

# iPhone SDK 3 Programming

Advanced Mobile Development for Apple  
iPhone and iPod touch

**Maher Ali, PhD**

*Bell Labs, Alcatel-Lucent*



A John Wiley and Sons, Ltd, Publication



# iPhone SDK 3 Programming



# iPhone SDK 3 Programming

Advanced Mobile Development for Apple  
iPhone and iPod touch

**Maher Ali, PhD**

*Bell Labs, Alcatel-Lucent*



A John Wiley and Sons, Ltd, Publication

This edition first published 2009  
© 2009, John Wiley & Sons, Ltd

*Registered office*

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom.

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at [www.wiley.com](http://www.wiley.com).

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

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

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Trademarks: Wiley and the Wiley Publishing logo are trademarks or registered trademarks of John Wiley and Sons, Inc. and/or its affiliates in the United States and/or other countries, and may not be used without written permission. iPhone and iPod are trademarks of Apple Computer, Inc. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in the book. This book is not endorsed by Apple Computer, Inc.

ISBN 978-0-470-68398-9

Typeset by Sunrise Setting Ltd, Torquay, UK.  
Printed in the United States of America.

# CONTENTS

<b>Preface</b>	<b>xv</b>
<b>1 Getting Started</b>	<b>1</b>
1.1 SDK and IDE Basics	1
1.1.1 Obtaining and installing the SDK	1
1.1.2 Creating a project	2
1.1.3 Familiarizing yourself with the IDE	3
1.1.4 Looking closely at the generated code	5
1.2 Creating Interfaces	6
1.2.1 Interface Builder	7
1.3 Using the Debugger	14
1.4 Getting More Information	15
1.5 Summary	16
Problems	17
<b>2 Objective-C and Cocoa</b>	<b>19</b>
2.1 Classes	20
2.1.1 Class declaration	20
2.1.2 How do I use other declarations?	21
2.1.3 Class definition	22
2.1.4 Method invocation and definition	22
2.1.5 Important types	23
2.1.6 Important Cocoa classes	24
2.2 Memory Management	24
2.2.1 Creating and deallocating objects	24
2.2.2 Preventing memory leaks	25
2.3 Protocols	27
2.3.1 Protocol conformance	28
2.4 Properties	29
2.4.1 Property declaration	29
2.4.2 Circular references	34
2.5 Categories	36
2.6 Posing	38

2.7	Exceptions and Errors	38
2.7.1	Exceptions	38
2.7.2	Errors	43
2.8	Key-value coding (KVC)	45
2.8.1	An example illustrating KVC	46
2.9	Multithreading	51
2.10	Notifications	55
2.11	The Objective-C Runtime	56
2.11.1	Required header files	57
2.11.2	The NSObject class	58
2.11.3	Objective-C methods	59
2.11.4	Examples	62
2.12	Summary	79
	Problems	79
<b>3</b>	<b>Collections</b>	<b>83</b>
3.1	Arrays	83
3.1.1	Immutable copy	86
3.1.2	Mutable copy	88
3.1.3	Deep copy	89
3.1.4	Sorting an array	93
3.2	Sets	96
3.2.1	Immutable sets	97
3.2.2	Mutable sets	99
3.2.3	Additional important methods	100
3.3	Dictionaries	101
3.3.1	Additional important methods	103
3.4	Summary	103
	Problems	104
<b>4</b>	<b>Anatomy of an iPhone Application</b>	<b>105</b>
4.1	Hello World Application	105
4.1.1	Create a main.m file	105
4.1.2	Create the application delegate class	106
4.1.3	Create the user interface subclasses	107
4.2	Building the Hello World Application	108
4.3	Summary	113
	Problems	113
<b>5</b>	<b>The View</b>	<b>115</b>
5.1	View Geometry	115
5.1.1	Useful geometric type definitions	115
5.1.2	The UIScreen class	117
5.1.3	The frame and center properties	118



---

5.1.4	The bounds property	119
5.2	The View Hierarchy	121
5.3	The Multitouch Interface	121
5.3.1	The UITouch class	122
5.3.2	The UIEvent class	123
5.3.3	The UIResponder class	123
5.3.4	Handling a swipe	128
5.3.5	More advanced gesture recognition	132
5.4	Animation	137
5.4.1	Using the UIView class animation support	137
5.4.2	Sliding view	141
5.4.3	Flip animation	142
5.4.4	Transition animation	142
5.5	Drawing	145
5.6	Summary	147
	Problems	147
<b>6</b>	<b>Controls</b>	<b>149</b>
6.1	The Foundation of all Controls	149
6.1.1	UIControl attributes	149
6.1.2	Target-action mechanism	150
6.2	The Text Field	153
6.2.1	Interacting with the keyboard	155
6.2.2	The delegate	158
6.2.3	Creating and working with a UITextField	159
6.3	Sliders	160
6.4	Switches	161
6.5	Buttons	163
6.6	Segmented Controls	164
6.7	Page Controls	167
6.8	Date Pickers	168
6.9	Summary	170
	Problems	170
<b>7</b>	<b>View Controllers</b>	<b>171</b>
7.1	The Simplest View Controller	171
7.1.1	The view controller	171
7.1.2	The view	173
7.1.3	The application delegate	174
7.1.4	Summary: creating a simple MVC application	175
7.2	Radio Interfaces	177
7.2.1	A detailed example	177
7.2.2	Some comments on tab bar controllers	182
7.3	Navigation Controllers	186

7.3.1	A detailed example	187
7.3.2	Customization	193
7.4	Modal View Controllers	197
7.4.1	A detailed example	197
7.5	Summary	203
	Problems	203
<b>8</b>	<b>Special-Purpose Views</b>	<b>205</b>
8.1	Picker View	205
8.1.1	The delegate	206
8.1.2	An example	207
8.2	Progress Views	211
8.2.1	An example	213
8.3	Scroll View	215
8.4	Text View	217
8.4.1	The delegate	218
8.4.2	An example	218
8.5	Alert View	221
8.6	Action Sheet	224
8.7	Web View	225
8.7.1	A simple web view application	226
8.7.2	Viewing local files	230
8.7.3	Evaluating JavaScript	235
8.7.4	The web view delegate	242
8.8	Summary	247
	Problems	247
<b>9</b>	<b>Table View</b>	<b>249</b>
9.1	Overview	249
9.2	The Simplest Table View Application	250
9.3	A Table View with both Images and Text	255
9.4	A Table View with Section Headers and Footers	257
9.5	A Table View with the Ability to Delete Rows	258
9.6	A Table View with the Ability to Insert Rows	265
9.7	Reordering Table Rows	270
9.8	Presenting Hierarchical Information	275
9.8.1	Detailed example	278
9.9	Grouped Table Views	285
9.10	Indexed Table Views	288
9.11	Dynamic Table Views	294
9.12	Whitening Text in Custom Cells	297
9.13	Summary	302
	Problems	303

---

<b>10 File Management</b>	<b>305</b>
10.1 The Home Directory	305
10.2 Enumerating a Directory	306
10.3 Creating and Deleting a Directory	308
10.4 Creating Files	309
10.5 Retrieving and Changing Attributes	312
10.5.1 Retrieving attributes	314
10.5.2 Changing attributes	315
10.6 Working with Resources and Low-level File Access	317
10.7 Summary	320
Problems	321
<b>11 Working with Databases</b>	<b>323</b>
11.1 Basic Database Operations	323
11.1.1 Opening, creating, and closing databases	325
11.1.2 Table operations	325
11.2 Processing Row Results	327
11.3 Prepared Statements	330
11.3.1 Preparation	330
11.3.2 Execution	331
11.3.3 Finalization	331
11.3.4 Putting it together	331
11.4 User-defined Functions	333
11.5 Storing BLOBs	337
11.6 Retrieving BLOBs	341
11.7 Summary	343
Problems	343
<b>12 XML Processing</b>	<b>345</b>
12.1 XML and RSS	345
12.1.1 XML	345
12.1.2 RSS	347
12.1.3 Configuring the XCode project	350
12.2 Document Object Model (DOM)	351
12.3 Simple API for XML (SAX)	358
12.4 An RSS Reader Application	367
12.5 Putting It Together	369
12.6 Summary	371
Problems	371
<b>13 Location Awareness</b>	<b>373</b>
13.1 The Core Location Framework	373
13.1.1 The CLLocation class	375
13.2 A Simple Location-aware Application	377

13.3	Google Maps API	380
13.3.1	A geocoding application	380
13.4	A Tracking Application with Maps	386
13.5	Working with ZIP Codes	392
13.6	Working with the Map Kit API	394
13.6.1	The MKMapView class	395
13.6.2	The MKCoordinateRegion structure	395
13.6.3	The MKAnnotation protocol	396
13.6.4	The MKAnnotationView class	397
13.6.5	The MKUserLocation class	399
13.6.6	The MKPinAnnotationView class	401
13.7	Summary	401
	Problems	402
<b>14</b>	<b>Working with Devices</b>	<b>403</b>
14.1	Working with the Accelerometer	403
14.1.1	Basic accelerometer values	403
14.1.2	Example	405
14.2	Working with Audio	408
14.2.1	Playing short audio files	408
14.2.2	Recording audio files	410
14.2.3	Playing audio files	412
14.2.4	Using the media picker controller	412
14.2.5	Searching the iPod library	415
14.3	Playing Video	418
14.3.1	Using the MPMoviePlayerController class	418
14.4	Accessing Device Information	419
14.5	Taking and Selecting Pictures	420
14.5.1	Overall approach	420
14.5.2	Detailed example	421
14.6	Monitoring Device Battery	424
14.6.1	Battery level	424
14.6.2	Battery state	424
14.6.3	Battery state and level notifications	424
14.6.4	Putting it together	425
14.7	Accessing the Proximity Sensor	426
14.7.1	Enabling proximity monitoring	427
14.7.2	Subscribing to proximity change notification	427
14.7.3	Retrieving the proximity state	427
14.8	Summary	428
	Problems	428
<b>15</b>	<b>Internationalization</b>	<b>429</b>
15.1	String Localization	430

---

15.2	Date Formatting	435
15.2.1	Custom formats	437
15.3	Number Formatting	438
15.4	Sorted List of Countries	441
15.5	Summary	441
	Problems	441
<b>16</b>	<b>Custom UI Components</b>	<b>443</b>
16.1	Text Field Alert View	443
16.2	Table Alert View	447
16.3	Progress Alert View	452
16.4	Summary	456
	Problems	456
<b>17</b>	<b>Advanced Networking</b>	<b>459</b>
17.1	Determining Network Connectivity	459
17.1.1	Determining network connectivity via EDGE or GPRS	460
17.1.2	Determining network connectivity in general	461
17.1.3	Determining network connectivity via WiFi	461
17.2	Uploading Multimedia Content	462
17.3	Computing MD5 Hash Value	465
17.4	Multithreaded Downloads	467
17.4.1	The application	467
17.5	Push Notification	474
17.5.1	Configuring push notification on the server	474
17.5.2	Configuring the client	481
17.5.3	Coding the client	484
17.5.4	Coding the server	487
17.6	Sending Email	487
17.6.1	Using the mail composition view controller	488
17.7	Summary	490
	Problems	491
<b>18</b>	<b>Working with the Address Book Database</b>	<b>493</b>
18.1	Introduction	493
18.2	Property Types	494
18.3	Accessing Single-Value Properties	494
18.3.1	Retrieving single-value properties	495
18.3.2	Setting single-value properties	496
18.4	Accessing Multi-Value Properties	496
18.4.1	Retrieving multi-value properties	496
18.4.2	Setting multi-value properties	499
18.5	Person and Group Records	500
18.6	Address Book	501

18.7 Multithreading and Identifiers	503
18.8 Person Photo Retriever Application	503
18.9 Using the ABUnknownPersonViewController Class	505
18.10 Using the ABPeoplePickerNavigationController Class	507
18.11 Using the ABPersonViewController Class	509
18.12 Using the ABNewPersonViewController Class	510
18.13 Summary	512
Problems	513
<b>19 Core Data</b>	<b>515</b>
19.1 Core Data Application Components	515
19.2 Key Players	516
19.2.1 Entity	516
19.2.2 Managed object model	516
19.2.3 Persistent store coordinator	517
19.2.4 Managed object context	517
19.2.5 Managed object	517
19.2.6 Summary	518
19.3 Using the Modeling Tool	521
19.4 Create, Read, Update and Delete (CRUD)	527
19.4.1 Create	527
19.4.2 Delete	527
19.4.3 Read and update	527
19.5 Working with Relationships	530
19.6 A Search Application	531
19.6.1 The UISearchDisplayController class	531
19.6.2 Main pieces	533
19.7 Summary	538
Problems	538
<b>20 Undo Management</b>	<b>539</b>
20.1 Understanding Undo Management	539
20.1.1 Basic idea	539
20.1.2 Creating an undo manager	540
20.1.3 Registering an undo operation	540
20.1.4 Hooking into the undo management mechanism	541
20.1.5 Enabling shake to edit behavior	542
20.2 Detailed Example	543
20.2.1 The view controller class	543
20.2.2 First responder status	543
20.2.3 Editing mode and the NSUndoManager instance	544
20.2.4 Registering undo actions	544
20.3 Wrapping Up	546

---

20.4 Summary	546
Problems	546
<b>21 Copy and Paste</b>	<b>547</b>
21.1 Pasteboards	547
21.1.1 System pasteboards	547
21.1.2 Creating pasteboards	547
21.1.3 Properties of a pasteboard	548
21.2 Pasteboard Items	548
21.2.1 Pasteboard items	549
21.2.2 Manipulating pasteboard items	549
21.3 The Editing Menu	551
21.3.1 The standard editing actions	551
21.3.2 The UINavigationController class	551
21.3.3 The role of the view controller	552
21.4 Putting it Together	553
21.4.1 The image view	553
21.4.2 The view controller	554
21.5 Summary	558
Problems	559
 <b>Appendices</b>	 <b>561</b>
<b>A Saving and Restoring App State</b>	<b>563</b>
<b>B Invoking External Applications</b>	<b>567</b>
<b>C App Store Distribution</b>	<b>569</b>
<b>D Using XCode</b>	<b>571</b>
D.1 XCode Shortcuts	571
D.2 Creating Custom Templates	571
D.2.1 Changing template macro definitions	573
D.3 Build-Based Configurations	574
D.4 Using Frameworks	577
<b>E Unit Testing</b>	<b>581</b>
E.1 Adding a Unit Test Target	581
E.2 Adapting to Foundation	582
E.3 The Model	584
E.4 Writing Unit Tests for the Employee Class	586
E.4.1 The setUp and tearDown methods	587
E.4.2 Testing for equality	588
E.4.3 Testing for nullity	588

E.5	Adding a Build Dependency	589
E.6	Running the Tests	589
<b>F</b>	<b>Working with Interface Builder</b>	<b>591</b>
F.1	National Debt Clock Application	591
F.1.1	Creating the project	591
F.1.2	Creating the view controller class	591
F.1.3	The application delegate class	594
F.1.4	Building the UI	595
F.2	Toolbar Application	609
F.2.1	Writing code	609
F.2.2	Building the UI	611
F.2.3	Putting it together	617
	<b>References and Bibliography</b>	<b>619</b>
	<b>Index</b>	<b>621</b>



# PREFACE

Welcome to *iPhone SDK 3 Programming*, an introductory text to the development of mobile applications for the iPhone and the iPod touch. This text covers a wide variety of essential and advanced topics, including:

- The Objective-C programming language and runtime
- Collections
- Cocoa Touch
- Interface Builder
- Building advanced mobile user interfaces
- Core Animation and Quartz 2D
- Model-view-controller (MVC) designs
- Table views
- Core Data
- File management
- Parsing XML documents using SAX and DOM
- Working with the Map Kit API
- Push notification
- Working with the address book
- Consuming RESTful web services
- Building advanced location-based applications
- Developing database applications using the SQLite engine
- Cut, copy, and paste
- Undo management
- Unit testing
- Advanced networking
- Internationalization
- Building multimedia applications

## Is this book for you?

This book is aimed primarily at application developers with a basic understanding of the C language and object orientation concepts such as encapsulation and polymorphism. You don't need to be an expert C coder to follow this book. All you need is a basic understanding of structures, pointers, and functions. That said, you will find coverage of general topics such as databases and XML processing. These topics are covered assuming basic knowledge.

## What else do you need?

To master iPhone SDK programming, you will need the following:

- Intel-based Mac running Mac OS X Leopard.
- iPhone SDK 3. Download from: <http://developer.apple.com/iphone>.
- Optional: membership of the iPhone Developer Program so that you can use the device for development. (You will need to pay a fee for membership.)
- Source code. The source code of the applications illustrated in this book is available online at: <http://code.google.com/p/iphone3/>.

## Conventions used in this book

`Constant width` typeface is used for:

- Code examples and fragments.
- Anything that might appear in a program, including operators, method names, function names, class names, and literals.

**Constant-width bold** is used for:

- C, Objective-C, SQL, HTML, and XML keywords whether in text or in program listing.

*Italic* is used for:

- New terms and concepts when they are introduced.
- Specifying emphasis in text.

## Organization

**Chapter 1** This chapter serves as a quick introduction to the tools bundled with the SDK. It also shows you the basic development phases that include coding, UI design, and debugging.

**Chapter 2** This chapter presents the main features of the Objective-C language under the Cocoa environment. We introduce the main concepts behind classes in Objective-C. You will learn how to declare a new class, define it, and use it from within other classes. You will also be

exposed to important Cocoa classes and data types. You will learn about memory management in the iPhone OS. You will learn how to create new objects as well as how to deallocate them. You will also learn about your responsibility when obtaining objects from Cocoa frameworks or other frameworks. We also introduce the topic of Objective-C protocols. You will learn how to adopt protocols and how to declare new ones as well. This chapter also covers language features such as properties, categories, and posing. Exceptions and error handling techniques are both covered in this chapter, and you will be exposed to the concept of key-value coding (KVC). You will also learn how to utilize multithreading, use notifications, and will be exposed to the Objective-C runtime system.

**Chapter 3** This chapter addresses the topic of collections in Cocoa. It discusses arrays, sets, and dictionaries. You will learn about immutable and mutable collections, the different approaches used for copying collections, and several sorting techniques.

**Chapter 4** In this chapter, we discuss the basic steps needed to build a simple iPhone application. First, we demonstrate the basic structure of a simple iPhone application and then we show the steps needed to develop the application using XCode.

**Chapter 5** This chapter explains the main concepts behind views. You will learn about view geometry, view hierarchy, the multitouch interface, animation, and basic Quartz 2D drawing.

**Chapter 6** In this chapter, you will learn about the base class for all controls, `UIControl`, and the important *target-action* mechanism. This chapter also presents several important graphical controls that can be used in building attractive iPhone applications.

**Chapter 7** In this chapter, you will learn about the available view controllers that are provided to you in the iPhone SDK. Although you can build iPhone applications without the use of these view controllers, you shouldn't. As you will see in this chapter, view controllers greatly simplify your application. This chapter provides a gentle introduction to view controllers. After that, detailed treatment of tab bar controllers, navigation controllers, and modal view controllers is provided.

**Chapter 8** In this chapter, we present several important subclasses of the `UIView` class. We discuss picker views and show how they can be used for item selection. We investigate progress views and also talk about activity indicator views. After that, we show how to use scroll views in order to display large views. Next, we present text views used in displaying multiline text. After that, we show how to use alert views for the display of alert messages to the user. Similar to alert views are action sheets which are also discussed. We also deal with several aspects of web views.

**Chapter 9** This chapter will take you through a step-by-step journey to the world of table views. We start by presenting an overview of the main concepts behind table views. After that, we present a simple table view application and discuss the mandatory methods you need to implement in order to populate and respond to users' interactions with the table view. We show how easy it is to add images to table rows. We introduce the concept of sections and provide a table view application that has sections, with section headers and footers. We introduce the

concept of editing a table view. An application that allows the user to delete rows is presented and the main ideas are clarified. We address the insertion of new rows in a table view. An application is discussed that presents a data entry view to the user and adds that new data to the table's rows. We continue our discussion of the editing mode and present an application for reordering table entries. The main concepts of reordering rows are presented. We discuss the mechanism for presenting hierarchical information to the user. An application that uses table views to present three levels of hierarchy is discussed. We deal with grouped table views through an example. After that, we present the main concepts behind indexed table views. Next, we present a dynamic table view controller class which can be used to show cells with varying heights. Finally, we address the issue of turning the text color to white when a custom cell is selected.

**Chapter 10** This chapter covers the topic of file management. Here, you will learn how to use both high- and low-level techniques for storing/retrieving data to/from files. First, we talk about the `Home` directory of the application. Next, we show how to enumerate the contents of a given directory using the high-level methods of `NSFileManager`. You will learn more about the structure of the `Home` directory and where you can store files. After that, you will learn how to create and delete directories. Next, we cover the creation of files. We also cover the topic of file and directory attributes. You will learn how to retrieve and set specific file/directory attributes in this chapter. We also demonstrate the use of application bundles and low-level file access.

**Chapter 11** In this chapter, we will cover the basics of the SQLite database engine that is available to you, using the iPhone SDK. SQLite is an embedded database in the sense that there is no server running, and the database engine is linked to your application. First, we describe basic SQL statements and their implementation using SQLite function calls. Second, we discuss handling of result sets generated by SQL statements. Third, we address the topic of prepared statements. Fourth, we talk about extensions to the SQLite API through the use of user-defined functions. Finally, we present a detailed example for storing and retrieving BLOBs to/from the database.

**Chapter 12** In this chapter, you will learn how to effectively use XML in your iPhone application. The chapter follows the same theme used in other chapters and exposes the main concepts through a working iPhone application: an RSS feed reader. First, we explain the main concepts behind XML and RSS. Next, we present a detailed discussion of DOM and SAX parsing. After that, we present a table-based RSS reader application. Finally, we provide a summary of the main steps you need to take in order to effectively harness the power of XML from within your native iPhone application.

**Chapter 13** In this chapter, we will address the topic of location awareness. First, we will talk about the Core Location framework and how to use it to build location-aware applications. After that, we will discuss a simple location-aware application. Next, we cover the topic of geocoding. You will learn how to translate postal addresses into geographical locations. You will also learn how to sample movement of the device and display that information on maps. Next, we discuss how to relate ZIP codes to geographical information. Finally, we show you how to utilize the Map Kit API to add an interactive map to your view hierarchy.

- Chapter 14** In this chapter, we demonstrate the use of the several devices available on the iPhone. We discuss the use of the accelerometer, show how to play small sound files, and show how to play video files. After that, we discuss how to obtain iPhone/iPod touch device information. Using the built-in camera and the photo library are also discussed in this chapter. After that, we show you how to obtain state information regarding the battery of the device. Finally, we discuss the proximity sensor.
- Chapter 15** In this chapter, we start by looking at a step-by-step procedure for localizing strings for a set of supported languages. Next, we look at date formatting. After that, we cover formatting currencies and numbers. Finally, we discuss how to generate a sorted list of countries of the world.
- Chapter 16** In this chapter, we show how to marry various UI components and build custom reusable ones. First, we show how to build an alert view with a text field in it. Next, we present a table view inside an alert view. Finally, we show how to build a progress alert view.
- Chapter 17** This chapter addresses several advanced networking topics. We start by looking at how we can determine network connectivity of the device. After that, we tackle the issue of uploading multimedia content (e.g., photos) to remote servers. Next, we present a category on `NSString` that allows you to easily compute the MD5 digest of a string. This is important as some services, such as Flickr, require posting parameters with the appropriate signature. After that, we show you how to present a responsive table view whose data rows are fed from the Internet without sacrificing the user experience. Next, we address the topic of push notification. Finally, we discuss sending email from within your iPhone application.
- Chapter 18** In this chapter, we discuss the foundation of the address book API and several UI elements that can be used to modify the contacts database. First, we provide a brief introduction to the subject. Next, we discuss property types. After that, we show how to access single- and multi-value properties. Next, we go into the details of the person record and the address book. Issues related to multithreading and identifiers are then addressed. After covering the foundation of the address book API, we provide several sample applications.
- Chapter 19** In this chapter, you learn how to use the Core Data framework in your application. First, you learn about the main components in the Core Data application. Next, we talk about the major classes in the Core Data framework. After that, you learn how to use the graphical modeling tool to build a data model. Next, we address the basic operations in persistence storage using Core Data. After that, we show how to use relationships in the Core Data model. Finally, we present a search application that utilizes Core Data for storage.
- Chapter 20** In this chapter, you learn about undo management support in the iPhone OS. First, we discuss the basic steps needed to utilize undo management. After that, we present a detailed example that shows how to use undo management. Finally, we summarize the main rules in using the undo capabilities in an application.
- Chapter 21** This chapter examines the copy and paste capabilities of the iPhone OS and the supporting APIs. We start by discussing pasteboards. Next, you learn about pasteboard items

and the various methods available to you to manipulate them. After that, we address the subject of the editing menu which users use to issue editing commands. Finally, we put all the ideas behind copy and paste together and present a simple image editing application.

**Appendix A** In this appendix, you will learn how to use property lists for saving and restoring the application state. This will give the user the illusion that your application does not quit when he/she hits the Home button.

**Appendix B** Here, you will learn how to programmatically invoke iPhone applications from within your application. In addition, you will learn how to publish services that other iPhone applications can utilize.

**Appendix C** This appendix explains the major steps needed to submit your application to the App Store.

**Appendix D** In this appendix, we cover several topics related to using XCode. First, we show some useful shortcuts. Next, we talk about writing custom templates for your classes and after that we cover build configuration. Finally, we show you how to add references to other libraries (also known as frameworks).

**Appendix E** In this appendix, we show you how to add unit tests to your project. By adding unit testing support, you'll be able to write tests for your business logic. These tests will be added as a dependency on the building of your application. This will result in the tests being run before actually building your application. The appendix walks you through a step-by-step process for adding unit testing for a simple business model.

**Appendix F** In this appendix, we use Interface Builder to build a couple of iPhone applications. The techniques you learn from building these applications should prove to be useful in building similar iPhone applications.

# 1

## Getting Started

This chapter serves as a quick introduction to the tools bundled with the SDK. It also shows you basic development steps that include coding, UI design, and debugging. You do not have to understand everything in this chapter as we will go over these concepts throughout the book. What you need to get from this chapter is a feeling of iPhone development using XCode.

We start with some basics of the XCode IDE in Section 1.1. Next, Section 1.2 talks about the UI design tool Interface Builder. After that, we show you how to use the built-in debugger in XCode in Section 1.3. Next, Section 1.4 shows you different sources of information for obtaining additional help. Finally, we summarize the chapter in Section 1.5.

### 1.1 SDK and IDE Basics

In this section, we walk you through the process of creating your first iPhone application. But first, you need to obtain the iPhone SDK and install it on your Mac.

#### *1.1.1 Obtaining and installing the SDK*

Obtaining and installing the iPhone SDK is easy; just follow these steps:

1. Get your iPhone developer Apple ID and password from:  
<http://developer.apple.com/iphone/>
2. Download the latest iPhone SDK for iPhone OS from the site mentioned above.
3. Install the iPhone SDK on your Intel-based Mac.

Now, you're ready to create your first project – read on!

### 1.1.2 Creating a project

Locate XCode and launch it. You can use Spotlight to find it or you can navigate to /Developer/Applications/XCode. XCode is the central application for writing, designing, debugging, and deploying your iPhone applications. You will use it a lot, so go ahead and add it to the Dock.

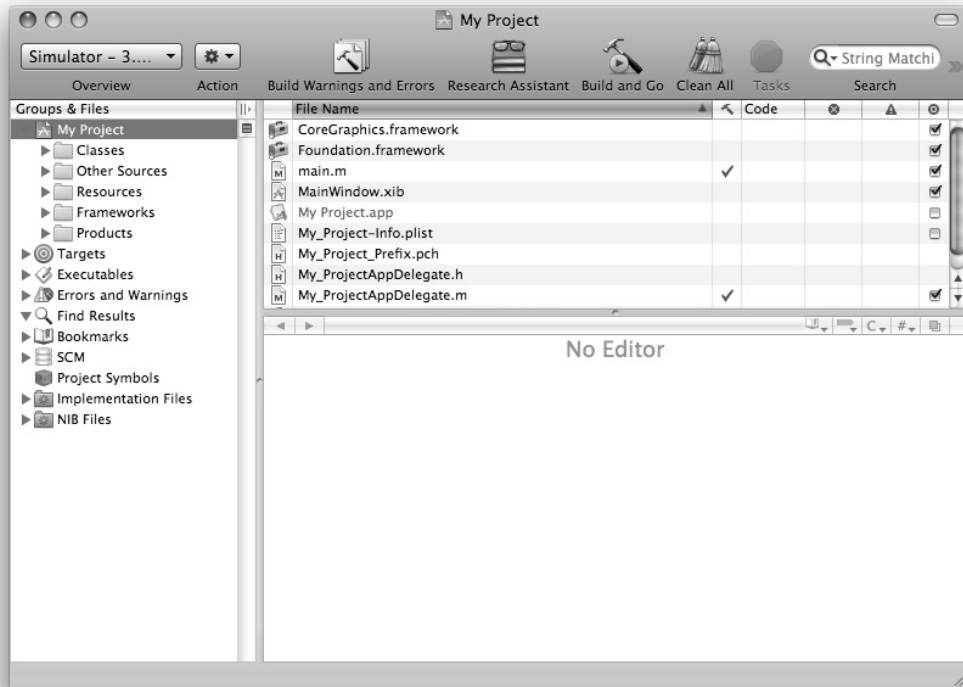
From XCode, select File->New Project. You should see a window, similar to the one shown in Figure 1.1, asking you for the type of project you want to create. Choose the default and create a window-based application. This is the most generic type of iPhone project and the one that can be customized for different needs.



**Figure 1.1** Choosing window-based application in the project creation process.

Click on Choose... and enter the name of your project (here, we're using My Project) and hit Save. A new directory is created with the name you entered, and several files are generated for you. You should now see the newly created iPhone project as in Figure 1.2.

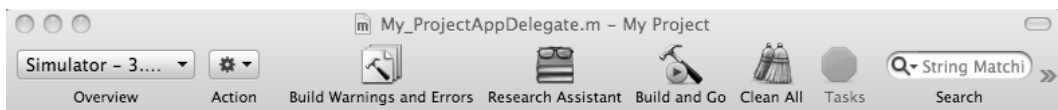




**Figure 1.2** A newly created iPhone project in XCode.

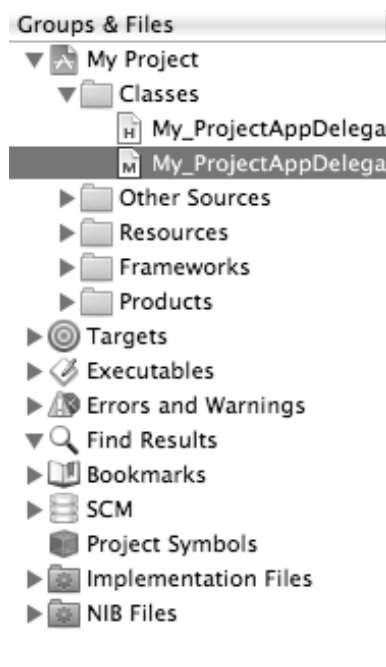
### 1.1.3 Familiarizing yourself with the IDE

As you can see from Figure 1.2, the main window is divided into several areas. On the top, you will find the Toolbar (Figure 1.3). The Toolbar provides quick access to common tasks. It is fully configurable; you can add and remove tasks as you want. To customize the Toolbar, Control-click it and choose *Customize Toolbar...* There, you can drag your favorite task on the Toolbar. Hit Done when you're finished. To remove an item, Control-click on it and choose *Remove Item*.



**Figure 1.3** The XCode Toolbar.

On the left-hand side, you'll see the Groups & Files list (Figure 1.4).

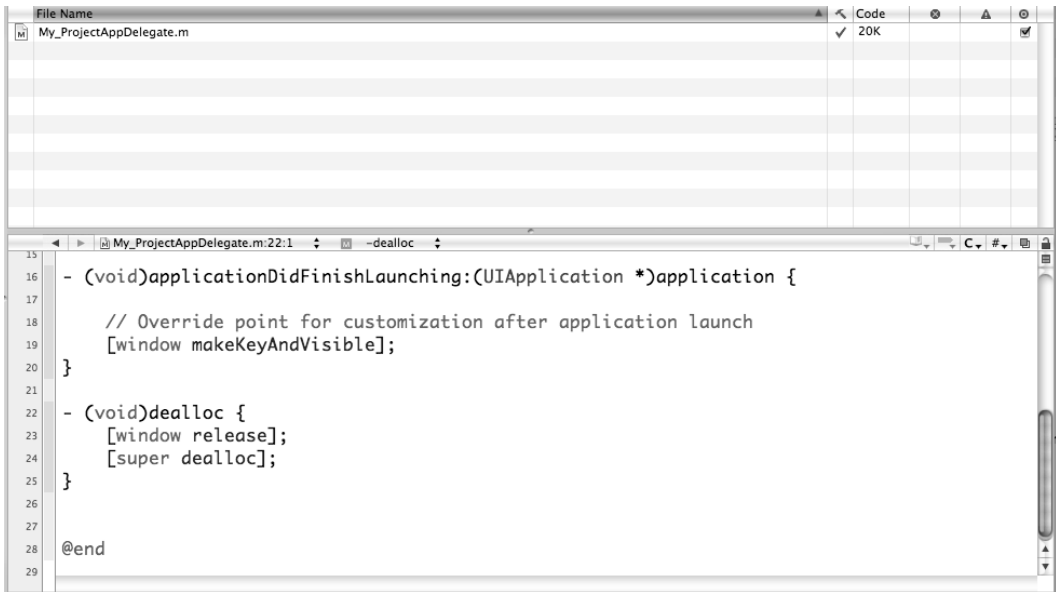


**Figure 1.4** The Groups & Files list in XCode.

This list is used to organize the source code, frameworks, libraries, executables, and other types of files in your project.

The list shows several files and groups. Groups can contain other groups and files. You can delete a group as well as create a new one. The group indicated by the blue icon whose name is the same as the name you've chosen as the project name is a *static group*. Underneath it, you see all your headers, implementations, resources (images, audio files, etc.), and other related files. The folder-like yellow groups act conceptually as containers. You can have containers inside other containers and all files inside these containers live in the same directory on the disk. The hierarchy only helps you organize things. You have full freedom to organize your project's layout as you like. The compiler will pick up the resources, headers, and implementation files when it builds your application.

The other kind of groups that are listed below the project group are called *smart groups*. There are two types of smart groups: 1) built-in smart groups, and 2) custom smart groups. The content of the built-in smart groups cannot be customized. Examples of these groups include executables, bookmarks, errors/warnings, and targets. Customized smart groups are shown in purple, and two predefined groups are created for you when you create a new project.



**Figure 1.5** The Details view with the text editor view.

Figure 1.5 shows the Details view and the text editor beneath it.

Selecting an item in the Groups & Files list will result in its details being shown in the Details view. You can go to a full-editor window using Command-shift-E.

#### 1.1.4 Looking closely at the generated code

Expand the Classes and Other Sources groups. You will notice several files that live underneath these two groups. Click on the `main.m` file and expand to a full-editor view.

The `main.m` file looks very similar to a C file with a `main()` function. As we will see later in this book, all that `main()` does is prepare for memory management and launch the application.

Click on the `My_ProjectAppDelegate.h` file under the Classes group. You will notice that the editor changes its content. This file contains the declaration of the application delegate class. Every application that runs on the iPhone OS has a delegate object that handles critical phases of its lifecycle.

Click on `My_ProjectAppDelegate.m`. This file with the `.m` extension is the counterpart of the previous `.h` file. In it, you see the actual implementation of the application delegate class. Two methods of this class are already implemented for you. The `applicationDidFinishLaunching:` method is one of those methods that handles a particular phase of the application lifecycle. The other

method, `dealloc`, is a method where memory used by this object is released. In iPhone OS, you manage the allocation and freeing of memory as there is no garbage collection. Memory management is crucial in iPhone development, and mastering it is very important. The first chapters are dedicated to teaching you exactly that – and much more.

The generated files and resources are adequate for starting the application. To launch the application, click on `Build and Go` in the Toolbar or press the `Command-Enter` key combination. You'll notice that the application starts in the Simulator and it only shows a white screen with the status bar on top. Not very useful, but it works!

## 1.2 Creating Interfaces

To be useful, an iPhone application needs to utilize the amazing set of UI elements available from the SDK. Our generated iPhone application contains a single UI element: a window.

All iPhone apps have windows (usually one.) A window is a specialized view that is used to host other views. A view is a rectangle piece of real-estate on the  $320 \times 480$  iPhone screen. You can draw in a view, animate a view by flipping it, and you can receive multi-touch events on it. In iPhone development, most of your work goes towards creating views, managing their content, and animating their appearance and disappearance.

Views are arranged into a hierarchy that takes the shape of a tree. A tree has a root element and zero or more child elements. In iPhone OS, the window is the root element and it contains several child views. These child views can in turn contain other child views and so on and so forth.

To generate views and manage their hierarchy, you can use both Interface Builder (IB) and Objective-C code. IB is an application that comes with the SDK that allows you to graphically build your view and save it to a file. This file is then loaded at run-time and the views stored within it come to life on the iPhone screen.

As we mentioned before, you can also use Objective-C code to build the views and manage their hierarchy. Using code is preferred over using IB for the following reasons. First, as beginner, you need to understand all aspects of the views and their hierarchy. Using a graphical tool, although it simplifies the process, does hide important aspects of the process. Second, in advanced projects, your views' layouts are not static and change depending on the data. Only code will allow you to manage this situation. Finally, IB does not support every UI element all the time. Therefore, you will sometimes need to go in there and generate the views yourself.

The following section teaches you how to use IB. However, for the most part in this book, Objective-C code is used to illustrate the UI concepts. For extensive coverage of Interface Builder, please see Appendix F.

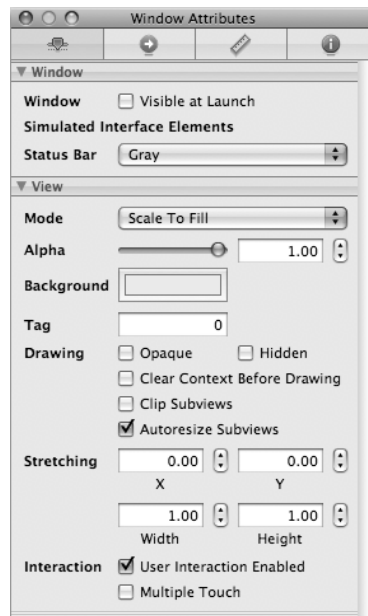
### 1.2.1 Interface Builder

The project has a basic window resource file. This file can be found under the `Resources` group. Expand the `Resources` group and locate the file `MainWindow.xib`. This file contains the main window of the application. This file is an `.xib` file that stores the serialized objects in the interface. When the project is built, this file is converted to the more optimized format `.nib` and loaded into memory when one or more of the UI components stored in it are requested.

Double-click on the `MainWindow.xib` file to launch IB. IB starts by opening four windows. The first window shows the main window stored in the file. The second window shows the document window listing the different objects stored in the file. The third window is the Library window containing all the UI objects that you can add to the file. The fourth and final window is the Inspector window with its four panes.

The Inspector window shows the attributes of the currently selected object. If you click on an object, the Inspector window shows you its attributes distributed among four different panes. Each pane has several sections. You can change these attributes (such as color, position, and connections) and the changes will propagate to your project's user interface.

The main window of the application is white; let's change it to yellow. Click on the window object in the document window. In the Inspector window, make sure that the left-most pane is selected. In the `View` section of this pane, change the background color to yellow as shown in Figure 1.6.



**Figure 1.6** The attributes pane in the Inspector window of Interface Builder.

Go to XCode and run the application. Notice how the main window of the application has changed to yellow. It is important to keep the project open in XCode while working with IB. XCode and IB communicate well when both applications are open.

To build a user interface, you start with a view and add to it subviews of different types. You are encouraged to store separate views in separate `.xib` files. This is important as referencing one object in a file will result in loading all objects to main memory. Let's go ahead and add a label view to our window. This label will hold the static text "Hello iPhone."

A label is one of the many UI components available for you. These components are listed under several groups in the **Library**. Locate the **Library** window and click on **Inputs & Values** as shown in Figure 1.7.

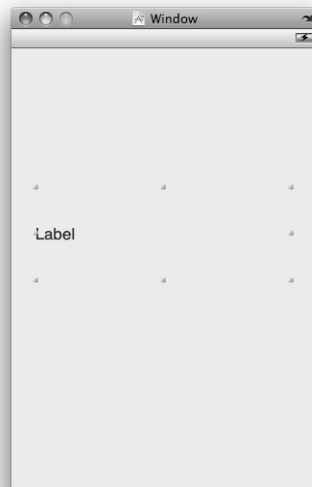


**Figure 1.7** The Library window of Interface Builder.

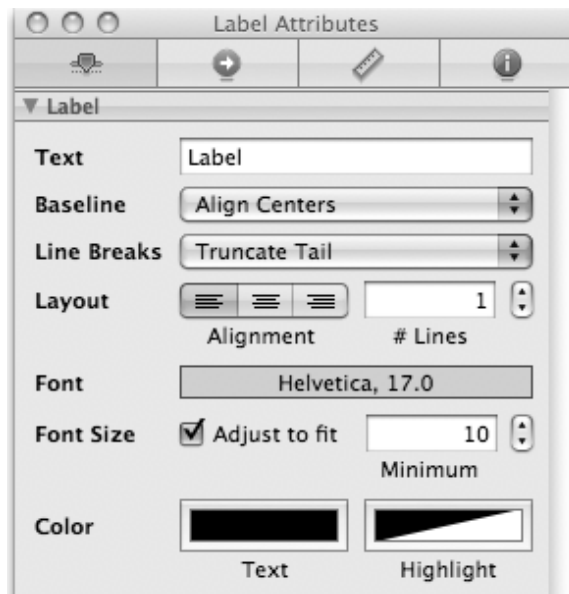
Click on the `Label` item and drag it onto the middle of the window. Expand the dimensions of the label as shown in Figure 1.8.

When the label is selected, the **Inspector** window changes to reflect the attributes of the label. Figure 1.9 shows a portion of the attributes of a label in the **Inspector** window. You can change these attributes and observe the effect they have on the object instantaneously.

The label's text is left justified; let's make it center. In the **Layout** item of the attributes, click on the icon indicating center. Notice how the label text becomes centered. The text of the label can be changed in the **Text** item. Change `Label` to `Hello iPhone`. Go to XCode and hit **Build and Go**. You will notice the window showing `Hello iPhone` in the middle.



**Figure 1.8** Adding a label view to a window in IB.



**Figure 1.9** Attributes of a label in the Inspector window.

The text of the label is small, so let's make it bigger. Click on the `Text` item and choose a text size of 48 points. Go to XCode and hit `Build` and `Go`. Figure 1.10 shows a screenshot of the completed `Hello iPhone` application.



**Figure 1.10** A screenshot of the completed `Hello iPhone` application.

Congratulations on your first successful iPhone application!

You deliver the product to the client and he is happy. However, he wants the application to have more interaction with the user. He asks you to revise the application by adding a button that the user can tap on to change the text displayed in the label.

Open the `MainWindow.xib` document if it is not already open. Locate the `Round Rect Button` item under `Items & Values` in the `Library` window. Drag and drop it under the label in the main window. Change the button's title by entering "Change" in the `Title` field found in the fourth section of the attributes window. The main window should look like the one shown in Figure 1.11.

Now that we have a button, we want to have a method (a function) in our code to get executed when the user touches the button. We can achieve that by adding a connection between the button's touch event and our method.





**Figure 1.11** The main window after adding a new button.

Click on the button so that it becomes selected. Click on the second pane in the Inspector window. This pane shows the connections between an object and our code. The pane should look like the one in Figure 1.12.



**Figure 1.12** The connections pane of our new button.

Now, we want to add a connection between the Touch Down event and a method we call `buttonTapped`. Let's first add this method in `My_ProjectAppDelegate` class.

In the `My_ProjectAppDelegate.h` file, add the following before **@end**.

```
-(IBAction)buttonTapped;
```

In the `My_ProjectAppDelegate.m` file, add the `buttonTapped` method body. The `My_ProjectAppDelegate.m` file should look something like the one in Listing 1.1.

---

**Listing 1.1** The application delegate class after adding a new method.

---

```
#import "My_ProjectAppDelegate.h"
@implementation My_ProjectAppDelegate
@synthesize window;
- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    [window makeKeyAndVisible];
}

- (IBAction)buttonTapped{
    UILabel *label = (UILabel*)[window viewWithTag:55];
    if([label.text isEqualToString:@"Hello iPhone"])
        label.text = @"Hello World";
    else
        label.text = @"Hello iPhone";
}

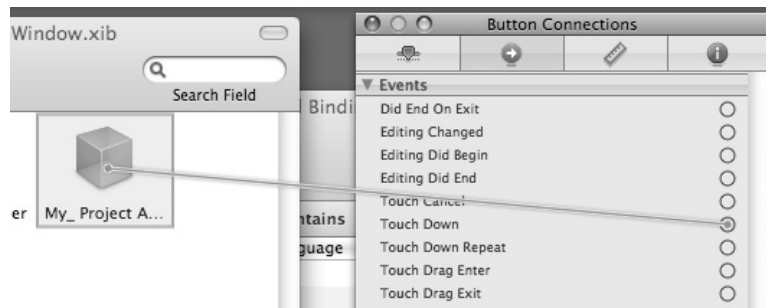
- (void)dealloc {
    [window release];
    [super dealloc];
}
@end
```

---

The `buttonTapped` method simply obtains a reference to the label and changes its text to either "Hello World" or "Hello iPhone". You don't need to understand this code at this stage. All you need to understand is that the label on the screen is encapsulated by the `UILabel` class and it's tagged with the number 55.

Now, let's switch to IB and add a tag to the label so that it can be retrieved from the code. Click on the label and in the Inspector window, choose the first pane. In the second section, enter 55 for the Tag field (fourth item.)

We still need to perform one last step. We need to connect the touch event with the method we just created. Click on the button and choose the connections pane (second pane). Control-click or right-click on the circle on the right-hand side of Touch Down event and drag it on top of the `My_ProjectAppDelegate` object in the Document window and let go as shown in Figure 1.13.



**Figure 1.13** Making a connection between an event and a method in another object.

When you release the mouse, IB shows you potential methods (actions) that you can connect this event to. Right now we only have one action and that action is `buttonTapped`. Select that action and you'll notice that a connection has been made as shown in Figure 1.14.



**Figure 1.14** A connection between a touch event and an action.

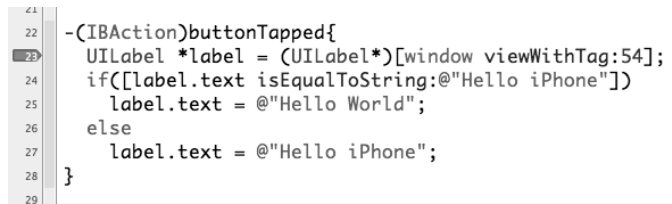
Now, switch to XCode and hit **Build** and **Go**. You'll notice that tapping on the button changes the text value of the label.

### 1.3 Using the Debugger

During the development of your applications, often things go wrong and the feature that you've just added is not functioning properly. At these moments, the built-in debugger becomes invaluable.

Let's introduce a bug into our code. Go to `My_ProjectAppDelegate.m` file and change the tag's value used to obtain the label from 55 to 54, then **Build** and **Go**. Now, tapping the button has no effect on the label's text.

First, you want to make sure that the `buttonTapped` method gets called. In XCode, click in the left margin of the first line in the `buttonTapped` method as shown in Figure 1.15. After you click there, a breakpoint (shown in blue) is added.



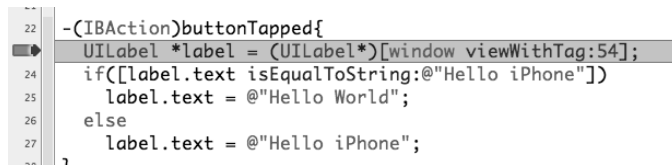
```

21
22 -(IBAction)buttonTapped{
23     UILabel *label = (UILabel*)[window viewWithTag:54];
24     if([label.text isEqualToString:@"Hello iPhone"])
25         label.text = @"Hello World";
26     else
27         label.text = @"Hello iPhone";
28 }
29

```

**Figure 1.15** Adding a breakpoint in the `buttonTapped` method.

Click **Build** and **Go** to debug the application. When the application launches, tap on the button. You'll notice that the execution hits the breakpoint as shown in Figure 1.16. At least we know that we made our connection correctly.



```

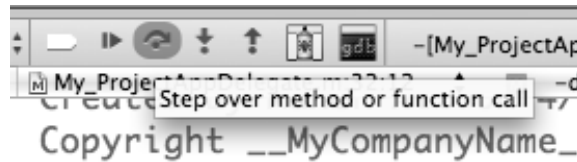
22 -(IBAction)buttonTapped{
23     UILabel *label = (UILabel*)[window viewWithTag:54];
24     if([label.text isEqualToString:@"Hello iPhone"])
25         label.text = @"Hello World";
26     else
27         label.text = @"Hello iPhone";
28 }
29

```

**Figure 1.16** Hitting a breakpoint in the `buttonTapped` method.

Let's step over the statement that obtains the label from the window. Click on the **Step Over** button located beneath the Toolbar as shown in Figure 1.17.

After stepping over the statement, we need to inspect the value obtained. Hover the mouse over `label` in the statement just executed as shown in Figure 1.18. A tip appears showing its value. Notice that the value is `0x0`. In Objective-C, this value is called `nil` and means that no object is stored in this variable. After inspecting the tag value and going back-and-forth between XCode and IB, we find the problem, fix it, remove the breakpoint by clicking on it to turn it off, and hit **Build** and **Go**.



**Figure 1.17** Step over a function or a method call button.

```

-(IBAction)buttonTapped{
    UILabel *label = (UILabel *)[window viewWithTag:54];
    if([label.text isEqualToString:@"Hello iPhone 0x0"]){
        label.text = @"Hello World";
    }
    else
        label.text = @"Hello iPhone";
}

```

**Figure 1.18** Inspecting the value of the label after obtaining it from the window.

## 1.4 Getting More Information

There are plenty of sources for information on the SDK. These sources include the following:

- **Developer Documentation.** The best locally stored source of information is the Developer Documentation. In XCode, select Help->Documentation. The documentation window appears as shown in Figure 1.19. You can search using the search box on the left-hand corner for any defined type in the SDK. The documentation is hyper-linked and you can go back-and-forth between different pieces of information. It's easy to use and it will become your friend.
- **Developer Documentation from within XCode.** If you're in XCode and you need more information about something, Option-double-click it and the Developer Documentation opens with more information.
- **Other help from within XCode.** If you're in XCode and you need to get the declaration and possible implementation of a given token (e.g., class, tag, variable, etc.), Command-double-click it. If there are multiple pieces of information, or disambiguation is needed, a list of items to choose from will be shown.
- **iPhone Dev Center.** The center is located at <http://developer.apple.com/iphone/>. The iPhone Dev Center has a large collection of technical resources and sample code to help you master the latest iPhone technologies.
- **Apple's Fora.** You can start with the site at <https://devforums.apple.com/>.
- **The Web.** There is plenty of information on the web. Just enter a relevant query and let Google do its magic!

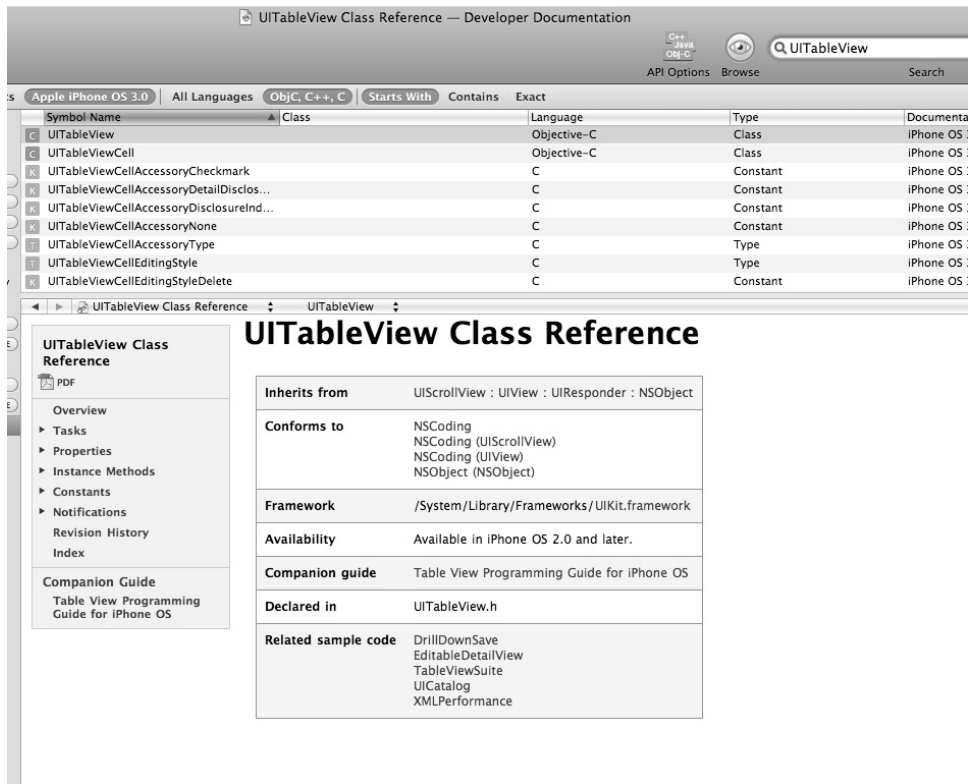


Figure 1.19 The Developer Documentation in XCode.

## 1.5 Summary

This chapter provided a gentle introduction to the world of iPhone development. We showed you in Section 1.1 how to download and install the iPhone SDK. After that, we iterated through the development of an iPhone application and showed you how to use Interface Builder to design user interfaces. Next, Section 1.3 discussed how to debug an iPhone application using the built-in visual debugger in XCode. You were also exposed to different sources for obtaining further help on the tools and the SDK in general in Section 1.4.

The rest of the book will detail all aspects of iPhone development. However, from now on, since we want to teach you everything you need, we will stop using Interface Builder and show you how to build your UI using code. This will help you gain a solid understanding of the process. You can, of course, mix and match with Interface Builder as you wish.

The next two chapters cover the Objective-C language and the coding environment that you will be working with: Cocoa. We hope you're as excited as we are!

## Problems

- (1) Check out the `UILabel.h` header file and read about the `UILabel` class in the documentation.
- (2) What's an `IBOutlet` and `IBAction`? Use Command-double-click to see their definitions in the `UINibDeclarations.h` header file.
- (3) Explore the XCode IDE by reading the `XCode Workspace Guide` under the `Help` menu of the XCode application.
- (4) Explore Interface Builder by choosing `Interface Builder Help` under the `Help` menu of the Interface Builder application.

