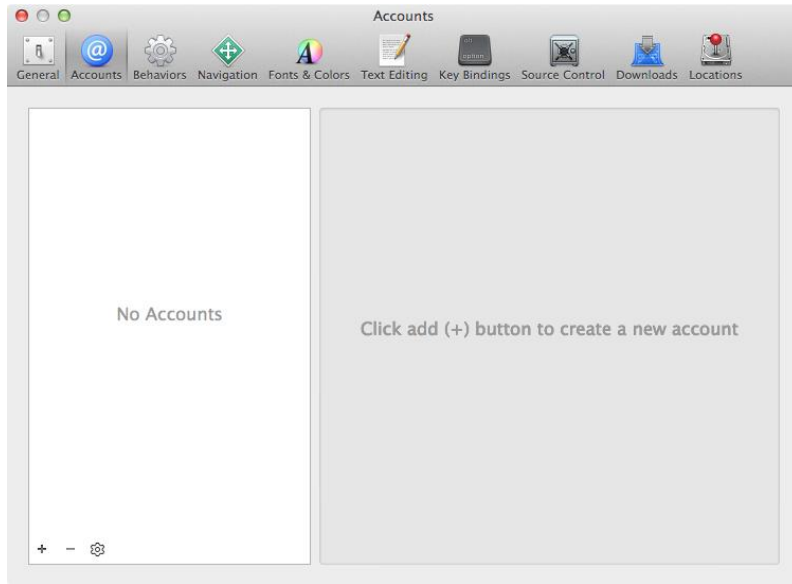


Figure 5-1

To add an Apple ID, click on the + button in the lower left hand corner and select *Add Apple ID...* from the drop down menu. When prompted to do so (Figure 5-2), either enter the Apple ID and password associated with your Apple Developer Program membership, or click on the *Join a Program...* button if you are not yet a member.

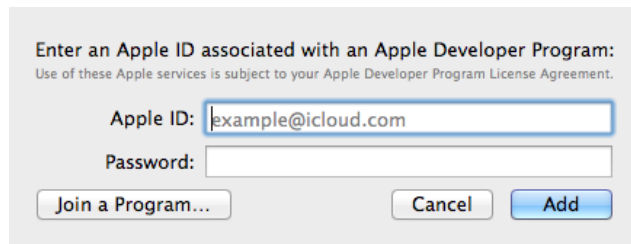


Figure 5-2

Repeat these steps to add additional Apple IDs if you are associated with more than one development team. Once the information has been entered, the accounts will be listed in the preferences window.

5.2 Generating Signing Identities

Before an application can be run on a physical iOS device for testing purposes it must first be signed with a *developer signing identity*. When the application is finished and ready to be placed on sale in the App Store it must then be signed with a *distribution signing identity*. Signing identities are comprised of a certificate and a private key.

Signing identities can be generated from within the Xcode account preferences panel. Begin by selecting the Apple ID for which the identities are to be generated before clicking on the *View Details...* button located in the lower right

hand corner of the window. This will display a list of signing identities and any provisioning profiles associated with those identities. If no valid signing identities are listed (as is the case in Figure 5-3), the next step is to generate them.

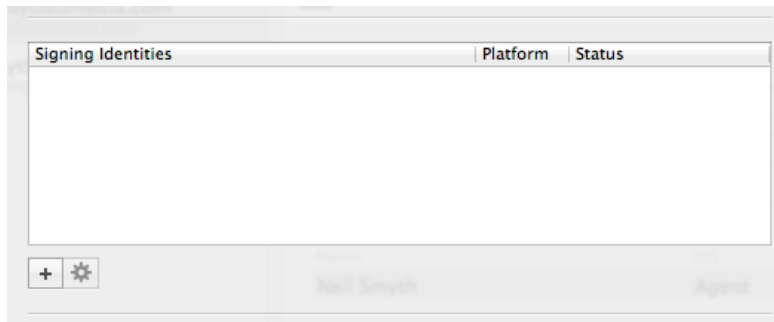


Figure 5-3

Begin by clicking on the + button and selecting the *iOS Development* option from the resulting menu. Xcode will then contact the Apple Developer Member Center portal and request and download a developer signing identity. Repeat these steps, this time selecting *iOS Distribution* from the menu to create and download a distribution signing identity. Once completed, the two identities should now be listed as shown in Figure 5-4:

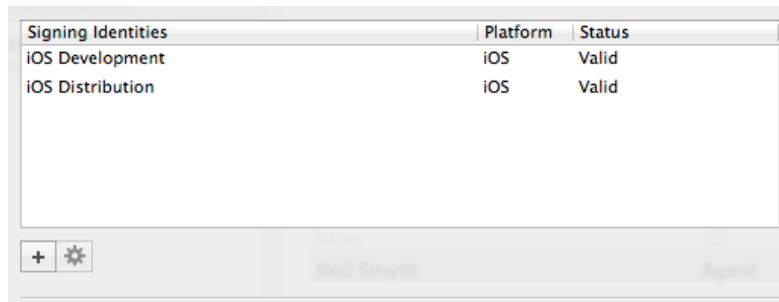


Figure 5-4

Once created, signing identities and account information can be migrated to other development computer systems by clicking on the button displaying a gear cog on the first account settings page and selecting the *Export Accounts...* menu option. On the destination system repeat these steps, this time selecting the *Import Accounts...* option.

