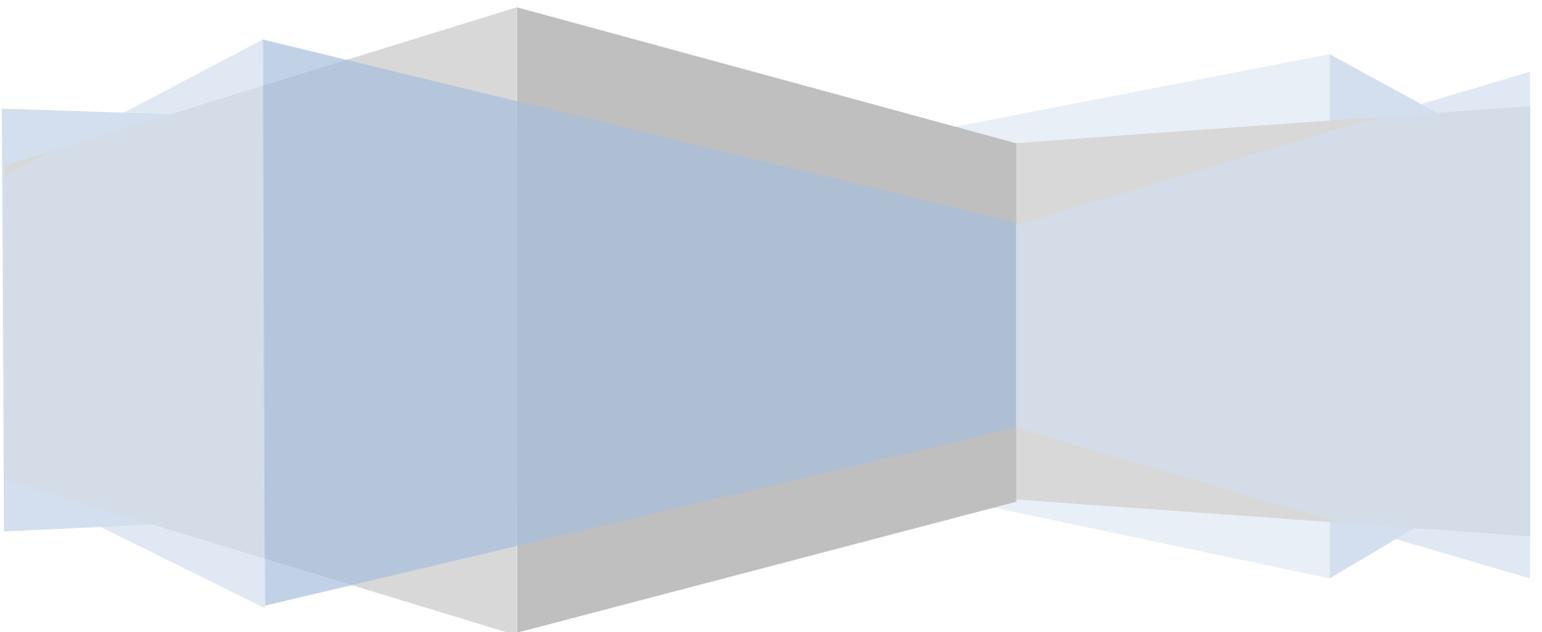


iPhone iOS 4 Development Essentials



Xcode 4 Edition

iPhone iOS 4 Development Essentials Xcode 4 Edition



iPhone iOS 4 Development Essentials – Xcode 4 Edition (Revision 1)

ISBN-13: 978-0-9832282-3-3

© 2011 Payload Media. This eBook 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.

Table of Contents

Chapter 1.	About iPhone iOS 4 App Development Essentials	22
Chapter 2.	The Anatomy of an iPhone 4	23
2.1	iOS 4	23
2.2	Display	23
2.3	Wireless Connectivity	24
2.4	Wired Connectivity	24
2.5	Memory	24
2.6	Camera	24
2.7	Sensors	24
2.8	Location Detection	25
2.9	Central Processing Unit (CPU)	25
2.10	Graphics Processing Unit (GPU).....	25
2.11	Speaker and Microphone	25
2.12	Vibration	25
2.13	Summary.....	26
Chapter 3.	iOS 4 Architecture and SDK Frameworks	27
3.1	iPhone OS becomes iOS	27
3.2	An Overview of the iOS 4 Architecture	27
3.3	The Cocoa Touch Layer	28
3.3.1	UIKit Framework (UIKit.framework)	29
3.3.2	Map Kit Framework (MapKit.framework).....	30
3.3.3	Push Notification Service	30
3.3.4	Message UI Framework (MessageUI.framework)	30
3.3.5	Address Book UI Framework (AddressUI.framework).....	30
3.3.6	Game Kit Framework (GameKit.framework).....	30
3.3.7	iAd Framework (iAd.framework)	31

3.3.8	Event Kit UI Framework	31
3.4	The iOS Media Layer	31
3.4.1	Core Video Framework (CoreVideo.framework)	31
3.4.2	Core Text Framework (CoreText.framework)	31
3.4.3	Image I/O Framework (ImageIO.framework).....	31
3.4.4	Assets Library Framework (AssetsLibrary.framework)	31
3.4.5	Core Graphics Framework (CoreGraphics.framework)	31
3.4.6	Quartz Core Framework (QuartzCore.framework).....	32
3.4.7	OpenGL ES framework (OpenGLES.framework).....	32
3.4.8	iOS Audio Support.....	32
3.4.9	AV Foundation framework (AVFoundation.framework)	32
3.4.10	Core Audio Frameworks (CoreAudio.framework, AudioToolbox.framework and AudioUnit.framework)	32
3.4.11	Open Audio Library (OpenAL).....	33
3.4.12	Media Player framework (MediaPlayer.framework).....	33
3.4.13	Core Midi Framework (CoreMIDI.framework)	33
3.5	The iOS Core Services Layer	33
3.5.1	Address Book framework (AddressBook.framework)	33
3.5.2	CFNetwork Framework (CFNetwork.framework).....	33
3.5.3	Core Data Framework (CoreData.framework)	33
3.5.4	Core Foundation Framework (CoreFoundation.framework)	33
3.5.5	Core Media Framework (CoreMedia.framework)	34
3.5.6	Core Telephony Framework (CoreTelephony.framework).....	34
3.5.7	EventKit Framework (EventKit.framework)	34
3.5.8	Foundation Framework (Foundation.framework)	34
3.5.9	Core Location Framework (CoreLocation.framework)	34
3.5.10	Mobile Core Services Framework (MobileCoreServices.framework)	34
3.5.11	Store Kit Framework (StoreKit.framework)	35

3.5.12	SQLite library	35
3.5.13	System Configuration Framework (SystemConfiguration.framework)	35
3.5.14	Quick Look Framework (QuickLook.framework)	35
3.6	The iOS Core OS Layer	35
3.6.1	Accelerate Framework (Accelerate.framework)	35
3.6.2	External Accessory framework (ExternalAccessory.framework)	36
3.6.3	Security Framework (Security.framework)	36
3.6.4	System (LibSystem)	36
Chapter 4.	Installing Xcode 4 and the iOS 4 SDK	37
4.1	Identifying if you have an Intel or PowerPC based Mac	37
4.2	Installing Xcode 4 and the iOS 4 SDK	38
4.3	Starting Xcode 4	39
Chapter 5.	Creating a Simple iPhone iOS 4 App	41
5.1	Starting Xcode 4	41
5.2	Creating the iOS App User Interface	46
5.3	Changing Component Properties.....	49
5.4	Adding Objects to the User Interface	49
5.5	Building and Running an iOS App in Xcode 4	50
5.6	Dealing with Build Errors.....	51
Chapter 6.	Testing iOS 4 Apps on the iPhone – Developer Certificates and Provisioning Profiles	53
6.1	Joining the iOS Developer Program	53
6.2	Creating an iOS Development Certificate Signing Request.....	54
6.3	Submitting the iOS Development Certificate Signing Request	56
6.4	Installing an iOS Development Certificate	58
6.5	Assigning Devices	59
6.6	Creating an App ID	61
6.7	Creating an iOS Development Provisioning Profile	62

6.8	Enabling an iPhone Device for Development	63
6.9	Associating an App ID with an App	64
6.10	iOS and SDK Version Compatibility	65
6.11	Installing an App onto a Device	66
Chapter 7. The Basics of Objective-C Programming		67
7.1	Objective-C Data Types and Variables	67
7.2	Objective-C Expressions	68
7.3	Objective-C Flow Control with <i>if</i> and <i>else</i>	72
7.4	Looping with the <i>for</i> Statement	73
7.5	Objective-C Looping with <i>do</i> and <i>while</i>	74
7.6	Objective-C <i>do ... while</i> loops	75
Chapter 8. The Basics of Object Oriented Programming in Objective-C		76
8.1	What is an Object?	76
8.2	What is a Class?	76
8.3	Declaring an Objective-C Class Interface	76
8.4	Adding Instance Variables to a Class.....	77
8.5	Define Class Methods.....	78
8.6	Declaring an Objective-C Class Implementation	79
8.7	Declaring, Initializing and Releasing a Class Instance	81
8.8	Calling Methods and Accessing Instance Data	82
8.9	Creating the Program Section	82
8.10	Bringing it all Together	83
8.11	Structuring Object-Oriented Objective-C Code	85
Chapter 9. An Overview of the iPhone iOS 4 Application Development Architecture.....		89
9.1	Model View Controller (MVC)	89
9.2	The Target-Action pattern, IBOutlets and IBActions	90
9.3	Subclassing.....	91
9.4	Delegation	91

9.5	Summary.....	92
Chapter 10.	Creating an Interactive iOS 4 iPhone App	93
10.1	Creating the New Project	93
10.2	Creating the User Interface	93
10.3	Building and Running the Sample Application	96
10.4	Adding Actions and Outlets	96
10.5	Connecting the Actions and Outlets to the User Interface	102
10.6	Building and Running the Finished Application.....	107
10.7	Summary.....	108
Chapter 11.	Writing iOS 4 Code to Hide the iPhone Keyboard	109
11.1	Creating the Example App	109
11.2	Hiding the Keyboard when the User Touches the Return Key.....	110
11.3	Hiding the Keyboard when the User Taps the Background	111
11.4	Summary.....	113
Chapter 12.	Understanding iPhone iOS 4 Views, Windows and the View Hierarchy.....	114
12.1	An Overview of Views	114
12.2	The UIWindow Class	114
12.3	The View Hierarchy	115
12.4	View Types.....	116
12.4.1	The Window.....	117
12.4.2	Container Views.....	117
12.4.3	Controls	117
12.4.4	Display Views	117
12.4.5	Text and Web Views	117
12.4.6	Navigation Views and Tab Bars	117
12.4.7	Alert Views and Action Sheets.....	117
12.5	Summary.....	118
Chapter 13.	iOS 4 iPhone Rotation, View Resizing and Layout Handling	119

13.1	Setting up the Example	119
13.2	Enabling Rotation.....	119
13.3	Testing Rotation Behavior	121
13.4	Configuring View Autosizing.....	121
13.5	Coding Layout and Size Changes	125
Chapter 14.	Creating an iOS 4 iPhone Multiview Application using the Tab Bar	130
14.1	An Overview of the Tab Bar.....	130
14.2	Understanding View Controllers in a Multiview Application	130
14.3	Setting up the Tab Bar Example Application	131
14.4	Configuring the App Delegate	131
14.5	Creating the UITabBarController	132
14.6	Connecting the App Delegate Outlet to the Tab Bar Controller	134
14.7	Creating the Content Views and View Controllers	134
14.8	Associating Content Views with Tabs	136
14.9	Designing the Content Views.....	136
14.10	Testing the Multiview Application	136
Chapter 15.	Creating a Simple iOS 4 iPhone Table View Application	138
15.1	An Overview of the Table View	138
15.2	The Table View Delegate and dataSource	138
15.3	Table View Styles	139
15.4	Table View Cell Styles.....	140
15.5	Setting up the Project	140
15.6	Adding the Table View Component	141
15.7	Making the Delegate and dataSource Connections	141
15.8	Implementing the dataSource	142
15.9	Building and Running the Application.....	145
15.10	Adding Table View Images and Changing Cell Styles	146
Chapter 16.	Creating a Navigation based iOS 4 iPhone Application using TableViews.....	150

16.1	Understanding the Navigation Controller	150
16.2	An Overview of the Example	150
16.3	Setting up the Project	151
16.4	Reviewing the Project Files.....	152
16.5	Setting up the Data in the Root View Controller	152
16.6	Writing Code to Display the Data in the Table View	154
16.7	Creating the Second View Controller	156
16.8	Connecting the Second View Controller to the Root View Controller	157
16.9	Creating the NIB File for the Second Table View.....	159
16.10	Implementing the Functionality of the Second View Controller	160
16.11	Popping the View Controller off the Navigation Controller Stack	164
16.12	Adding the Navigation Code	164
Chapter 17. Using the UIPickerView and UIDatePicker Components in iOS 4 iPhone Applications		167
17.1	The DatePicker and UIPickerView Components	167
17.2	A DatePicker Example	168
17.3	Designing the User Interface	168
17.4	Coding the Date Picker Example Functionality.....	169
17.5	Releasing Memory	171
17.6	Building and Running the iPhone Date Picker Application	171
Chapter 18. An iOS 4 iPhone UIPickerView Example		173
18.1	Creating the iOS 4 UIPickerView Project	173
18.2	UIPickerView Delegate and DataSource	173
18.3	The pickerViewController.h File	174
18.4	Designing the User Interface	175
18.5	Initializing the Arrays.....	176
18.6	Implementing the DataSource Protocol	177
18.7	Implementing the Delegate.....	177

18.8	Hiding the Keyboard.....	178
18.9	Testing the Application	179
Chapter 19.	Working with Directories on iOS 4	180
19.1	The Application Documents Directory	180
19.2	The Objective-C NSFileManager, NSFileHandle and NSData Classes	180
19.3	Understanding Pathnames in Objective-C	181
19.4	Creating an NSFileManager Instance Object.....	181
19.5	Identifying the Current Working Directory	181
19.6	Identifying the Documents Directory	182
19.7	Identifying the Temporary Directory	183
19.8	Changing Directory.....	183
19.9	Creating a New Directory	184
19.10	Deleting a Directory.....	185
19.11	Listing the Contents of a Directory.....	186
19.12	Getting the Attributes of a File or Directory.....	187
Chapter 20.	Working with Files on iOS 4.....	189
20.1	Creating an NSFileManager Instance	189
20.2	Checking if a File Exists.....	189
20.3	Comparing the Contents of Two Files	190
20.4	Checking if a File is Readable/Writable/Executable/Deletable.....	190
20.5	Moving/Renaming a File	191
20.6	Copying a File	191
20.7	Removing a File	192
20.8	Creating a Symbolic Link	192
20.9	Reading and Writing Files with NSFileManager	193
20.10	Working with Files using the NSFileHandle Class	194
20.11	Creating an NSFileHandle Object	194
20.12	NSFileHandle File Offsets and Seeking	194

20.13	Reading Data from a File.....	195
20.14	Writing Data to a File.....	196
20.15	Truncating a File	197
Chapter 21. iOS 4 iPhone Directory Handling and File I/O – A Worked Example		198
21.1	The Example iPhone Application	198
21.2	Setting up the Application project.....	198
21.3	Defining the Actions and Outlets.....	198
21.4	Designing the User Interface	199
21.5	Checking the Data File on Application Startup.....	201
21.6	Implementing the Action Method	202
21.7	Building and Running the Example	203
Chapter 22. iOS 4 iPhone Data Persistence using Archiving		205
22.1	An Overview of Archiving.....	205
22.2	The Archiving Example Application	206
22.3	Implementing the Actions and Outlets.....	206
22.4	Releasing Memory	207
22.5	Designing the iPhone User Interface	208
22.6	Checking for the Existence of the Archive File on Startup.....	209
22.7	Archiving Object Data in the Action Method	210
22.8	Testing the Application	211
22.9	Summary.....	212
Chapter 23. iOS 4 iPhone Database Implementation using SQLite		213
23.1	What is SQLite?.....	213
23.2	Structured Query Language (SQL)	213
23.3	Trying SQLite on MacOS X	214
23.4	Preparing an iPhone Application Project for SQLite Integration.....	216
23.5	Key SQLite Functions.....	216
23.6	Declaring a SQLite Database.....	217

23.7	Opening or Creating a Database.....	217
23.8	Preparing and Executing a SQL Statement.....	218
23.9	Creating a Database Table.....	219
23.10	Extracting Data from a Database Table	219
23.11	Closing a SQLite Database.....	221
23.12	Summary.....	221
Chapter 24.	An Example SQLite based iOS 4 iPhone Application.....	222
24.1	About the Example SQLite iPhone Application	222
24.2	Creating and Preparing the SQLite Application Project.....	222
24.3	Importing sqlite3.h and declaring the Database Reference	223
24.4	Creating the Outlets and Actions.....	223
24.5	Releasing Memory	225
24.6	Creating the Database and Table.....	225
24.7	Implementing the Code to Save Data to the SQLite Database	227
24.8	Implementing Code to Extract Data from the SQLite Database	228
24.9	Designing the User Interface	230
24.10	Building and Running the Application	231
24.11	Summary.....	232
Chapter 25.	Working with iOS 4 iPhone Databases using Core Data	233
25.1	The Core Data Stack	233
25.2	Managed Objects	234
25.3	Managed Object Context	234
25.4	Managed Object Model	234
25.5	Persistent Store Coordinator.....	235
25.6	Persistent Object Store	235
25.7	Defining an Entity Description	235
25.8	Obtaining the Managed Object Context	237
25.9	Getting an Entity Description	237

25.10	Creating a Managed Object	238
25.11	Getting and Setting the Attributes of a Managed Object	238
25.12	Fetching Managed Objects	239
25.13	Retrieving Managed Objects based on Criteria	239
25.14	Summary	240
Chapter 26.	An iOS 4 iPhone Core Data Tutorial	241
26.1	The iPhone Core Data Example Application.....	241
26.2	Creating a Core Data based iPhone Application.....	241
26.3	Creating the Entity Description	241
26.4	Adding a View Controller	243
26.5	Connecting the View	245
26.6	Adding Actions and Outlets to the View Controller	248
26.7	Designing the User Interface	249
26.8	Saving Data to the Persistent Store using Core Data.....	250
26.9	Retrieving Data from the Persistent Store using Core Data	251
26.10	Releasing Memory.....	252
26.11	Building and Running the Example Application	252
Chapter 27.	An Overview of iOS 4 iPhone Multitouch, Taps and Gestures	254
27.1	The Responder Chain	254
27.2	Forwarding an Event to the Next Responder	255
27.3	Gestures	255
27.4	Taps	255
27.5	Touches	255
27.6	Touch Notification Methods.....	255
27.6.1	touchesBegan method	256
27.6.2	touchesMoved method.....	256
27.6.3	touchesEnded method.....	256
27.6.4	touchesCancelled method.....	256

27.7	Summary.....	256
Chapter 28.	An Example iOS 4 iPhone Touch, Multitouch and Tap Application	257
28.1	The Example iOS iPhone Tap and Touch Application	257
28.2	Creating the Example iOS Touch Project.....	257
28.3	Creating the Outlets.....	257
28.4	Designing the user Interface	258
28.5	Enabling Multitouch on the View	259
28.6	Implementing the touchesBegan Method	260
28.7	Implementing the touchesMoved Method.....	261
28.8	Implementing the touchesEnded Method	261
28.9	Getting the Coordinates of a Touch.....	262
28.10	Building and Running the Touch Example Application	262
Chapter 29.	Detecting iOS 4 iPhone Touch Screen Gesture Motions	264
29.1	The Example iOS 4 iPhone Gesture Application	264
29.2	Creating the Example Project	264
29.3	Creating Outlets	264
29.4	Designing the Application User Interface	265
29.5	Implementing the touchesBegan Method	266
29.6	Implementing the touchesMoved Method.....	266
29.7	Implementing the touchesEnded Method.....	267
29.8	Building and Running Gesture Example	267
29.9	Summary.....	267
Chapter 30.	Drawing iOS 4 iPhone 2D Graphics with Quartz.....	268
30.1	Introducing Core Graphics and Quartz 2D	268
30.2	The drawRect Method	268
30.3	Points, Coordinates and Pixels	268
30.4	The Graphics Context.....	269
30.5	Working with Colors in Quartz 2D	269

30.6	Summary.....	271
Chapter 31.	An iOS 4 iPhone Graphics Drawing Tutorial using Quartz 2D	272
31.1	The iOS iPhone Drawing Example Application	272
31.2	Creating the New Project	272
31.3	Creating the UIView Subclass	272
31.4	Locating the drawRect Method in the UIView Subclass	273
31.5	Drawing a Line	274
31.6	Drawing Paths.....	276
31.7	Drawing a Rectangle	278
31.8	Drawing an Ellipse or Circle	278
31.9	Filling a Path with a Color.....	279
31.10	Drawing an Arc	281
31.11	Drawing a Cubic Bézier Curve	282
31.12	Drawing a Quadratic Bézier Curve	283
31.13	Dashed Line Drawing.....	284
31.14	Drawing an Image into a Graphics Context	285
Chapter 32.	Basic iPhone Animation using Core Animation	288
32.1	UIView Core Animation Blocks	288
32.2	Understanding Animation Curves.....	289
32.3	Receiving Notification of Animation Completion	289
32.4	Performing Affine Transformations	290
32.5	Combining Transformations	290
32.6	Creating the Animation Example Application	291
32.7	Implementing the Interface File	291
32.8	Drawing in the UIView	292
32.9	Detecting Screen Touches and Performing the Animation.....	292
32.10	Building and Running the Animation Application	294
32.11	Summary.....	295

Chapter 33. Integrating iAds into an iOS 4 iPhone App	296
33.1 iOS iPhone Advertising Options	296
33.2 iAds Advertisement Formats	297
33.3 Basic Rules for the Display of iAds	297
33.4 Creating an Example iAds iPhone Application.....	298
33.5 Adding the iAds Framework to the Xcode Project	298
33.6 Configuring the View Controller	298
33.7 Designing the User Interface	299
33.8 Creating the Banner Ad	300
33.9 Displaying the Ad	301
33.10 Changing Ad Format during Device Rotation	302
33.11 Implementing the Delegate Methods	303
33.11.1 bannerViewActionShouldBegin	303
33.11.2 bannerViewActionDidFinish	304
33.11.3 bannerView:didFailToReceiveAdWithError	304
Chapter 34. An Overview of iOS 4 iPhone Multitasking	305
34.1 Understanding iOS Application States	305
34.2 A Brief Overview of the Multitasking Application Lifecycle.....	306
34.3 Disabling Multitasking for an iOS Application	307
34.4 Checking for Multitasking Support	307
34.5 Supported Forms of Background Execution.....	308
34.6 The Rules of Background Execution.....	309
34.7 Scheduling Local Notifications.....	310
Chapter 35. Scheduling iOS 4 iPhone Local Notifications	311
35.1 Creating the Local Notification iPhone App Project	311
35.2 Locating the Application Delegate Method	311
35.3 Adding a Sound File to the Project	312
35.4 Scheduling the Local Notification	312

35.5	Testing the Application	313
35.6	Cancelling Scheduled Notifications.....	314
35.7	Immediate Triggering of a Local Notification	315
35.8	Summary.....	315
Chapter 36. Getting iPhone Location Information using the iOS 4 Core Location Framework		316
36.1	The Basics of Core Location	316
36.2	Configuring the Desired Location Accuracy	316
36.3	Configuring the Distance Filter	317
36.4	The Location Manager Delegate.....	317
36.5	Obtaining Location Information from CLLocation Objects	318
36.5.1	Longitude and Latitude	318
36.5.2	Accuracy	318
36.5.3	Altitude.....	319
36.6	Calculating Distances	319
36.7	Location Information and Multitasking	319
36.8	Summary.....	319
Chapter 37. An Example iOS 4 iPhone Location Application.....		320
37.1	Creating the Example iOS iPhone Location Project	320
37.2	Adding the Core Location Framework to the Project	320
37.3	Configuring the View Controller	320
37.4	Designing the User Interface	321
37.5	Creating the CLLocationManager Object	323
37.6	Implementing the Action Method	323
37.7	Implementing the Application Delegate Methods	324
37.8	Releasing Memory	326
37.9	Building and Running the iPhone Location Application.....	327
Chapter 38. Accessing the iPhone Camera and Photo Library.....		329
38.1	The iOS 4 UIImagePickerController Class.....	329

38.2	Creating and Configuring a UIImagePickerController Instance.....	329
38.3	Configuring the UIImagePickerController Delegate	330
38.4	Detecting Device Capabilities	332
38.5	Saving Movies and Images	333
38.6	Summary.....	334
Chapter 39.	An Example iOS 4 iPhone Camera Application.....	335
39.1	An Overview of the Application.....	335
39.2	Creating the Camera Project	335
39.3	Adding Framework Support	335
39.4	Configuring Protocols, Outlets and Actions	335
39.5	Designing the User Interface	336
39.6	Implementing the Action Methods.....	337
39.7	Writing the Delegate Methods.....	339
39.8	Releasing Memory	341
39.9	Building and Running the Application.....	341
Chapter 40.	Video Playback from within an iOS 4 iPhone Application.....	344
40.1	An Overview of the MPMoviePlayerController Class	344
40.2	Supported Video Formats	344
40.3	The iPhone Movie Player Example Application.....	344
40.4	Adding the MediaPlayer Framework to the Project.....	345
40.5	Declaring the Action Method	345
40.6	Designing the User Interface	345
40.7	Adding the Video File to the Project Resources	345
40.8	Implementing the Action Method	346
40.9	The Target-Action Notification Method.....	347
40.10	Enabling Device Rotation.....	347
40.11	Build and Run the Application.....	348
40.12	Accessing a Network based Video File	348

Chapter 41. Playing Audio on an iPhone using AVAudioPlayer	349
41.1 Support Audio Formats	349
41.2 Receiving Playback Notifications	349
41.3 Controlling and Monitoring Playback	350
41.4 Creating the iPhone Audio Example Application.....	350
41.5 Adding the AVFoundation Framework	351
41.6 Adding an Audio File to the Project Resources	351
41.7 Creating Actions and Outlets.....	351
41.8 Implementing the Action Methods.....	352
41.9 Creating Initializing the AVAudioPlayer Object	352
41.10 Implementing the AVAudioPlayerDelegate Protocol Methods.....	353
41.11 Designing the User Interface	354
41.12 Releasing Memory.....	355
41.13 Building and Running the Application	356
Chapter 42. Recording Audio on an iPhone with AVAudioRecorder.....	357
42.1 An Overview of the iPhone AVAudioRecorder Tutorial.....	357
42.2 Creating the Recorder Project	357
42.3 Declarations, Actions and Outlets	357
42.4 Creating the AVAudioRecorder Instance	358
42.5 Implementing the Action Methods.....	360
42.6 Implementing the Delegate Methods.....	362
42.7 Designing the User Interface	363
42.8 Releasing Memory	364
42.9 Testing the Application	364
Chapter 43. Detecting when an iPhone Headphone or Docking Connector is Unplugged	365
43.1 Detecting a Change to the Audio Hardware Route	365
43.2 An Example iPhone Headphone and Dock Connector Detection Application.....	365
43.3 Adding the AudioToolBox Framework to the Project.....	366

43.4	Configuring the Property Listener	366
43.5	Writing the Property Listener Callback	367
43.6	Testing the Application	371

Chapter 1. About iPhone iOS 4 App Development Essentials

The iPhone and its peers in the smartphone market are remarkable technological achievements. In a device small enough to put in your pocket the iPhone can make phone calls, send and receive email, SMS and MMS messages, stream and play audio and video, detect movement and rotation, vibrate, adapt the display brightness based on the ambient lighting, surf the internet, run apps from a selection of hundreds of thousands, take high resolution photos, record video, tell you your exact location, provide directions to your chosen destination, play graphics intensive games and even detect when you put the device to your ear.

Perhaps the most amazing thing about the iPhone is that all of these capabilities and hardware features are available to you as an app developer. In fact, once you have an iPhone, an Intel-based Mac computer, the iOS SDK, a copy of the Xcode 4 development environment and the necessary skills, the only limit to the types of apps you can create is your own imagination (and, of course, the restrictions placed on apps accepted into the Apple App Store).

Beginning with the basics, this book provides an overview of the iPhone hardware and the architecture of iOS 4. An introduction to programming in Objective-C is provided followed by an in-depth look at the design of iPhone applications and user interfaces. More advanced topics such as file handling, database management, graphics drawing and animation are also covered, as are touch screen handling, multitasking, iAds integration, location management, local notifications, camera access and video playback support.

The aim of this book, therefore, is to teach you the skills necessary to build your own apps for the iPhone. Assuming you have downloaded the iOS SDK and Xcode 4, have an Intel-based Mac (sadly this book does not include one) and some ideas for some apps to develop, you are ready to get started.

1.1 Example Source Code

The source code and Xcode 4 project files for the examples contained in this book are available for download at http://www.ebookfrenzy.com/book_examples/iPhoneiOS4XC4.zip.

Chapter 2. The Anatomy of an iPhone 4

Most books covering the development of apps for the iPhone tend to overlook the underlying hardware of the device and instead dive immediately into the software development environment. This is a shame because the iPhone is an incredible technical achievement that we are already starting to take for granted.

Take, for example, the iPhone 4. This is a sleek device that is 115.2mm long, 58.6.1mm wide and 9.3 mm deep and weighs a mere 137 grams. Now, compare the size of your laptop or desktop computer to your iPhone. Then take a look at the specification for your computer and see if it has built in GPRS, EDGE and 3G wireless support, a digital compass, GPS, an accelerometer, a gyroscope, a proximity sensor, an ambient light sensor, Bluetooth capability, Wi-Fi, a multi-touch screen, a vibration generator and a 5 megapixel autofocus camera with built in flash and a second, 30 frame per second front facing camera. The chances are your much larger and heavier computer has only a small subset of these features. Next, check the expected battery life of your laptop and see if will allow you to play music for 40 hours or video for 10 hours, or talk non-stop to a friend for 14 hours without needing a recharge. When you consider these capabilities you will hopefully begin to appreciate the engineering achievements behind the iPhone and other similar smartphone devices.

Now that we have set the scene, we can move on to discuss some of the hardware features built into the iPhone in a little more detail. Once again, we will do this within the context of the iPhone 4.

2.1 iOS 4

Before we delve into the hardware of the iPhone we will start by talking about the operating system that sits on top of all the hardware. This operating system is called iOS 4 and is a variant of Apple's Mac OS X operating system that has been adapted to run on the iPhone. It is built upon a "UNIX-like" foundation called Darwin and consists of the Mach kernel, core services and media layers and the Cocoa Touch interface. iOS 4 is covered in greater detail in the chapter entitled [iOS 4 Architecture and SDK Frameworks](#).

2.2 Display

The iPhone 4 has a 3.5 inch display with a resolution of 960 x 640 pixels capable of displaying 326 pixels per inch (ppi) with an 800:1 contrast ratio. The underlying technology is an In Plane Switching (IPS) LED, capacitive touch screen. The screen has a scratch and oil and fingerprint resistant oleophobic coated surface and includes a proximity sensor that automatically turns off the screen when you put the phone to your ear (presumably to extend the battery life during a phone call and to avoid making user interface selections with your ear or the side of your face).

The device also has ambient light detection that adjusts the screen brightness to ensure the optimal screen visibility in a variety of lighting conditions from bright sunlight to darkness.

2.3 Wireless Connectivity

The iPhone 4 supports a wide range of connectivity options. When within range of a Wi-Fi network, the device can connect at either 802.11b, 802.11g or 802.11n speeds.

For making phone calls or transferring data when not connected to Wi-Fi, the AT&T device supports GSM/EDGE connectivity (otherwise known as 2G). For faster speeds, support is also provided for connectivity via Universal Mobile Telecommunications System (UMTS), High-Speed Downlink Packet Access (HSDPA) and High Speed Uplink Packet Access (HSUPA). This is better known as 3G and provides data transfer speeds of up to 7.2 megabits per second.

The iPhone 4 also includes Bluetooth v2.1 support with Enhanced Data Rate (EDR) technology.

2.4 Wired Connectivity

Given the wide array of wireless options it is not surprising that the iPhone has little need for wired connections. In fact the iPhone only has two. One is a standard 3.5 mm headset jack for the attachment of headphones or other audio devices. The second is a proprietary, 30-pin dock connector that, by default, is used to provide a USB v2.0 connection for synching with a computer system and battery charging. In practice, however, this connection also provides audio and TV output via specialty third party cables.

2.5 Memory

The iPhone 4 comes in two editions, one with containing 16GB of memory and another with 32GB. The memory is in the form of a flash drive. Unlike some devices, the iPhone lacks the ability to supplement the installed memory by inserting additional flash memory cards.

2.6 Camera

The iPhone 4 contains a 5 megapixel autofocus still camera that may also be used to record video at an HD resolution of 720p at a rate of 30 frames per second (fps). In addition, the device also incorporates an LED flash and a VGA resolution, 30 fps front facing camera.

2.7 Sensors

The latest generation of iPhone has an array of sensors that would make even the most die-hard 1960s science fiction fan jealous. These consist of a proximity sensor that detects when the front of the phone is covered or otherwise obscured, an accelerometer that uses the pull of gravity to detect when the device is moved or rotated, a three-axis gyroscope and an ambient light sensor that detects current environmental light levels.

2.8 Location Detection

The iPhone 4 contains a digital compass and GPS support with Assisted GPS (A-GPS) support. Essentially this enables the iPhone to detect the direction the device is facing and to identify the current location by detecting radio signals from GPS satellites. In the event that GPS signals are unavailable or too weak to establish the current coordinates, the iPhone can also gain an approximate location using cellular and Wi-Fi information.

2.9 Central Processing Unit (CPU)

The central processing unit (CPU) of the iPhone 4 is the Apple A4, an Apple designed system-on-a-chip (SoC) consisting of an ARM Cortex A8 chip combined with an Imagination Technologies PowerVR Graphics Processing Unit (GPU). This Cortex A8 processor is designed by a British company called ARM Holdings that specializes in designing chips and then licensing those designs to third parties who then manufacture them. This differs considerably from the approach taken by companies such as Intel who both design and manufacture their own chips.

The Cortex A8 chip is based on the ARMv7 processor architecture and was chosen by Apple for its combination of high performance and low power requirements.

2.10 Graphics Processing Unit (GPU)

As previously mentioned, iPhone 4 graphics are handled by an Imagination Technologies PowerVR SGX 535 Graphics Processing Unit (GPU). This is also the same GPU that is built into the iPad and provides support for OpenGL ES 1.1/2.0 (a lightweight version of SGI's OpenGL platform), OpenGL 2.0/3.0 and OpenVG 1.1 and DirectX 9/10.1 graphics drawing and manipulation and includes the Universal Scalable Shader Engine (USSE), all key requirements for graphics intensive games development.

The older iPhone 3G contains the PowerVR MBX GPU which only supports OpenGL ES 1.1 and OpenVG 1.0.

2.11 Speaker and Microphone

As with most other phones on the market, the iPhone includes both a built-in microphone and a speaker to enable the use of the device as a speakerphone. Both the speaker and microphone may be used by third party apps, though as is to be expected with a device the size of an iPhone, the sound quality of the speaker is widely considered to be poor.

2.12 Vibration

Though initially provided as a “silent ring” feature whereby the device vibrates to indicate an incoming call as an alternative to a ring tone (a feature common to most mobile phone

devices), the vibration feature of the iPhone may also be used within applications to notify the user of a new event (such as a breaking news story) or to provide tactile feedback such as for an explosion in a game.

2.13 Summary

As we have seen in this chapter, the iPhone packs an impressive amount of technology into a very small amount of space. Perhaps the most exciting aspect of all this technology is that you can, almost without exception, access and utilize all this hardware within your own applications.

Chapter 3. iOS 4 Architecture and SDK Frameworks

In [The Anatomy of an iPhone 4](#) we looked at the hardware that is contained within an iPhone 4 device. When we develop apps for the iPhone Apple does not allow us direct access to any of this hardware. In fact, all hardware interaction takes place exclusively through a number of different layers of software that act as intermediaries between the application code and device hardware. These layers make up what is known as an *operating system*. In the case of the iPhone, this operating system is known as iOS.

In order to gain a better understanding of the iPhone development environment, this chapter will look in detail at the different layers that comprise the iOS operating system and the frameworks that allow us, as developers, to write iPhone applications.

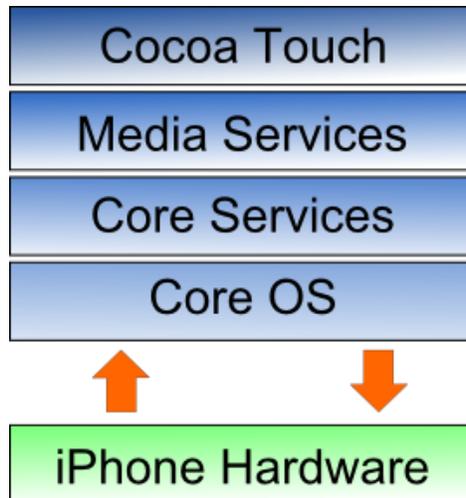
3.1 iPhone OS becomes iOS

Prior to the release of the iPad in 2010, the operating system running on the iPhone was referred to as *iPhone OS*. Given that the operating system used for the iPad is essentially the same as that on the iPhone it didn't make much sense to name it *iPad OS*. Instead, Apple decided to adopt a more generic and non-device specific name for the operating system. Given Apple's predilection for names prefixed with the letter 'i' (iTunes, iBookstore, iMac etc) the logical choice was, of course, *iOS*. Unfortunately, iOS is also the name used by Cisco for the operating system on its routers (Apple, it seems, also has a predilection for ignoring trademarks). When performing an internet search for iOS, therefore, be prepared to see large numbers of results for Cisco's iOS which have absolutely nothing to do with Apple's iOS.

3.2 An Overview of the iOS 4 Architecture

As previously mentioned, iOS consists of a number of different software layers, each of which provides programming frameworks for the development of applications that run on top of the underlying hardware.

These operating system layers can be presented diagrammatically as illustrated in the following figure:



Some diagrams designed to graphically depict the iOS software stack show an additional box positioned above the Cocoa Touch layer to indicate the applications running on the device. In the above diagram we have not done so since this would suggest that the only interface available to the app is Cocoa Touch. In practice, an app can directly call down any of the layers of the stack to perform tasks on the physical device.

That said, however, each operating system layer provides an increasing level of abstraction away from the complexity of working with the hardware. As an iOS developer you should, therefore, always look for solutions to your programming goals in the frameworks located in the higher level iOS layers before resorting to writing code that reaches down to the lower level layers. In general, the higher level of layer you program to, the less effort and fewer lines of code you will have to write to achieve your objective. And as any veteran programmer will tell you, the less code you have to write the less opportunity you have to introduce bugs.

Now that we have identified the various layers that comprise iOS 4 we can now look in more detail at the services provided by each layer and the corresponding frameworks that make those services available to us as application developers.

3.3 The Cocoa Touch Layer

The Cocoa Touch layer sits at the top of the iOS stack and contains the frameworks that are most commonly used by iPhone application developers. Cocoa Touch is primarily written in Objective-C, is based on the standard Mac OS X Cocoa API (as found on Apple desktop and laptop computers) and has been extended and modified to meet the needs of the iPhone.

The Cocoa Touch layer provides the following frameworks for iPhone app development:

3.3.1 UIKit Framework (UIKit.framework)

The UIKit framework is a vast and feature rich Objective-C based programming interface. It is, without question, the framework with which you will spend most of your time working. Entire books could, and probably will, be written about the UIKit framework alone. Some of the key features of UIKit are as follows:

- User interface creation and management (text fields, buttons, labels, colors, fonts etc)
- Application lifecycle management
- Application event handling (e.g. touch screen user interaction)
- Multitasking
- Wireless Printing
- Data protection via encryption
- Cut, copy, and paste functionality
- Web and text content presentation and management
- Data handling
- Inter-application integration
- Push notification in conjunction with Push Notification Service
- Local notifications (a mechanism whereby an application running in the background can gain the user's attention)
- Accessibility
- Accelerometer, battery, proximity sensor, camera and photo library interaction.
- Touch screen gesture recognition
- File sharing (the ability to make application files stored on the device available via iTunes)
- Blue tooth based peer to peer connectivity between devices
- Connection to external displays

To get a feel for the richness of this framework it is worth spending some time browsing Apple's UIKit reference material which is available online at:

http://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIKit_Framework/index.html

3.3.2 Map Kit Framework ([MapKit.framework](#))

If you have spent any appreciable time with an iPhone then the chances are you have needed to use the Maps application more than once, either to get a map of a specific area or to generate driving directions to get you to your intended destination. The Map Kit framework provides a programming interface that enables you to build map based capabilities into your own applications. This allows you to, amongst other things, display scrollable maps for any location, display the map corresponding to the current geographical location of the device and annotate the map in a variety of ways.

3.3.3 Push Notification Service

The Push Notification Service allows applications to notify users of an event even when the application is not currently running on the device. Since the introduction of this service it has most commonly been used by news based applications. Typically when there is breaking news the service will generate a message on the device with the news headline and provide the user the option to load the corresponding news app to read more details. This alert is typically accompanied by an audio alert and vibration of the device. This feature should be used sparingly to avoid annoying the user with frequent interruptions.

3.3.4 Message UI Framework ([MessageUI.framework](#))

The Message UI framework provides everything you need to allow users to compose and send email messages from within your application. In fact, the framework even provides the user interface elements through which the user enters the email addressing information and message content. Alternatively, this information can be pre-defined within your application and then displayed for the user to edit and approve prior to sending.

3.3.5 Address Book UI Framework ([AddressUI.framework](#))

Given that a key function of the iPhone is as a communications device and digital assistant it should not come as too much of a surprise that an entire framework is dedicated to the integration of the address book data into your own applications. The primary purpose of the framework is to enable you to access, display, edit and enter contact information from the iPhone address book from within your own application.

3.3.6 Game Kit Framework ([GameKit.framework](#))

The Game Kit framework provides peer-to-peer connectivity and voice communication between multiple devices and users allowing those running the same app to interact. When this feature was first introduced it was anticipated by Apple that it would primarily be used in multi-player games (hence the choice of name) but the possible applications for this feature clearly extend far beyond games development.

3.3.7 iAd Framework (`iAd.framework`)

The purpose of the iAd Framework is to allow developers to include banner advertising within their applications. All advertisements are served by Apple's own ad service.

3.3.8 Event Kit UI Framework

The Event Kit UI framework was introduced in iOS 4 and is provided to allow the calendar events to be accessed and edited from within an application.

3.4 The iOS Media Layer

The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities. As with the other layers comprising the iOS stack, the Media layer comprises a number of frameworks that may be utilized when developing iPhone apps. In this section we will look at each one in turn.

3.4.1 Core Video Framework (`CoreVideo.framework`)

A new framework introduced with iOS 4 to provide buffering support for the Core Media framework. Whilst this may be utilized by application developers it is typically not necessary to use this framework.

3.4.2 Core Text Framework (`CoreText.framework`)

The iOS Core Text framework is a C-based API designed to ease the handling of advanced text layout and font rendering requirements.

3.4.3 Image I/O Framework (`ImageIO.framework`)

The Image IO framework, the purpose of which is to facilitate the importing and exporting of image data and image metadata, was introduced in iOS 4. The framework supports a wide range of image formats including PNG, JPEG, TIFF and GIF.

3.4.4 Assets Library Framework (`AssetsLibrary.framework`)

The Assets Library provides a mechanism for locating and retrieving video and photo files located on the iPhone device. In addition to accessing existing images and videos, this framework also allows new photos and videos to be saved to the standard device photo album.

3.4.5 Core Graphics Framework (`CoreGraphics.framework`)

The iOS Core Graphics Framework (otherwise known as the Quartz 2D API) provides a lightweight two dimensional rendering engine. Features of this framework include PDF document creation and presentation, vector based drawing, transparent layers, path based drawing, anti-aliased rendering, color manipulation and management, image rendering and

gradients. Those familiar with the Quartz 2D API running on MacOS X will be pleased to learn that the implementation of this API is the same on iOS.

3.4.6 Quartz Core Framework (`QuartzCore.framework`)

The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone. It provides the foundation for the majority of the visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.

3.4.7 OpenGL ES framework (`OpenGLES.framework`)

For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL. Originally developed by the now defunct Silicon Graphics, Inc (SGI) during the 1990s in the form of GL, the open version of this technology (OpenGL) is now under the care of a non-profit consortium comprising a number of major companies including Apple, Inc., Intel, Motorola and ARM Holdings.

OpenGL for Embedded Systems (ES) is a lightweight version of the full OpenGL specification designed specifically for smaller devices such as the iPhone.

iOS 3 or later supports both OpenGL ES 1.1 and 2.0 on certain iPhone models (such as the iPhone 3GS and iPhone 4). Earlier versions of iOS and older device models support only OpenGL ES version 1.1.

3.4.8 iOS Audio Support

iOS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, μ -law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.

3.4.9 AV Foundation framework (`AVFoundation.framework`)

An Objective-C based framework designed to allow the playback, recording and management of audio content.

3.4.10 Core Audio Frameworks (`CoreAudio.framework`, `AudioToolbox.framework` and `AudioUnit.framework`)

The frameworks that comprise Core Audio for iOS define supported audio types, playback and recording of audio files and streams and also provide access to the device's built-in audio processing units.

3.4.11 Open Audio Library (OpenAL)

OpenAL is a cross platform technology used to provide high-quality, 3D audio effects (also referred to as positional audio). Positional audio can be used in a variety of applications though is typically using to provide sound effects in games.

3.4.12 Media Player framework (MediaPlayer.framework)

The iOS Media Player framework is able to play video in .mov, .mp4, .m4v, and .3gp formats at a variety of compression standards, resolutions and frame rates.

3.4.13 Core Midi Framework (CoreMIDI.framework)

Introduced in iOS 4, the Core MIDI framework provides an API for applications to interact with MIDI compliant devices such as synthesizers and keyboards via the iPhone's dock connector.

3.5 The iOS Core Services Layer

The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built and consists of the following frameworks.

3.5.1 Address Book framework (AddressBook.framework)

The Address Book framework provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.

3.5.2 CFNetwork Framework (CFNetwork.framework)

The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets. This enables application code to be written that works with HTTP, FTP and Domain Name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

3.5.3 Core Data Framework (CoreData.framework)

This framework is provided to ease the creation of data modeling and storage in Model-View-Controller (MVC) based applications. Use of the Core Data framework significantly reduces the amount of code that needs to be written to perform common tasks when working with structured data in an application.

3.5.4 Core Foundation Framework (CoreFoundation.framework)

The Core Foundation is a C-based Framework that provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication. Additional XML capabilities beyond those included with this framework are provided via the libXML2

library. Though this is a C-based interface, most of the capabilities of the Core Foundation framework are also available with Objective-C wrappers via the Foundation Framework.

3.5.5 Core Media Framework (`CoreMedia.framework`)

The Core Media framework is the lower level foundation upon which the AV Foundation layer is built. Whilst most audio and video tasks can, and indeed should, be performed using the higher level AV Foundation framework, access is also provided for situations where lower level control is required by the iOS application developer.

3.5.6 Core Telephony Framework (`CoreTelephony.framework`)

The iOS Core Telephony framework is provided to allow applications to interrogate the device for information about the current cell phone service provider and to receive notification of telephony related events.

3.5.7 EventKit Framework (`EventKit.framework`)

An API designed to provide applications with access to the calendar and alarms on the device.

3.5.8 Foundation Framework (`Foundation.framework`)

The Foundation framework is the standard Objective-C framework that will be familiar to those that have programmed in Objective-C on other platforms (most likely Mac OS X). Essentially, this consists of Objective-C wrappers around much of the C-based Core Foundation Framework.

3.5.9 Core Location Framework (`CoreLocation.framework`)

The Core Location framework allows you to obtain the current geographical location of the device (latitude and longitude) and compass readings from within your own applications. The method used by the device to provide coordinates will depend on the data available at the time the information is requested and the hardware support provided by the particular iPhone model on which the app is running (GPS and compass are only featured on recent models). This will either be based on GPS readings, Wi-Fi network data or cell tower triangulation (or some combination of the three).

3.5.10 Mobile Core Services Framework (`MobileCoreServices.framework`)

The iOS Mobile Core Services framework provides the foundation for Apple's Uniform Type Identifiers (UTI) mechanism, a system for specifying and identifying data types. A vast range of predefined identifiers have been defined by Apple including such diverse data types as text, RTF, HTML, JavaScript, PowerPoint .ppt files, PhotoShop images and MP3 files.

3.5.11 Store Kit Framework (StoreKit.framework)

The purpose of the Store Kit framework is to facilitate commerce transactions between your application and the Apple App Store. Prior to version 3.0 of iOS, it was only possible to charge a customer for an app at the point that they purchased it from the App Store. iOS 3.0 introduced the concept of the “in app purchase” whereby the user can be given the option make additional payments from within the application. This might, for example, involve implementing a subscription model for an application, purchasing additional functionality or even buying a faster car for you to drive in a racing game.

3.5.12 SQLite library

Allows for a lightweight, SQL based database to be created and manipulated from within your iPhone application.

3.5.13 System Configuration Framework (SystemConfiguration.framework)

The System Configuration framework allows applications to access the network configuration settings of the device to establish information about the “reachability” of the device (for example whether Wi-Fi or cell connectivity is active and whether and how traffic can be routed to a server).

3.5.14 Quick Look Framework (QuickLook.framework)

One of the many new additions included in iOS 4, the Quick Look framework provides a useful mechanism for displaying previews of the contents of files types loaded onto the device (typically via an internet or network connection) for which the application does not already provide support. File format types supported by this framework include iWork, Microsoft Office document, Rich Text Format, Adobe PDF, Image files, public.text files and comma separated (CSV).

3.6 The iOS Core OS Layer

The Core OS Layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware. The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

3.6.1 Accelerate Framework (Accelerate.framework)

Introduced with iOS 4, the Accelerate Framework provides a hardware optimized C-based API for performing complex and large number math, vector, digital signal processing (DSP) and image processing tasks and calculations.

3.6.2 External Accessory framework (`ExternalAccessory.framework`)

Provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

3.6.3 Security Framework (`Security.framework`)

The iOS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, keychains, encryption, digests and Hash-based Message Authentication Code (HMAC).

3.6.4 System (`LibSystem`)

As we have previously mentioned, the iOS is built upon a UNIX-like foundation. The System component of the Core OS Layer provides much the same functionality as any other UNIX like operating system. This layer includes the operating system kernel (based on the Mach kernel developed by Carnegie Mellon University) and device drivers. The kernel is the foundation on which the entire iOS is built and provides the low level interface to the underlying hardware. Amongst other things the kernel is responsible for memory allocation, process lifecycle management, input/output, inter-process communication, thread management, low level networking, file system access and thread management.

As an app developer your access to the System interfaces is restricted for security and stability reasons. Those interfaces that are available to you are contained in a C-based library called `LibSystem`. As with all other layers of the iOS stack, these interfaces should be used only when you are absolutely certain there is no way to achieve the same objective using a framework located in a higher iOS layer.

Chapter 4. Installing Xcode 4 and the iOS 4 SDK

iPhone apps are developed using the iOS SDK in conjunction with Apple's Xcode 4 development environment. The iOS SDK contains the development frameworks that were outlined in [iOS 4 Architecture and Frameworks](#). Xcode 4 is an integrated development environment (IDE) within which you will code, compile, test and debug your iOS iPhone applications. The Xcode 4 environment also includes a feature called Interface Builder that 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 4 and the iOS 4 SDK on Mac OS X.

4.1 Identifying if you have an Intel or PowerPC based Mac

Only Intel based Mac OS X systems can be used to develop applications for the iOS. If you have an older, PowerPC based Mac then you will need to purchase a new system before you can begin your iPhone app development project. If you are unsure of the processor type inside your Mac, you can find this information by opening the Finder and selecting the *About This Mac* option from the Apple menu. In the resulting dialog check the *Processor* line. The following figure 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 iPhone iOS app development.

In addition, the iOS 4.3 SDK with Xcode 4 environment requires that the version of Mac OS X running on the system be version 10.6.6 or later. If the “About This Mac” dialog does not indicate that Mac OS X 10.6.6 or later is running, click on the *Software Update...* button to download and install the appropriate operating system upgrades.

4.2 Installing Xcode 4 and the iOS 4 SDK

The best way to obtain the latest versions of Xcode 4 and the iOS SDK is to download them from the Apple iOS Dev Center web site at:

<http://developer.apple.com/devcenter/ios/index.action>

In order to download Xcode 4 with the iOS SDK, you will either need to be a member of the iOS or Mac Developer programs or purchase a copy from the Mac App Store at:

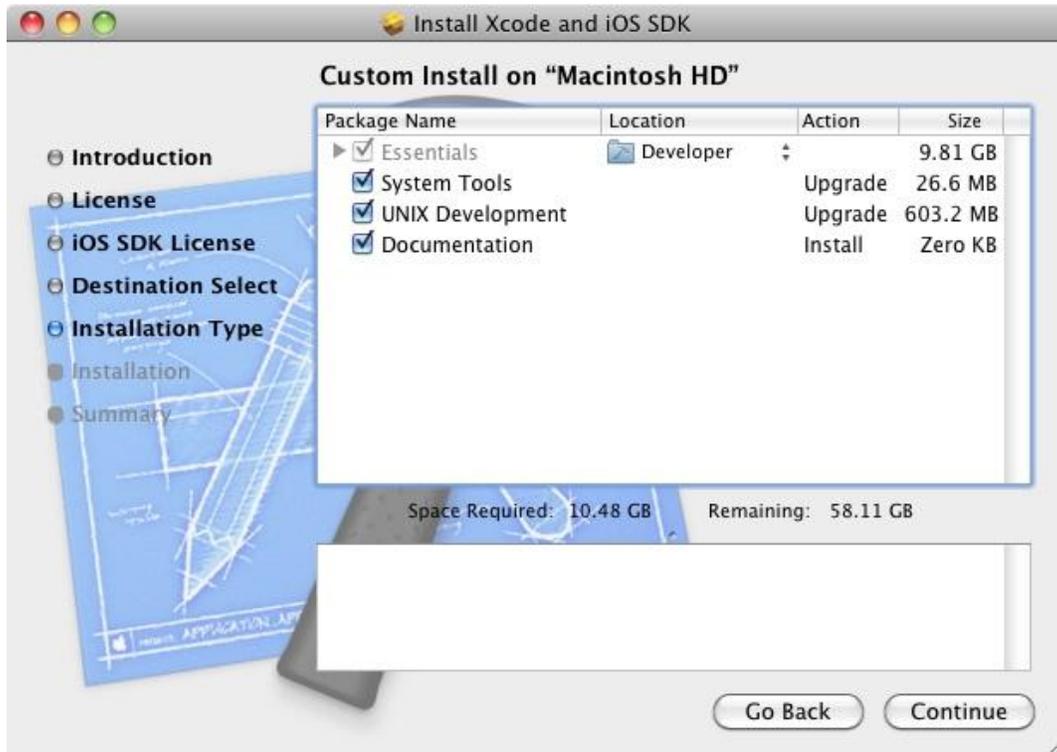
<http://itunes.apple.com/us/app/xcode/id422352214?mt=12&ls=1>

The download is over 3.5GB in size and may take a number of hours to complete depending on the speed of your internet connection. The package takes the form of a disk image (.dmg) file. Once the download has completed, a new window will open as follows displaying the contents of the .dmg file:



If this window does not open by default, it can be opened by clicking on the SDK disk drive icon on the desktop or by navigating to the Downloads directory of your home folder and double clicking on the corresponding dmg file.

Initiate the installation by double clicking on the package icon (the one that looks like an opening box) and follow the instructions until you reach the *Custom Install* screen:



The default selections on this screen are adequate for most requirements so unless you have specific needs there is no necessity to alter these selections. Continue to the next screen, review the information and click *Install* to begin the installation. Note that you may first be prompted to enter your password as a security precaution. The duration of the installation process will vary depending on the speed and current load on the computer, but typically completes in 25 - 45 minutes.

4.3 Starting Xcode 4

Having successfully installed the SDK and Xcode 4, the next step is to launch it so that we can write and then create a sample iPhone application. To start up Xcode 4, open the Finder, click on the *Macintosh HD* device in the left hand panel then double click on the *Developer* folder, followed by the *Applications* folder. Within this folder you should see an icon 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.

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:



Having installed the iPhone SDK and successfully launched Xcode 4 we can now look at [Creating a Simple iPhone iOS 4 App](#).

Chapter 5. Creating a Simple iPhone iOS 4 App

It is traditional in books covering programming topics to provide a very simple example early on. This practice, though still common, has been maligned by some authors of recent books. Those authors, however, are missing the point of the simple example. One key purpose of such an exercise is to provide a very simple way to verify that your development environment is correctly installed and fully operational before moving on to more complex tasks. A secondary objective is to give the reader a quick success very early in the learning curve to inspire an initial level of confidence. There is very little to be gained by plunging into complex examples that confuse the reader before we have taken time to explain the underlying concepts of the technology.

With this in mind, *iPhone iOS 4 Development Essentials – Xcode 4 Edition* will remain true to tradition and provide a very simple example to get us started. In doing so, we will also be honoring another time honored tradition by providing this example in the form of a simple “Hello World” program. The “Hello World” example was first used in a book called the C Programming Language written the creators of C, Brian Kernighan and Dennis Richie. Given that the origins of Objective-C can be traced back to the C programming language it is only fitting that we use this example for iOS 4 and the iPhone.

5.1 Starting Xcode 4

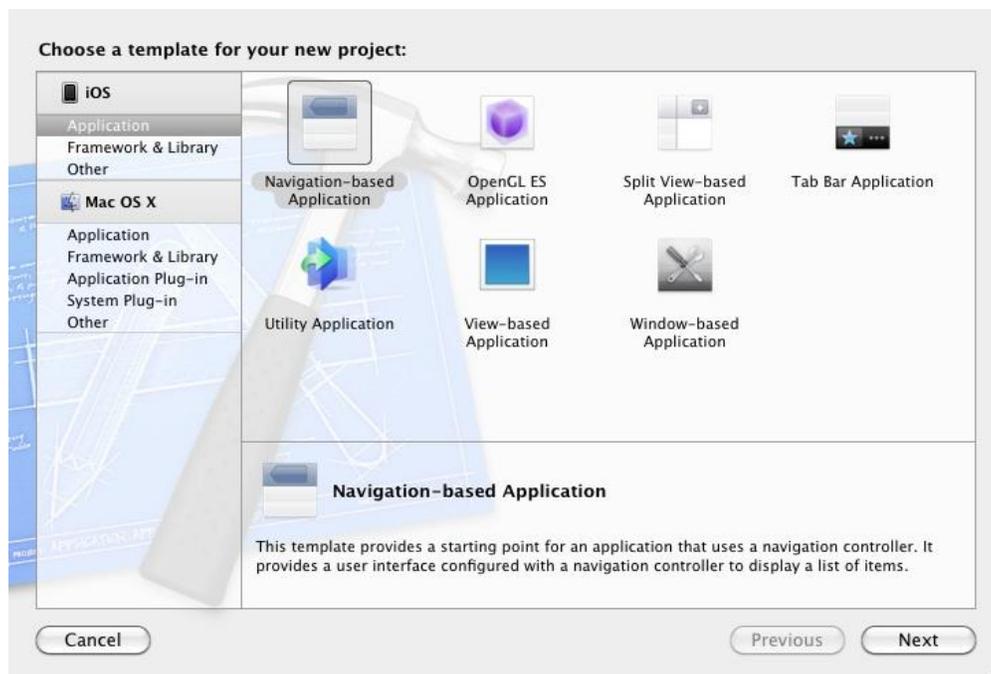
As with all iPhone iOS apps, the development of our example will take place within the Xcode 4 development environment. If you have not already installed this tool together with the latest iOS SDK refer first to [Installing Xcode 4 and the iOS 4 SDK](#). Assuming that the installation is complete, launch Xcode either by clicking on the icon on the dock (assuming you created one) or use the Finder to navigate to *Macintosh HD -> Developer -> Applications -> Xcode*.

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



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 4 project window together with the *New Project* panel where we are able to select a template matching the type of project we want to develop:

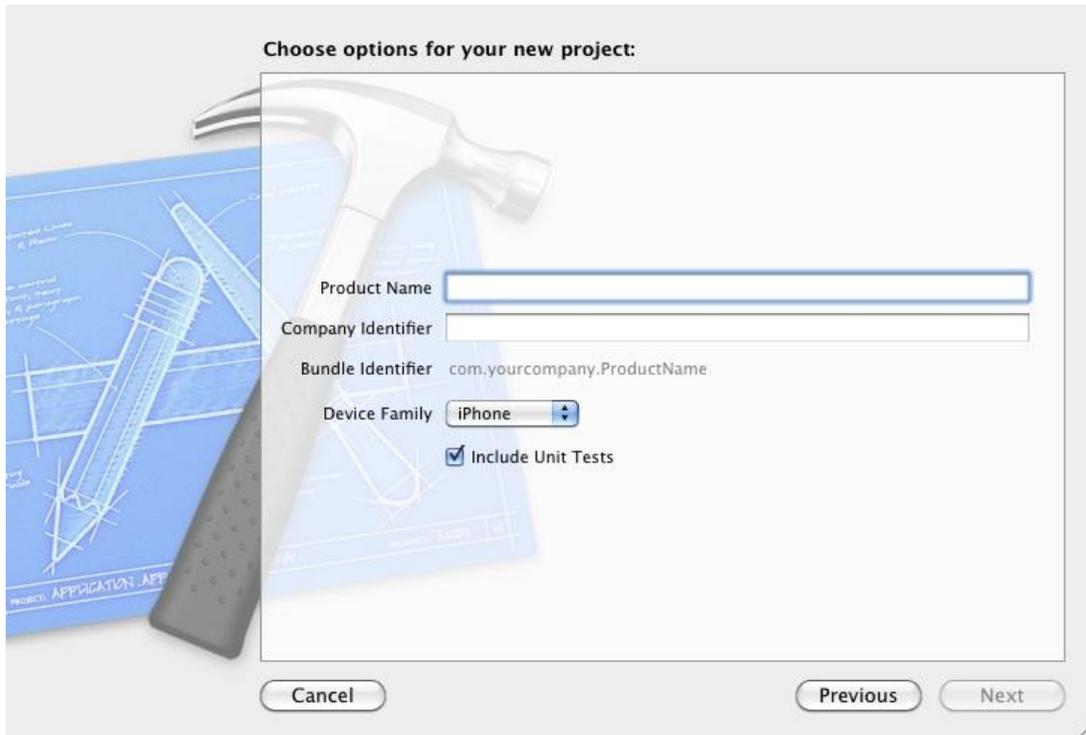


The panel located on the left hand side of the window allows for the selection of the target platform providing options to develop an application either for an iOS based device or Mac OS X.

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:

- **Navigation-based Application** – Though the name may suggest otherwise, this type of application has nothing to do with maps and street directions. In fact it allows you to create a list based application. Selecting an item from a list displays a new 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.
- **Open GL ES Application** – As discussed in [iOS 4 Architecture and SDK Frameworks](#), the OpenGL ES framework provides an API for developing advanced graphics drawing and animation capabilities. The Open GL ES Application template creates a basic application containing an Open GL ES view upon which to draw and manipulate graphics.
- **Split View Application** – An iPad only application template with a user interface containing two views in a master-detail configuration whereby a one view provides a list and second displays detailed information corresponding to the current list selection.
- **Tab Bar 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 which, 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 move between favorites, contacts, keypad and voicemail.
- **Utility Application** – Creates a template consisting of a two sided view. For an example of a utility application in action, load up the standard iPhone weather application. Pressing the blue info button flips the view to the configuration page. Selecting *Done* rotates the view back to the main screen.
- **View-based Application** – Creates a basic template for an application containing a single view.
- **Window-based Application** – The most basic of templates and creates only a window and a delegate. If none of the above templates match your requirements then this is the option to take.

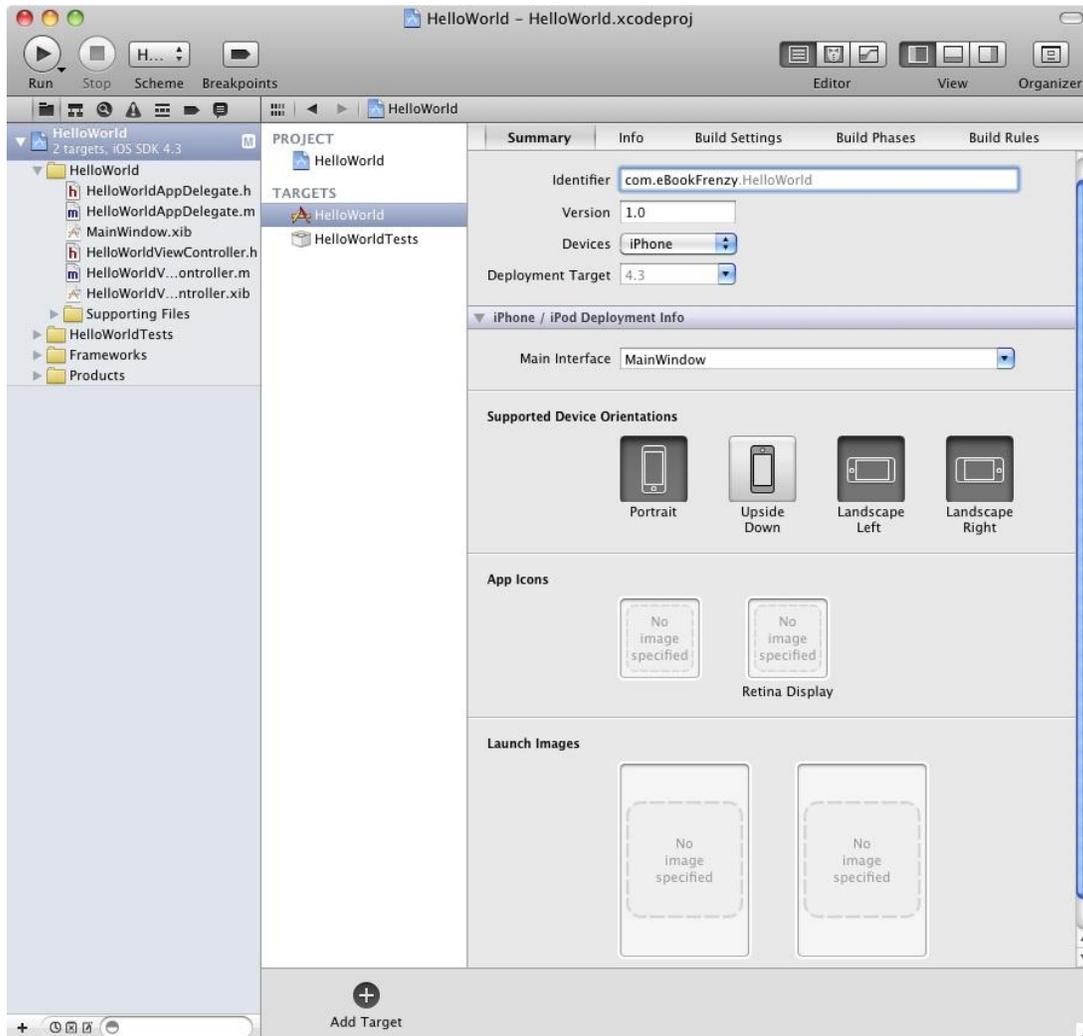
For the purposes of our simple example, we are going to use the View-based Application template so select this option from the new project window and click *Next* to configure some project options:



On this screen, enter a 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 device (covered in more detail in [Testing iOS 4 Apps on the iPhone – Developer Certificates and Provisioning Profiles](#))

Make sure that *iPhone* is currently selected from the *Device Family* menu before clicking the *Next* button to proceed. On the final screen, choose a location on the file system for the new project to be created can click on *Create*.

Once the new project has been created the main Xcode window will appear as follows:



Before we proceed 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 Objective-C source code files (with a `.m` extension) where we will enter the code to make our application work, others are header or interface files (`.h`) that are included by the source files and are where we will also need to put our own declarations and definitions. In addition, the `.xib` files are the save files used by the Interface Builder tool to hold the user interface designs we will create. Older versions of Interface Builder saved designs in files with a `.nib` extension so these files, even today, are called NIB files. Also present will be one or more files with a `.plist` file extension. These are *Property List* files that contain key/value pair information. For example, the `HelloWorld-info.plist` file contains resource settings relating to items such as the language, icon file, executable name and app identifier. The list of files is displayed in the *Project Navigator* located in 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, compilation errors and warnings, symbol navigator and a search panel.

The center panel of the window, by default, shows a summary of the settings for the application. 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 an icon (the small image the user selects on the device screen to launch the application) and splash screen image (displayed to the user while the application loads) for the application.

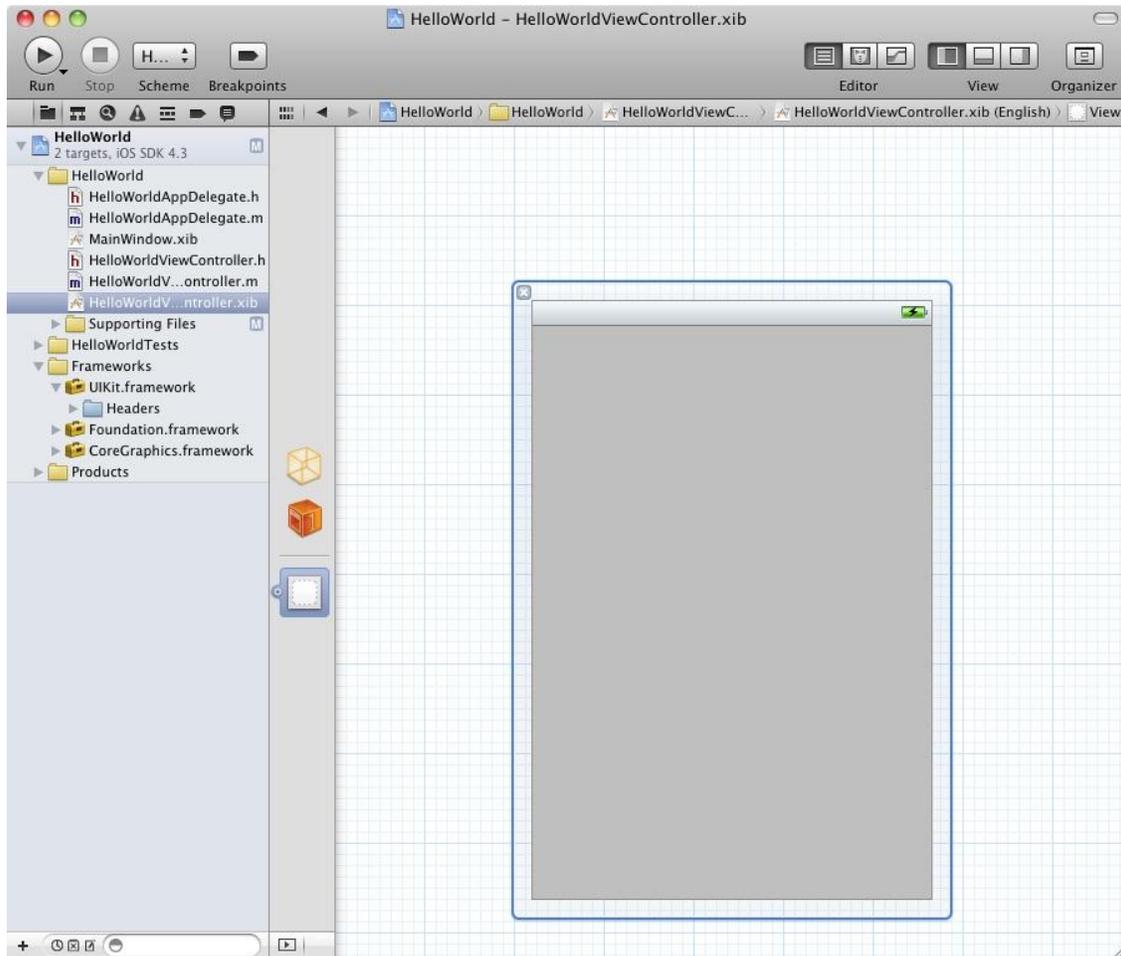
In addition to the Summary screen, tabs are provided to view and modify additional settings consisting of Info, Build Settings, Build Phases and Build Rules. As we progress through subsequent chapters of this book we will explore some of these other configuration options in greater detail. To return to the Summary 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 main panel, the contents of that file will appear in 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.

5.2 Creating the iOS App User Interface

Simply by the very nature of the environment in which they run, iPhone 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 that 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, some of which have a .xib filename extension. These are Interface Builder save files (remember that they are called NIB files, not XIB files). The file we are interested in for our HelloWorld project is called *HelloWorldViewController.xib*. 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 the following figure:



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 that was added to our design by Xcode when we selected the View-based 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. This is achieved by selecting the right hand button in the *View* section of the Xcode toolbar:



The right hand panel, once displayed, will appear as illustrated in the following figure:



Along the top edge of the panel is a row of buttons that 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 defaults 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 in the

appropriate toolbar button (the three dimensional cube). 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. In addition, the objects are categorized into groups that may be selected using the menu beneath the toolbar. The layout buttons may also be used to switch from a single column of objects with descriptions to multiple columns without descriptions.

5.3 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. Begin by making sure the View is selected and that the Attribute Inspector (*View -> Utilities -> Attribute Inspector*) is displayed in the right hand panel. Click on the gray 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 gray to the new color selection.

5.4 Adding Objects to the User Interface

The next step is to add a Label object to our view. To achieve this, select *Controls* from the library panel menu, click on the *Label* object and drag it to the center of the view. Once it is in position release the mouse button to drop it at that location:



Using the blue markers surrounding the label border, stretch first the left and then right side of the label out to the edge of the view until the vertical blue dotted lines marking the recommended border of the view appear. With the Label still selected, click on the centered alignment button in the *Layout* attribute section of the Attribute Inspector (*View -> Utilities -> Attribute Inspector*) to center the text in the middle of the screen. Click on the current font attribute setting to choose a larger font setting, for example a Georgia bold typeface with a size of 24.

Finally, double click on the text in the label that currently reads “Label” and type in “Hello World”. At this point, your View window will hopefully appear as outlined in the following figure (allowing, of course, for differences in your color and font choices):



Having created our simple user interface design we now need to save it. To achieve this, select *File -> Save* or use the Command+S keyboard shortcut.

5.5 Building and Running an iOS App in Xcode 4

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 iOS 4 Apps on the iPhone – Developer Certificates and Provisioning Profiles](#). For the purposes of this chapter, however, it is sufficient to run the app in the simulator.

Within the main Xcode 4 project window make sure that the menu located in the top left hand corner of the window (to the right of the Stop button) has the *iPhone Simulator* option selected and then click on the *Play* toolbar button to compile the code and run the app in the simulator. The small iTunes style window in the center of the Xcode toolbar 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:



5.6 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 4 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.

Chapter 6. Testing iOS 4 Apps on the iPhone – Developer Certificates and Provisioning Profiles

In the chapter entitled [Creating a Simple iPhone iOS 4 App](#) we were able to see an app that we had created running in the iOS Simulator environment bundled with the iOS 4 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). If we really want to thoroughly test an iOS application in the real world, therefore, then we need to install the app onto a physical iPhone device.

In order to achieve this there are a number of steps that must be performed. These include signing up to the iOS Developer Program, generating and installing a developer certificate, creating an App ID and provisioning profile for your application, and registering the devices onto which you wish to directly install your apps for testing purposes. In the remainder of this chapter we will cover these steps in detail.

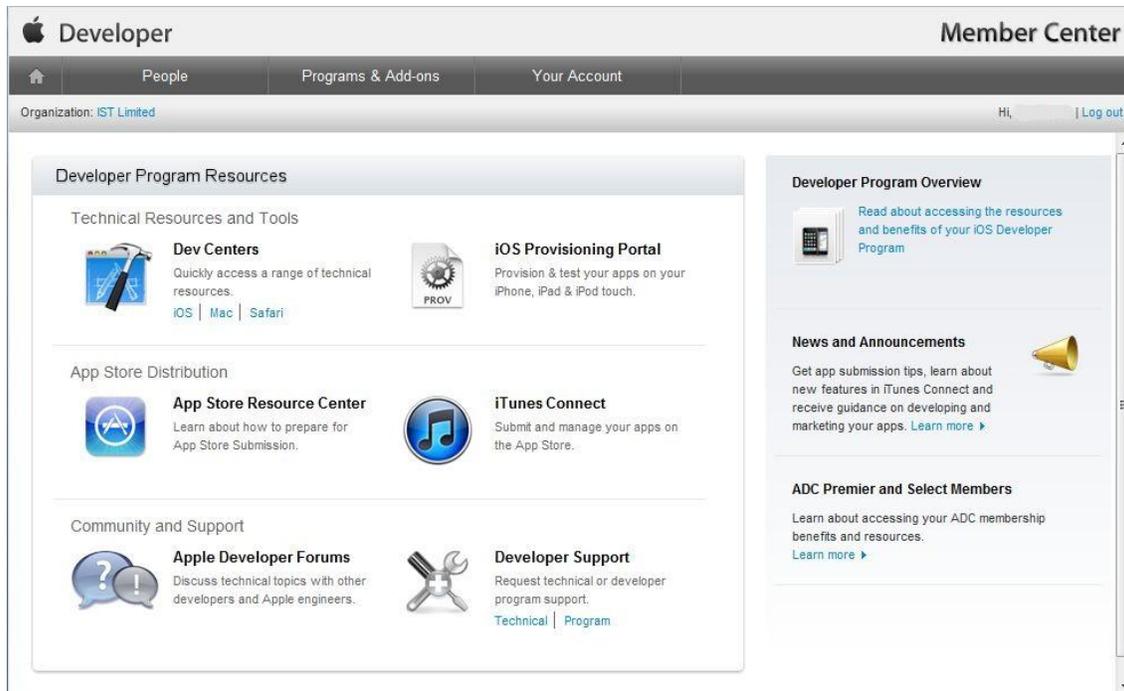
6.1 Joining the iOS Developer Program

Being a member of the iOS Developer Program should not be confused with being a registered Apple developer. Being a registered Apple developer only gives you the ability to download the iOS SDK and access to additional developer related information. Membership of the iOS Developer Program, however, allows you to set up certificates and provisioning profiles to test apps on physical devices and, ultimately, submit completed apps for possible acceptance into the Apple App Store.

Enrollment into this program currently costs \$99 per year. It is also possible that your employer already has membership, in which case contact the program administrator in your company and ask them to send you an invitation to join. Once they have done this 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/>

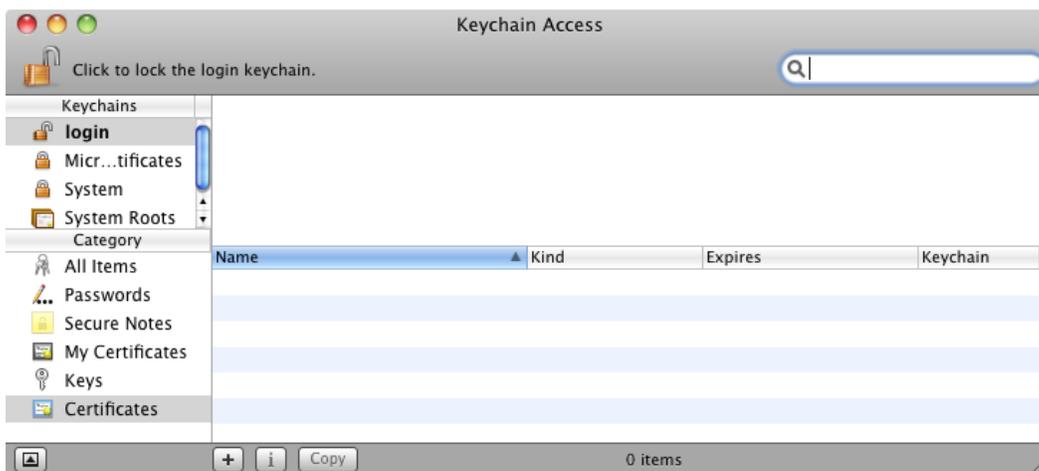
Having completed the enrollment process, navigate to <http://developer.apple.com> and click on the *Member Center* link located near the top right hand corner of the screen. On the resulting page enter the Apple ID and password associated with your iOS Developer Program membership to access the member center home page as illustrated in the following figure:



Having gained access to the iOS Developer Program the next step is to create and install a certificate on the Mac OS X system on which you are developing your iPhone apps.

6.2 Creating an iOS Development Certificate Signing Request

Any apps that are to be installed on a physical iPhone device must first be signed using an iOS Development Certificate. In order to generate a certificate the first step is to generate a Certificate Signing Request (CSR). Begin this process by opening the Keychain Access tool on your Mac system. This tool can be found in the *Applications* -> *Utilities* folder. Once launched, the Keychain Access main window will appear as follows:



Within the Keychain Access utility, perform the following steps:

1. Select the *Keychain Access -> Preferences* menu and select *Certificates* in the resulting dialog:



2. Within the Preferences dialog make sure that the online Certificate Status Protocol (OCPS) and Certificate Revocation List (CRL) settings are both set to *Off*, then close the dialog.
3. Select the *Keychain Access -> Certificate Assistant -> Request a Certificate from a Certificate Authority...* menu option and enter your email and name exactly as registered with the iOS Developer Program. Leave the *CA Email Address* field blank and select the *Saved to Disk* and *Let me specify key pair information* options:

