Chapter 1

The iPhone Software Development Kit (SDK)

Key Skills & Concepts

- Understand the App Store
- Understand how to obtain Xcode and the iPhone SDK
- Understand if this book is right for you
- Understand Xcode's help and Apple's online documentation
- Understand this book is about User Interface controls and using Interface Builder

I am mostly a loner—nobody calls me—so why do I pay over 100 dollars a month for an iPhone? It is a darn useful toy—I mean tool. The last time I got lost, I started the Maps application, and within seconds, it had located my position and provided me with a map. I can check my e-mail anywhere, and the last time I needed to impress my friends, I bought and installed the iFart application. I use the iPod app to listen to music, and every once in a long while, someone calls.

As well as being mostly a loner, I am also an old guy and not the best candidate for expounding the iPhone's many virtues. For instance, I think texting is a time-waster. But my 14-year-old nephew, here on vacation the other week, certainly didn't think so. He spent the majority of his time texting friends back home. About what, who can guess, but he did it, and my brother was paying for it. I should also mention he downloaded apps from the App Store, and my brother was paying for those, too.

If you want some of my nephew's money—I mean my brother's money—you can get some by writing and selling an iPhone application on the App Store. Unfortunately, the sure path to riches, iPhone pornography and iPhone gambling, is off limits on the App Store, but there are plenty other applications you might write.

The App Store

The App Store is a unique concept. The App Store is an Apple application on iPhones and iPod touches. You use the App Store to browse and download applications from Apple's iTunes Store. Some applications are free, while others have a (usually) nominal charge. Using your iTunes account, you can download applications directly to your iPhone or iPod Touch. What I like is that I can use an iTunes Gift Card that I can buy at my local grocery store; no credit card needed.

Don't know what to buy? You can go to one of the many Web sites dedicated to reviewing applications on the App Store. For instance, www.appstoreapps.com (Figure 1-1) provides reviews of both free and paid applications. Most applications are junk, but some are quite good.

Downloading applications from the App Store is both easy and inexpensive. That makes it a lucrative market for independent developers wishing to take advantage of the iTunes Store's large user base. Independent developers can develop applications for the App Store by

Figure 1-1 The appstoreapps.com Web site reviews most App Store applications.

downloading the iPhone SDK, developing an application, and joining the iPhone Developer Program. Apple then reviews your application, and if it passes the review process, it is added to the iTunes Store.

The Software Development Kit (SDK)

So you have decided to try your hand at developing applications for the App Store. The first thing you must do if you wish being an iPhone developer is register as a member at the iPhone Dev Center at http://developer.apple.com/iphone. Membership is free and allows downloading the SDK.

The second thing you must do, arguably the first, is install Xcode and the iPhone SDK by downloading it from Apple's Developer Connection. Step-by-step installation instructions are available on Apple's Web site. After installing the iPhone SDK, the absolute next thing you should do is start Xcode and download the documentation—all the documentation (Figure 1-2). It will take awhile, but it is well worth it.

Figure 1-2 The iPhone Reference Library in Xcode

NOTE

You will find Apple's documentation surprisingly complete and well written. I refer to this documentation often in this book, so it is best to download it before continuing.

Paid Membership

Testing applications on an iPhone or iPod touch and selling applications on the App Store require that you register with the iPhone Developer Program. This membership is different from membership to the iPhone Dev Center. The iPhone Developer Program for individuals costs $99 and entitles you to the tools needed to test on an iPod touch or iPhone. It is also how you submit and distribute your application to the App Store, and Apple distributes any profit you might earn through your iPhone Developer Program membership.

Objective-C, Foundation Framework, Cocoa Touch, and UIKit

Apple describes the iPhone's technology as layers. The base layer is the Core OS layer. On top of that layer is the Core Services. On top of the Core Services is the Media layer. The topmost layer is Cocoa Touch (Figure 1-3).

You can simplify the iPhone operating system (OS) even more; think of it as two layers—a C layer and a Cocoa layer (Figure 1-4). The C layer comprises the operating system's layer. You use BSD UNIX–style C functions to manipulate this layer. This layer consists of things like low-level file I/O, network sockets, POSIX threads, and SQLite. The Media layer is also rather low-level and contains C application programming interfaces (APIs) like OpenGL ES, Quartz, and Core Audio. The Cocoa layer overlays the C layer, and it simplifies iPhone programming. For instance, rather than manipulating C strings, you use the Foundation framework string, NSString.

Cocoa Touch

On the iPhone, Cocoa is called Cocoa Touch, rather than simply Cocoa, because the iPhone OS contains touch events. If you have ever tapped, flicked, swiped, or pinched your iPhone's display, you know what touch events are. Touch events allow you to program responses to a user's touching the screen with his or her fingers.

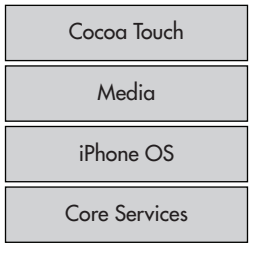

Figure 1-3 The iPhone's technology layers

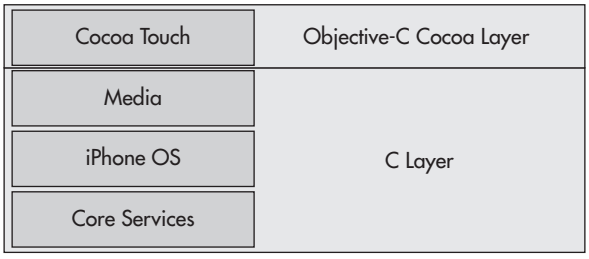

Figure 1-4 The iPhone's programming layers

Cocoa Touch also provides the primary class libraries needed for iPhone development. The two Cocoa Touch frameworks you will use in every iPhone application you write are the Foundation framework and the UIKit framework. A framework is collection of code devoted to a similar task. The Foundation framework is dedicated to standard programming topics, such as collections, strings, file I/O, and other basic tasks. The UIKit is dedicated to the iPhone's interface and contains classes such as the UIView. In this book, you spend most your time learning the UIKit.

Foundation Framework

The Foundation framework contains Objective-C classes that wrap lower-level core functionality. For instance, rather than working with low-level C file I/O, you can work with the NSFileManager foundation class. The Foundation framework provides many useful classes that you really should learn if you wish programming robust iPhone applications. The Foundation framework makes programming using collections, dates and time, binary data, URLs, threads, sockets, and most other lower-level C functionality easier by wrapping the C functions with higher-level Objective-C classes.

TIP

See Apple's Foundation Framework Reference for a complete listing of the classes and protocols provided by the Foundation framework.

NOTE

If you are a Java programmer, think of the iPhone's programming environment like this: Objective-C is equivalent to Java's core syntax. The Foundation framework is equivalent to Java's core classes, such as ArrayList, Exception, HashMap, String, Thread, and other Java Standard Edition classes, and the UIKit is the equivalent of SWING. I realize it's a simplification, but it works for me.

The iPhone Frameworks

Table 1-1 lists the frameworks available to you as an iPhone developer. Of these frameworks, this book dedicates itself to the UIKit rather than trying to cover a little bit of every framework.

Framework	Purpose
AddressBook	Accessing user's contacts
AddressBookUI	Displaying Addressbook
AudioToolbox	Audio data streams; playing and recording audio
AudioUnit	Audio units
CFNetwork	WiFi and cellular networking
CoreAudio	Core audio classes
CoreFoundation	Similar to Foundation framework, but lower level (don't use unless you absolutely must)
CoreGraphics	Quartz 2D
CoreLocation	User's location/GPS
Foundation	Cocoa foundation layer
MediaPlayer	Video playback
OpenAL	Positional audio library
OpenGLES	Embedded OpenGL (2-D and 3-D graphics rendering)
QuartzCore	Core animation
Security	Certificates, keys, and trust policies
SystemConfiguration	Network configuration
UIKit	iPhone user interface layer

Table 1-1 Frameworks on the iPhone

It is this book's premise that once you understand how to create an iPhone application using the UIKit classes, you should learn the other frameworks.

iPhone Limitations

If you have never programmed for a small device like an iPhone, there are some limitations you should be aware of before you begin programming. Memory and processor speed are constrained, and the screen is small. Security is also tight on an iPhone, and applications are limited in what they can do.

Memory and Processor Speed

An iPhone's memory is constrained. Chances are, you have a Mac with a dual-core and 2GB of memory. Not so on the iPhone. Although Apple hasn't divulged this information, according to hacker Craig Hockenberry of furborg.org, he has estimated that an iPhone has about a 600 MHz processing speed with 128MB of available physical memory. The memory of the device is limited compared to your desktop.

CAUTION

If your application uses too much memory, the iPhone OS X may abruptly terminate your application to prevent a system crash.

Small Screen

An iPhone screen is 480 × 320 pixels. There is not much room to work with. Of course, controls such as buttons are smaller on an iPhone, but the layout space is still significantly constrained. If you are accustomed to programming user interfaces on a 1280 × 800 pixel display, you must adjust your thinking. Screen size is limited.

The small screen size also results in only one window being visible at a time. In fact, every application you develop in this book consists of one window. There will rarely be any reason to create another window when programming an iPhone application. Instead, what you do is swap views into and out of an application's window. But only one view is visible at a time—no exceptions. This restriction is sensible, as the screen is so small.

Security

You can only read or write to directories that are part of your application's bundle. Areas accessible to your application are said to be in your application's sandbox. You cannot read files created by other applications. You also cannot write to anywhere outside your application's sandbox. Applications written by SDK users cannot share resources, period.

Short-Lived Applications

Another iPhone application limitation is that it cannot be memory-resident. A memory-resident application can run in the background while a user runs other applications. Forget about memory-resident applications when programming for the iPhone. You can't do it.

An iPhone can only have one program running at once. This restriction puts your application in constant danger of the OS terminating it. Think about it: Allegedly, an iPhone's primary purpose is still that of a cellular phone. A phone call might arrive while your application is running. In this situation, the OS asks a user if he or she wishes answering the call. If the user chooses to answer the call, the iPhone OS terminates your application.

Because of this constant probability of sudden termination, you should program defensively and anticipate abrupt terminations. You will see that the UIKit makes this easy by providing event handlers you can implement whenever your application is about to terminate.

NOTE

Before you rail against Apple on this limitation, consider the alternative. Suppose you develop a long-running and battery-eating application that is memory-resident. Your application's users notice a short battery life for their iPhone. Who do they blame, you or Apple? Apple.

Manual Memory Management

One of the big improvements in Objective-C 2.0 is garbage collection. Garbage collection frees developers from having to worry about memory management, as the system does so automatically. But the iPhone, with its limited resources, does not include Objective-C 2.0 garbage collection. You must still manage memory yourself. You can use something called autorelease, which makes memory management a little easier, but even autorelease is not recommended. Instead, you should manage memory manually. Although not a huge limitation, it is a pain, as forgetting to release an object is all too easy a mistake to make. Of course, as you will see in Chapter 5, there are tools to help you track down and fix these errors.

Relevant Documentation

Apple has considerable online documentation. You have access to that documentation through both your Developer Connection membership and through Xcode's help. You should refer to that documentation often. Most documentation is also available as PDF documents. The first two documents you should download and print are the iPhone Application Programming Guide and iPhone Development Guide. You might then consider downloading and printing Cocoa Fundamentals Guide. You will also find documents on Objective-C and various Cocoa classes. If you followed this chapter's earlier recommendation and downloaded the documentation, you will find that all this information is at your fingertips using Xcode's help. This book tries not to duplicate these online and desktop sources, but rather complement them by providing step-by-step examples illustrating how to do things. Once you understand how, the online documentation shows you more options to expand upon this book's tutorial.

Try This Getting a Quick Start on iPhone Development

This chapter ends with a quick-start example to whet your appetite. The next four chapters cover prerequisites that you should have prior to learning the iPhone's UIKit and Cocoa Touch. But you are probably ready to start programming using these frameworks now, so this chapter ends with a simple iPhone application. This quick start also familiarizes you with the IBOutlet and IBAction keywords and their use, and it familiarizes you with Xcode and Interface Builder.

NOTE

Almost every Try This example in this book has an accompanying video available at my Web site (www.jamesabrannan.com). The first video—this Try This application—has accompanying audio explaining the steps taken. The remaining videos have no sound; however, they follow their corresponding Try This application's numbered steps exactly, so you can follow the video by referring to the book.

(continued)

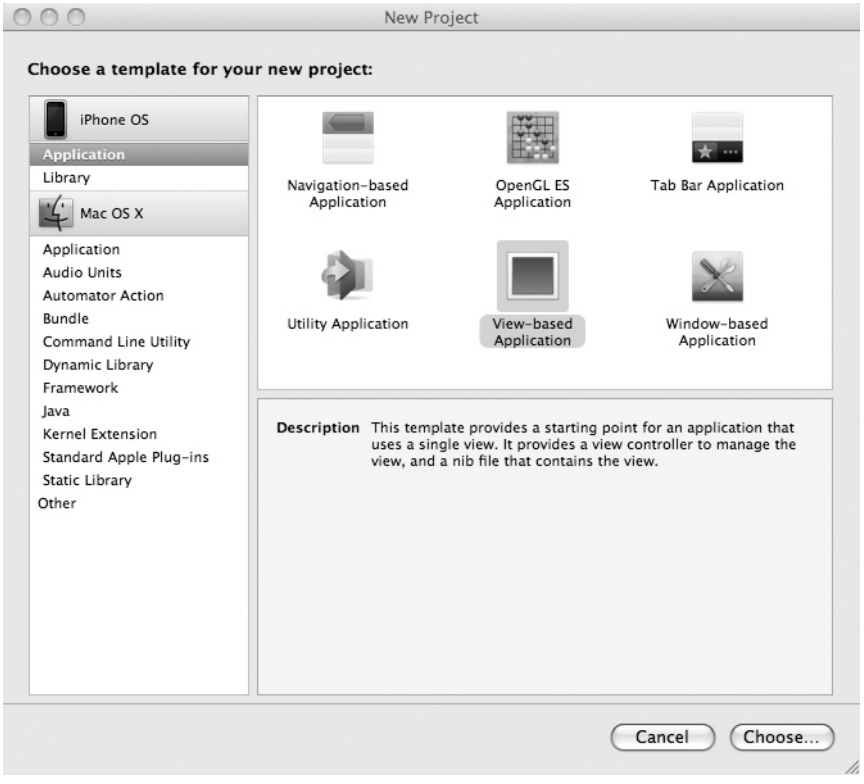

Figure 1-5 New Project dialog

1. Open Xcode. From the menu select File | New Project and the New Project dialog appears (Figure 1-5).
2. Select View-based Application and click Choose. In the Save As dialog, give the application the name QuickStart (Figure 1-6).

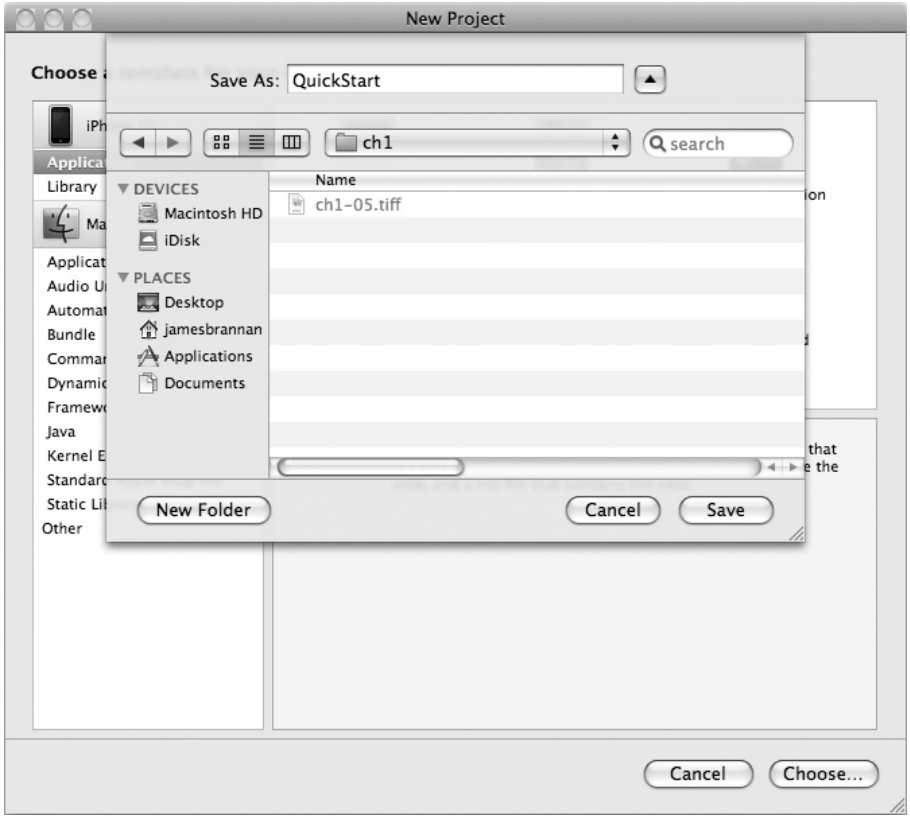

Figure 1-6 Save As dialog

3. Xcode should create the project. In the Groups & Files pane, expand the Classes and Resources folders (Figure 1-7).
4. Double-click QuickStartViewController.xib to open it in Interface Builder.

(continued)

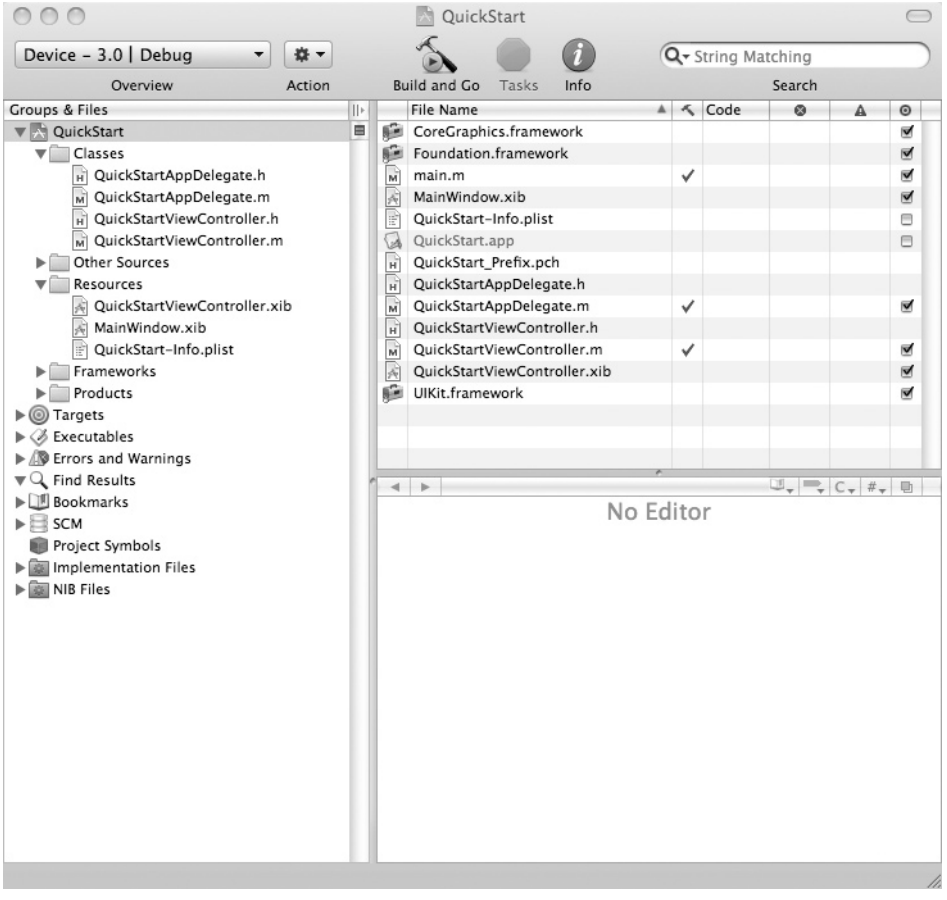

Figure 1-7 Xcode with Classes and Resources folders expanded

5. If a canvas like the one shown in Figure 1-8 is not visible, double-click View in the document window (Figure 1-9).
6. Ensure the library is visible by selecting Tools | Library from Interface Builder's main menu. Ensure the library shows all Cocoa Touch classes by going to the library's top pane, expanding Library, and clicking Cocoa Touch (Figure 1-10).

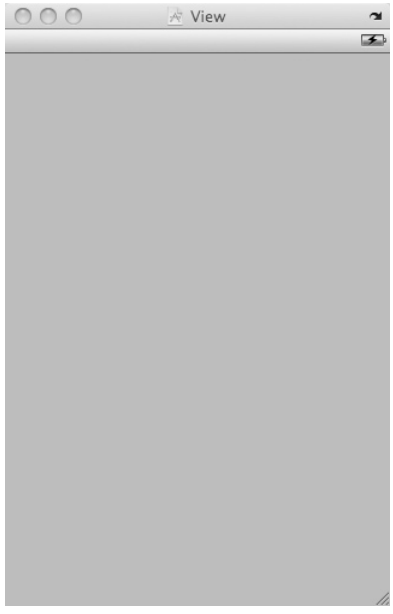

Figure 1-8 A view's canvas in Interface Builder

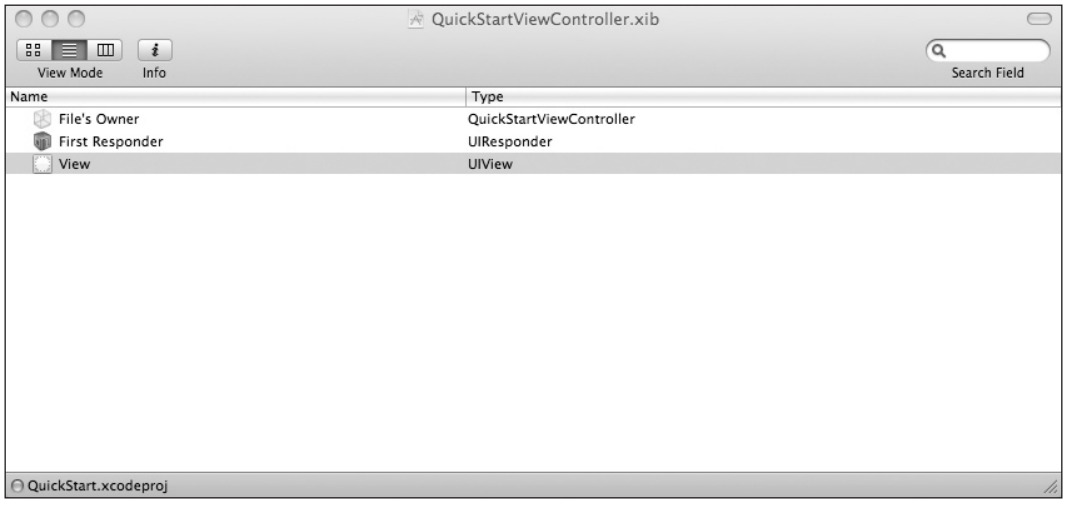

Figure 1-9 The document window

(continued)

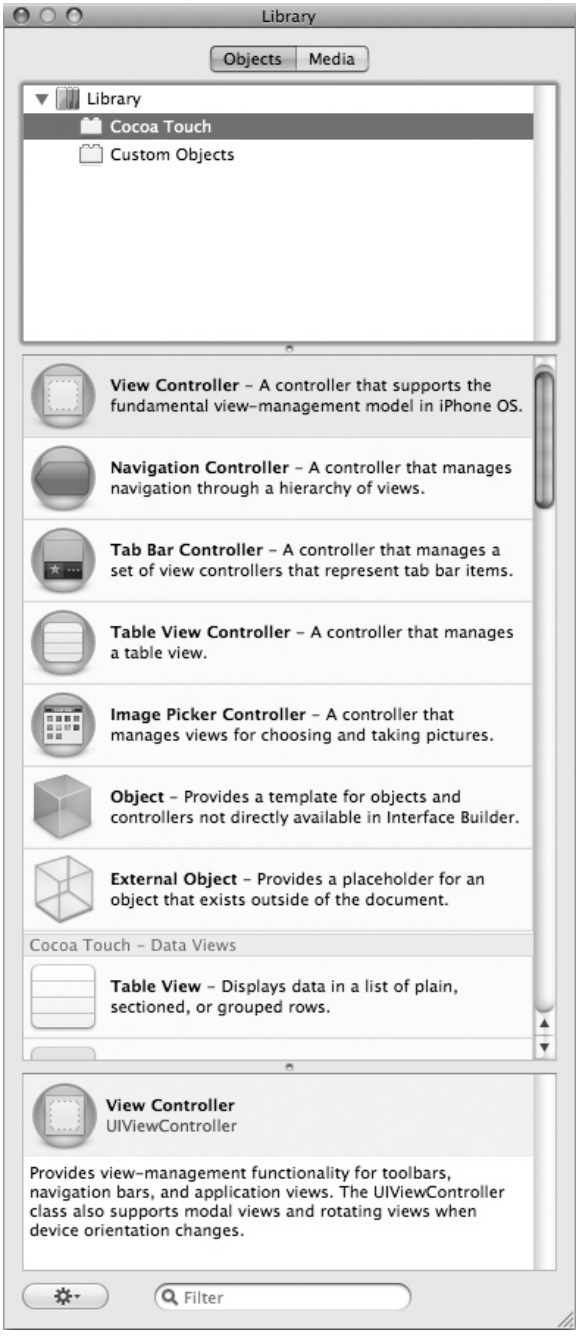

Figure 1-10 The library

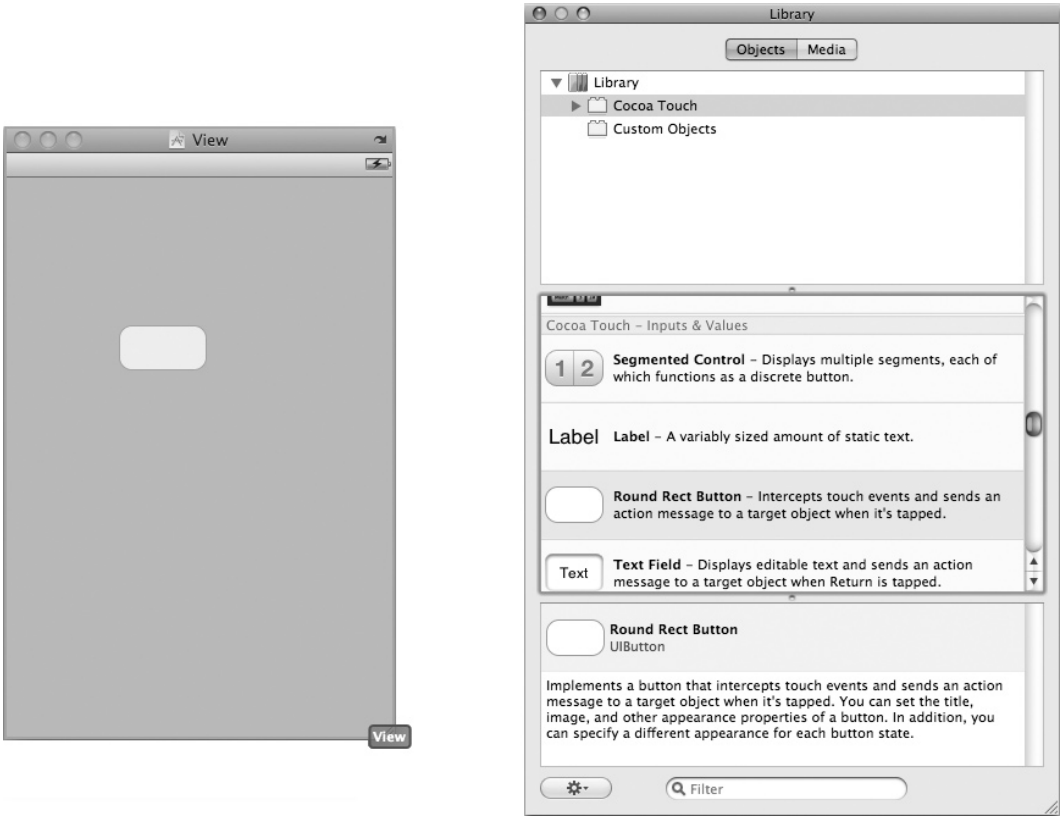

Figure 1-11 Adding a button

7. Scroll through the controls until you find a Round Rect Button. Drag-and-drop the button to the canvas (Figure 1-11).
8. Double-click the button on the canvas, and give the button a title.
9. Drag a label from the library to the canvas (Figure 1-12).

(continued)

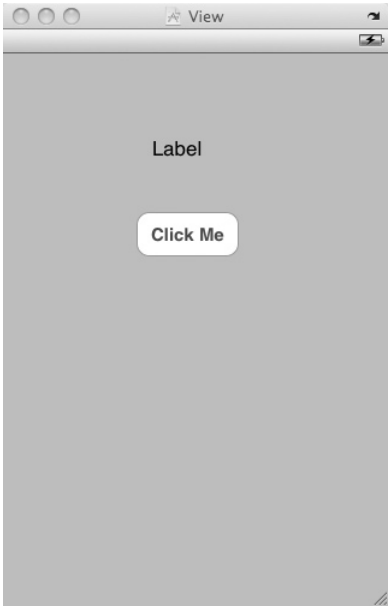

Figure 1-12 Adding a label

10. Save and exit Interface Builder.
11. Select QuickStartViewController.m in the Classes folder in Groups & Files. Xcode should display the file in the editor pane (Figure 1-13).
12. Change QuickStartViewController.m so it matches Listing 1-1.
13. Open QuickStartViewController.h and modify the file so it matches Listing 1-2.
14. Select Build | Build from Xcode's main menu to build the application.
15. Open QuickStartViewController.xib in Interface Builder.

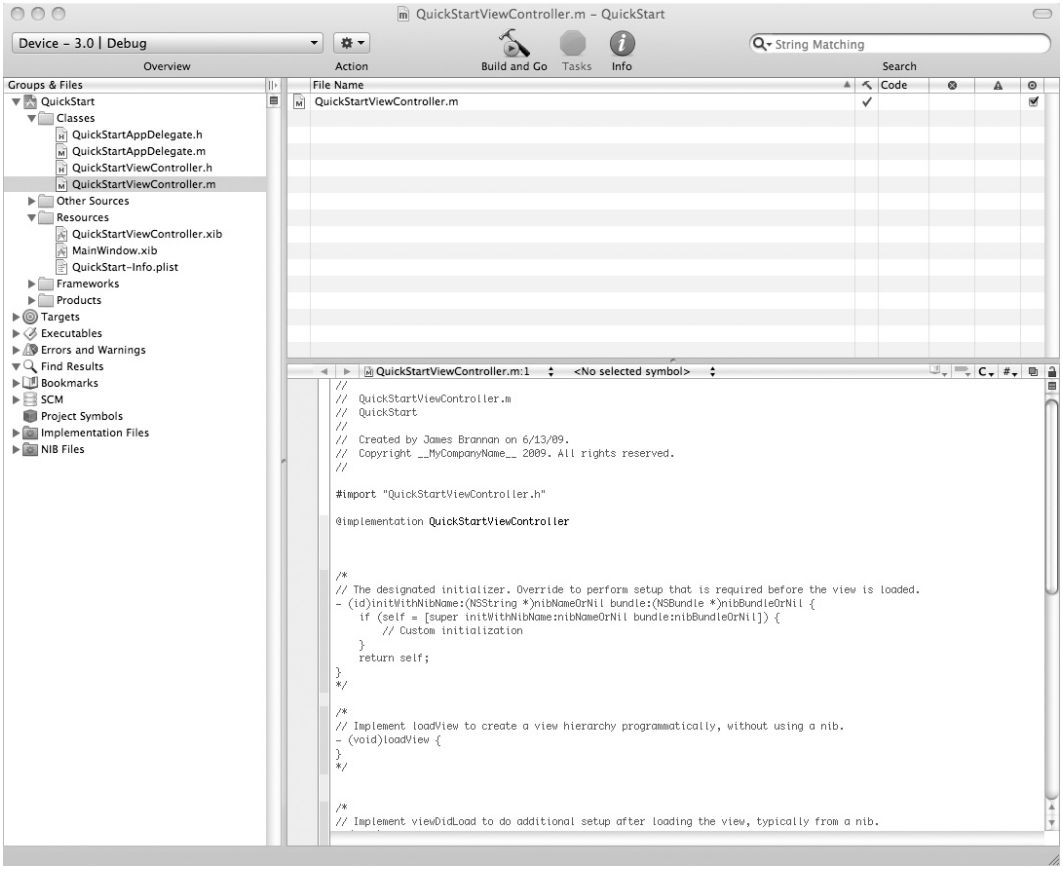

Figure 1-13 Xcode displaying QuickStartViewController.m

16. Select the button. Select Tools | Inspector from Interface Builder's main menu to show the Inspector (Figure 1-14). Open the Button Inspector by clicking the inspector's second tab (Figure 1-15).

17. Next to Touch Up Inside, click and hold on the little circle. Move your cursor to File's Owner in the document window and release. Select sayHello: from the pop-up window (Figure 1-16).

(continued)

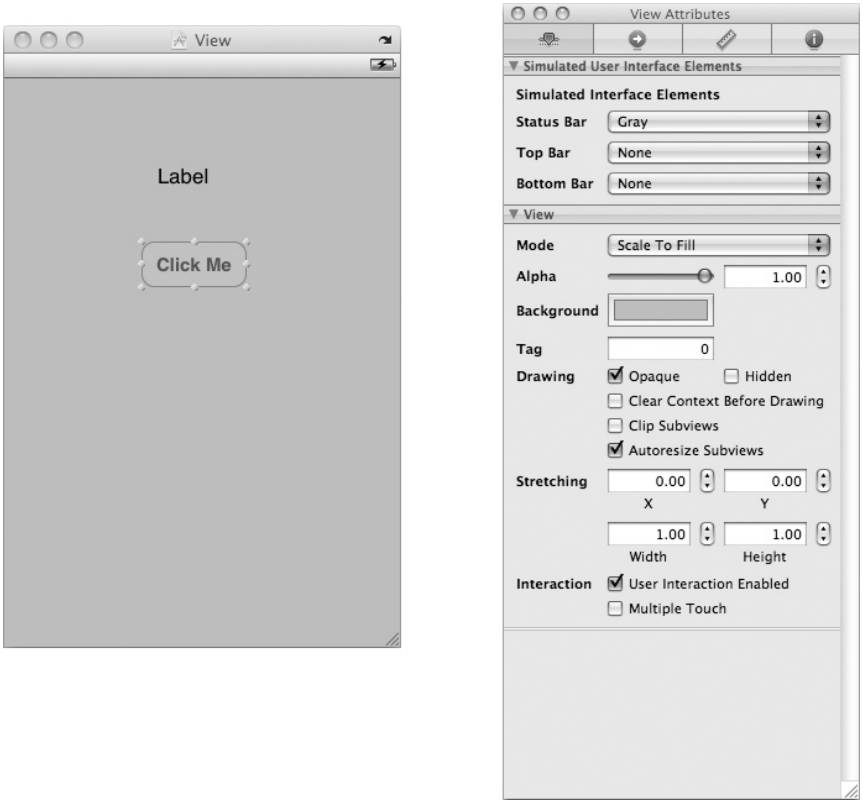

Figure 1-14 The Inspector

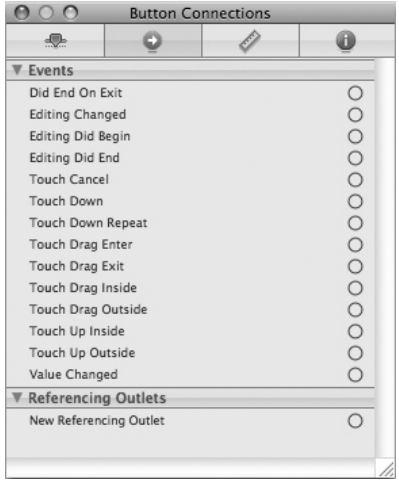

Figure 1-15 The button's Button Inspector

Figure 1-16 Connecting a button to an IBAction

(continued)

18. Click the label on the canvas, and the Inspector's content should change to match the label. Click the circle next to New Referencing Outlet, and drag-and-drop on the File's Owner. Select myLabel from the pop-up window. Be careful not to select View.
19. Save and exit Interface Builder.
20. In Xcode, ensure the Active SDK shows the Simulator and Debug options selected (Figure 1-17).

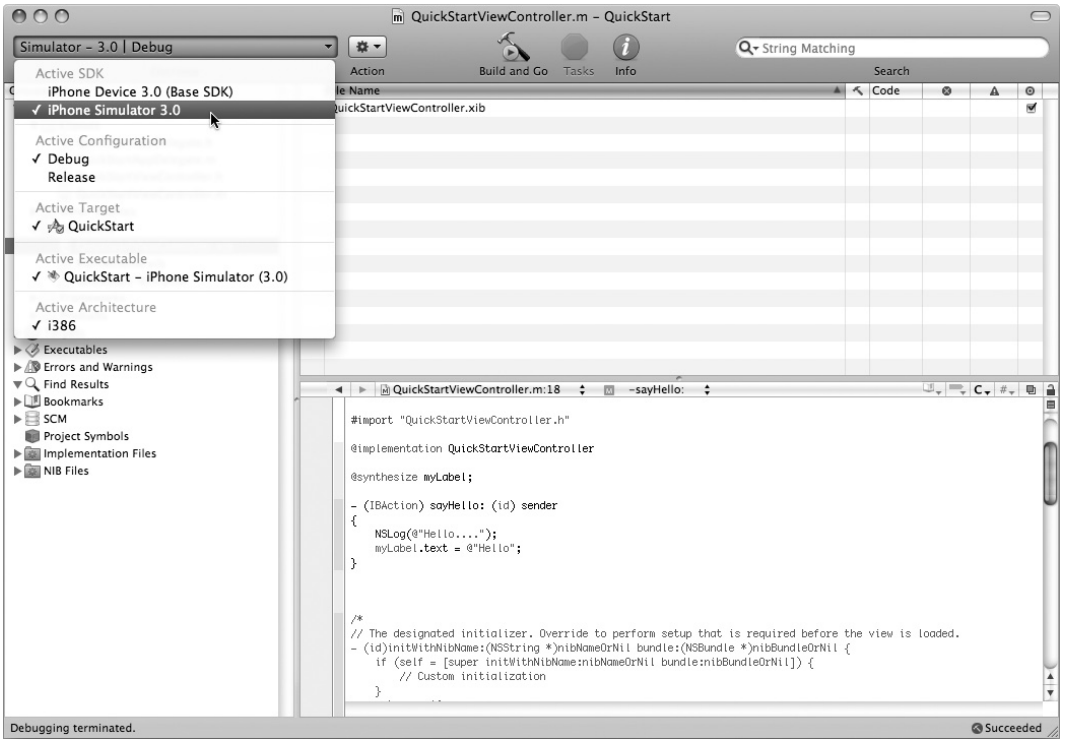

Figure 1-17 Ensuring Active SDK shows Debug and the Simulator selected

Figure 1-18 The application running in the iPhone Simulator

21. From Xcode's main menu, select Build | Build And Run. Xcode should start the simulator, install your application in it, and start your application (Figure 1-18).
22. Select Run | Console to show the Debugger Console.
23. Click the button, and the label's text changes to Hello and the console displays the log (Figure 1-19).

Listing 1-1 QuickStartViewController.m

```
#import "QuickStartViewController.h"
@implementation QuickStartViewController
@synthesize myLabel;
- (IBAction) sayHello: (id) sender {
 NSLog(@"Hello....");
```

(continued)

```
 self.myLabel.text = @"Hello";
}
- (void) dealloc {
 [super dealloc];
 [myLabel release];
}
@end
```

Listing 1-2 QuickStartViewController.h

```
#import <UIKit/UIKit.h>
@interface QuickStartViewController : UIViewController {
 IBOutlet UILabel * myLabel;
}
@property (nonatomic, retain) IBOutlet UILabel * myLabel;
- (IBAction) sayHello: (id) sender;
@end
```

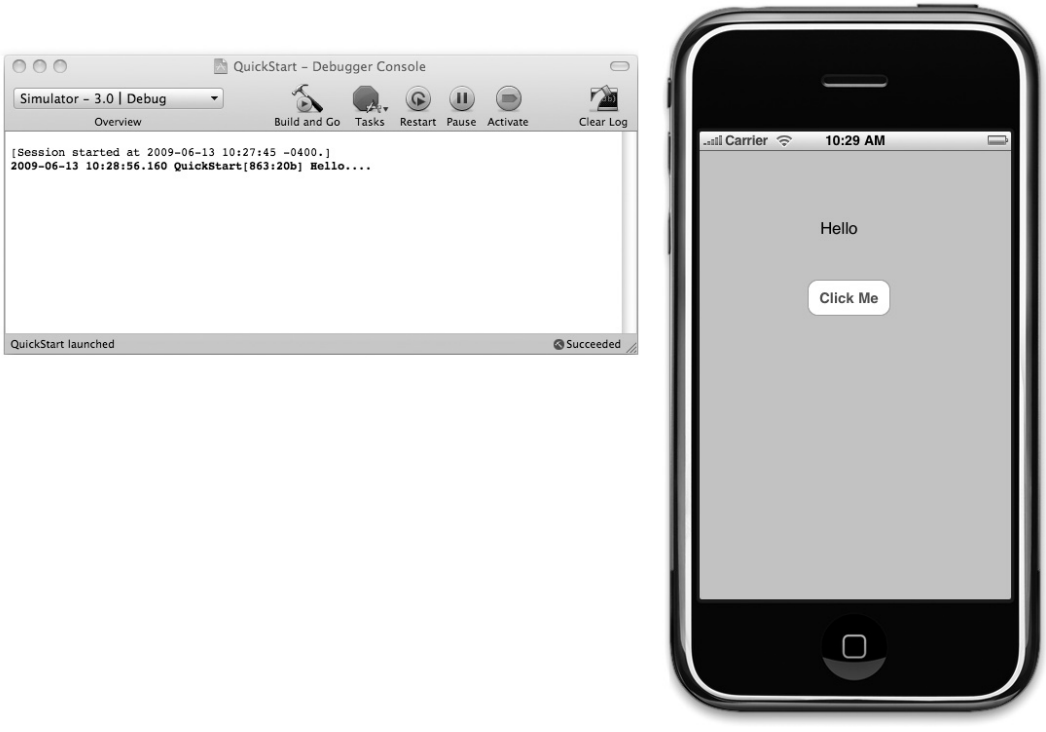

Figure 1-19 The application after clicking the button

You just did a lot of steps with no explanation. But what you did in this Try This will be second nature by this book's end. The biggest concept you must take from this simple example is using the IBAction and IBOutlet keywords.

IBAction and IBOutlet are covered several times in this book. IBActions are how you connect methods in classes in Xcode to events fired by components created using Interface Builder. IBOutlets are how you connect properties in classes in Xcode to graphical components created using Interface Builder.

These graphical components reside in a nib file, so a more correct explanation would be that IBActions and IBOutlets connect code to components in a nib file. For instance, you connected the button's Touch Up Inside event to the sayHello: action. The button lives in the nib, while the sayHello method lives in the compiled class. Making the sayHello method an IBAction connects the two. Like the button, the label also lives in the nib, while the myLabel property lives in the compiled class. Making the myLabel property an IBOutlet in the class file and then connecting the two in Interface Builder allows the class to manipulate the label via the myLabel property. Don't worry if this is still somewhat confusing—it won't be by the book's end. If you must know more now, Chapter 7 has a more "official" explanation of IBOutlets and IBActions.

Summary

This chapter introduced you to this book's content. Anyone with basic programming skills can write and release an application on Apple's App Store. Moreover, he or she can make money selling the application. Although the easy applications have all been released, there is room for high-quality applications on the App Store. All it takes is for Apple to feature your application on its Web site, and you are looking at a few thousand dollars for your efforts.

I love iPhone programming, and I find Objective-C a beautiful, elegant language. I am certain that by this book's end, you shall, too.