It is worth noting that the certificates associated with the signing identities can also be viewed and created within the Apple Developer Member Center portal. Within a browser, navigate to the following URL and log in using your Apple ID credentials:

<https://developer.apple.com/membercenter>

Within the member center, click on the Certificates, Identifiers and Profiles option and choose *Certificates* from the list of options under the *iOS Apps* category. On the resulting page, the certificates for both signing identities should be listed. Clicking on a certificate will display details such as the expiration date as outlined in Figure 5-5:

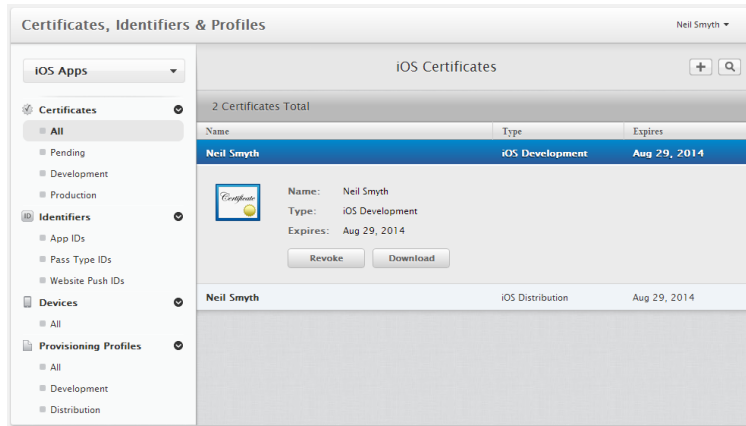


Figure 5-5

As can be seen in the left hand panel of Figure 5-5, the member center also provides options to manually create App IDs and Provisioning Profiles. With Xcode 6, however, these are typically created automatically.

5.3 Device Registration

Having generated signing identities the next step is to register a device for testing purposes. With the introduction of Xcode 6, device registration takes place automatically when an iPhone or iPad device is connected to the development system. Simply run Xcode, attach the device to the computer and wait for it to appear as an option on the *run destinations* menu. Figure 5-6, for example, shows two physical devices available for testing together with the standard iOS Simulator options:

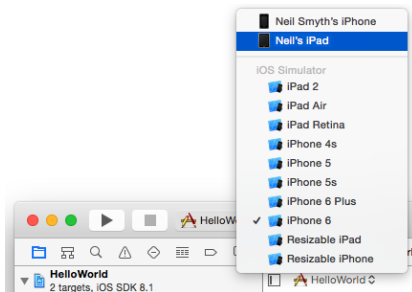


Figure 5-6

Details of the devices connected to the system can be obtained via the Xcode Devices window (*Window -> Devices*) as shown in Figure 5-7:

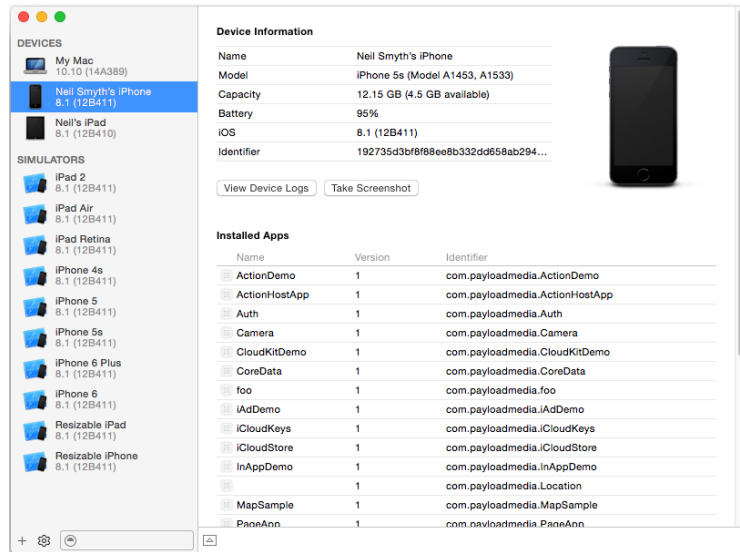


Figure 5-7

To exclude a connected device from the list of potential targets on the device scheme menu, select the device from the Devices screen and use the settings menu in the bottom left hand corner of the window to deselect the *Show in Run Destinations Menu* option:

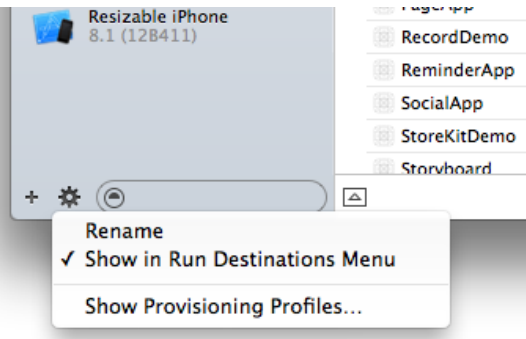


Figure 5-8

5.4 Manually Adding Test Devices

In addition to allowing Xcode to automatically register connected devices for testing purposes, it is also possible to manually enter devices from within the Developer Portal. This is a useful option when the device is not currently available to be attached to the development system for registration (perhaps it belongs to a co-worker who has volunteered to perform some app testing).

To manually register a device, the UDID of that device is required. This can be obtained from the Xcode Devices window or from within iTunes when the device is attached. To find the UDID from within iTunes, connect the device,

Testing Apps on iOS 8 Devices with Xcode 6

select it in iTunes and display the Summary screen. By default the summary screen will display the device serial number. Clicking on this number will cycle through a number of different values, eventually listing the UDID:

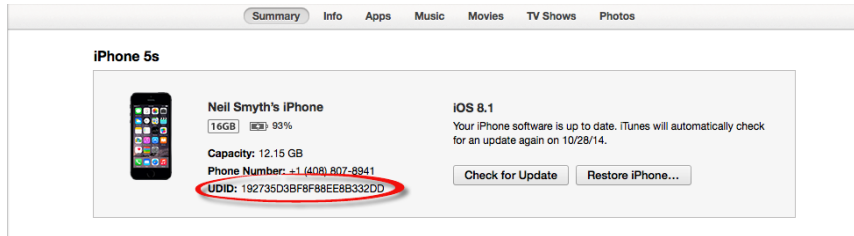
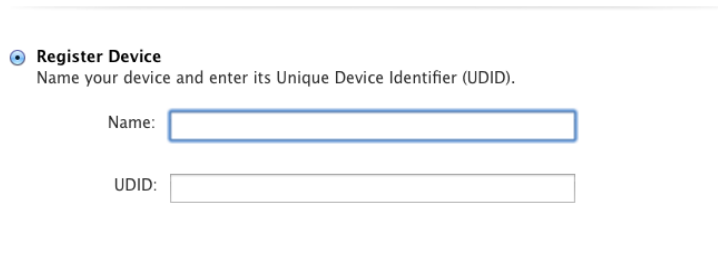


Figure 5-9

Once the UDID of the device has been obtained, log into the Apple Developer Member Center portal, select the *Certificates, Identifiers & Profiles* option and on the resulting page choose the *Devices* option listed under *iOS Apps*.

On the *iOS Devices* screen, click on the + button to add a new device and enter a name for the device and the UDID into the *Register Device* section:



• Register Device
Name your device and enter its Unique Device Identifier (UDID).

Name:

UDID:

Figure 5-10

5.5 Running an Application on a Registered Device

With a registered device connected to the development system, and an application ready for testing, refer to the device menu located in the Xcode toolbar. There is a reasonable chance that this will have defaulted to one of the iOS Simulator configurations (in the case of Figure 5-11, this is the iPhone 6 simulator).

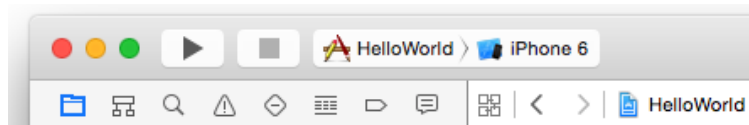


Figure 5-11

Switch to the physical device by selecting this menu and changing it to the device name as shown in Figure 5-12:

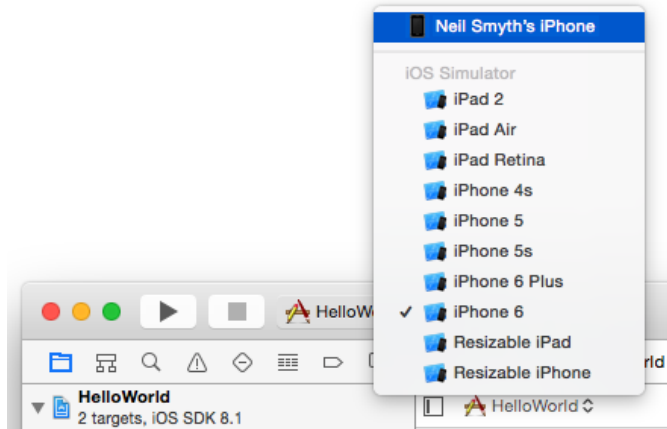


Figure 5-12

Xcode will request a provisioning profile that matches the App ID of the application and includes permission to run on the specified device, build the application using the developer signing identity before installing the application and provisioning profile on the device. Finally the application will be launched on the device.

5.6 Summary

Without question, the iOS Simulator included with the iOS 8 SDK is an invaluable tool for testing applications during the development process. There are, however, a number of situations where it is necessary to test an application on a physical iOS device. In this chapter we have covered the steps involved in provisioning applications for installation and testing on iPhone and iPad devices.

6. An Introduction to Swift Playgrounds

Along with iOS 8 and Xcode 6, Apple has introduced the new Swift programming language. Intended as a replacement for Objective-C as the basis for developing iOS apps, the significance of this new language is such that nine chapters of this book are dedicated solely to introducing the basics of Swift. In addition, all of the code examples in this book have been developed entirely using this new programming language.

Before introducing the Swift programming language in the chapters that follow, however, it is first worth learning about a feature known as *Swift playgrounds*. Playgrounds are another new feature introduced in Xcode 6 that make learning Swift and experimenting with the iOS 8 SDK much easier. The concepts covered in this chapter can be put to use when experimenting with many of the introductory Swift code examples contained in the chapters that follow.

6.1 What is a Swift Playground?

A playground is an interactive environment where Swift code can be entered and executed with the results appearing in real-time. This makes an ideal environment in which to learn the syntax of Swift without the need to work continuously through the edit/compile/run/debug cycle that would ordinarily accompany a standard Xcode iOS project.

6.2 Creating a New Swift Playground

To create a new Playground, start Xcode and select the *Get started with a playground* option from the welcome screen or select the *File -> New -> Playground* menu option. On the resulting options screen, name the playground *LearnSwift* and set the Platform menu to *iOS*. Click *Next* and choose a suitable file system location into which the playground should be saved.

Once the playground has been created, the following screen will appear ready for Swift code to be entered:

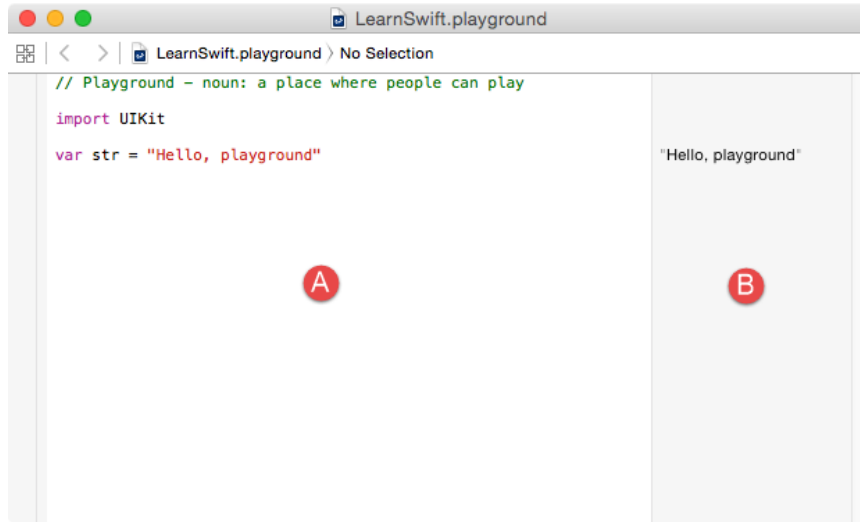


Figure 6-1

The panel on the left hand side of the window (marked A in Figure 6-1) is the *playground editor* where the lines of Swift code are entered. The right hand panel (marked B) is referred to as the *results panel* and is where the results of each Swift expression entered into the playground editor panel are displayed.

By far the quickest way to gain familiarity with the playground environment is to work through some simple examples.

6.3 A Basic Swift Playground Example

Perhaps the simplest of examples in any programming language (that at least does something tangible) is to write some code to output a single line of text. Swift is no exception to this rule so, within the playground window, begin by deleting the current Swift expression from the editor panel:

```
var str = "Hello, playground"
```

Next, enter a line of Swift code that reads as follows:

```
println("Welcome to Swift")
```

All that the code does is make a call to the built-in Swift *println* function which takes as a parameter a string of characters to be displayed on the console. Those familiar with other programming languages will note the absence of a semi-colon at the end of the line of code. In Swift, semi-colons are optional and generally only used as a separator when multiple statements occupy the same line of code.

Note that after entering the line of code, the results panel to the right of the editing panel is now showing the output from the `println` call as highlighted in Figure 6-2: