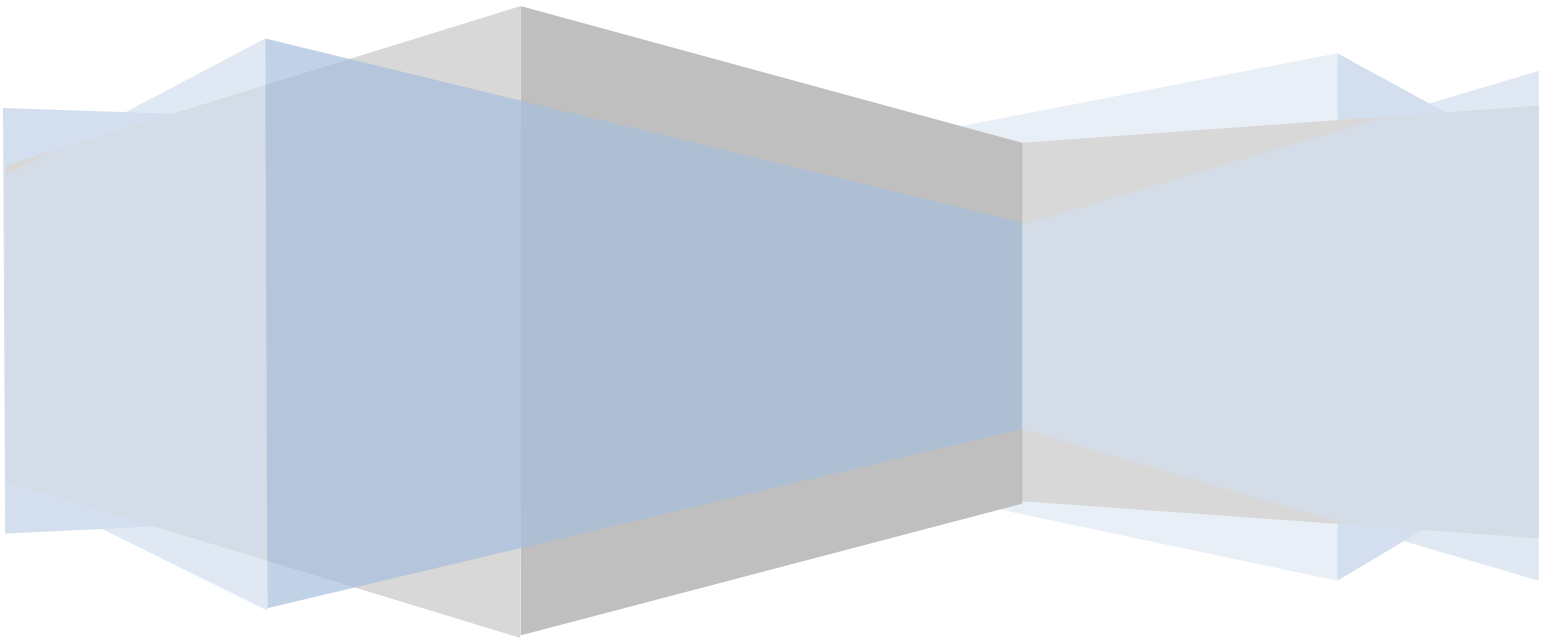
iPhone iOS 4 Development

Essentials

iPhone iOS 4 Development Essentials



iPhone iOS 4 Development Essentials – First Edition

ISBN-13: 978-0-9832282-2-6

Table of Contents

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 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. Both the iOS SDK and Xcode can be downloaded for free and assuming you 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.

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 oleophopic 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 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.

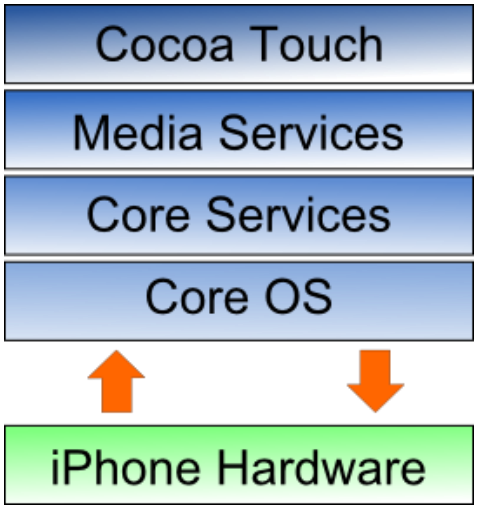
3.1 iPhone OS becomes iOS

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 with 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 and the iOS 4 SDK

iPhone apps are developed using the iOS SDK in conjunction with Apple's Xcode development environment. The iOS SDK contains the development frameworks that were outlined in iOS 4 Architecture and Frameworks. Xcode is an integrated development environment (IDE) within which you will code, compile, test and debug your iOS iPhone applications. The Xcode environment also includes a tool 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 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* 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.

4.2 Installing Xcode and the iOS 4 SDK

The best way to obtain the latest versions of Xcode 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 with the iOS SDK, you will need a *Registered Apple Developer* account. Fortunately membership is free and can be activated using your existing Apple account (for example the one you use to buy music on iTunes).

Once you have registered, you will gain access to the Apple iOS Developer Center web site where download links are provided for Xcode and the iOS SDK.

The download is over 3.5GB in size and will 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 15 - 30 minutes.

4.3 Starting Xcode

Having successfully installed the SDK and Xcode, the next step is to launch it so that we can write and then create a sample iPhone application. To start up Xcode, 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 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* 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 the iPhone.

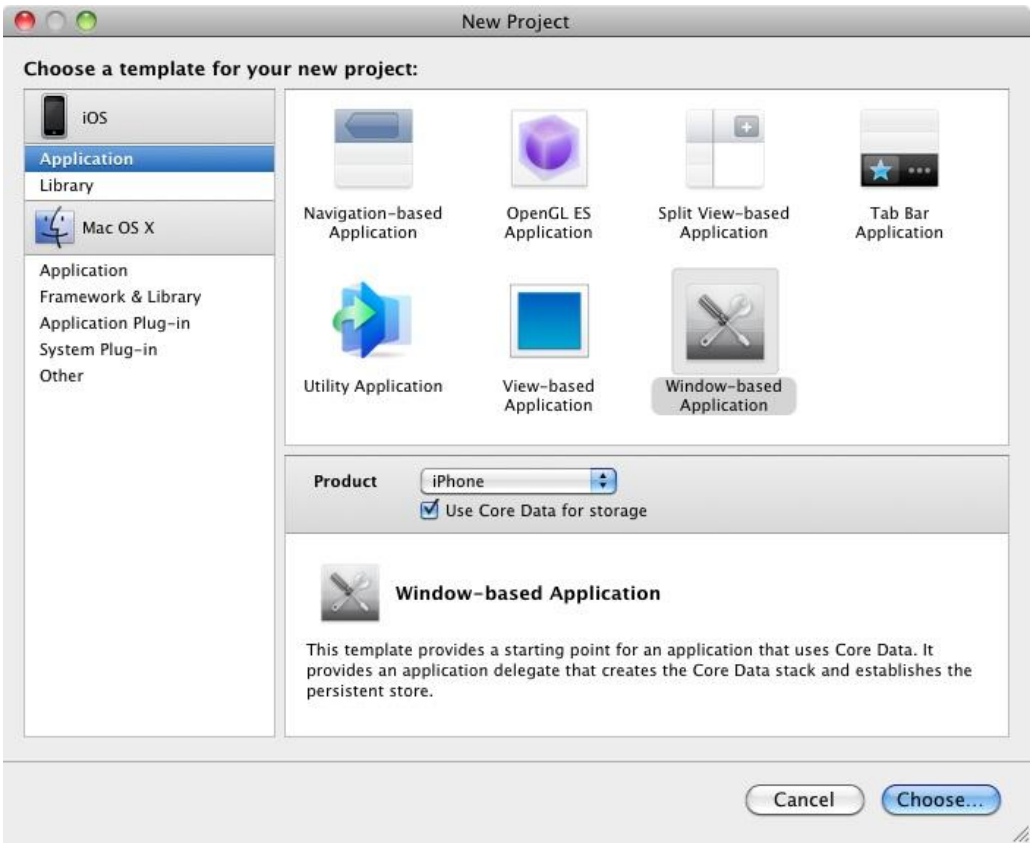
5.1 Starting Xcode

As with all iPhone iOS apps, the development of our example will take place within the Xcode development environment. If you have not already installed this tool together with the latest iOS SDK refer first to Installing Xcode 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 *Help -> 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 *New Project* window 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. In the center of the window when iOS selected is a *Product* menu where the target device needs to be selected. Options include iPhone, iPad or Universal (in other words an application for both the iPhone and the iPad). Note that certain templates are only available for specific devices. For example, the Navigation-based template is only available for iPhone apps and the Split View template is exclusive to iPad.

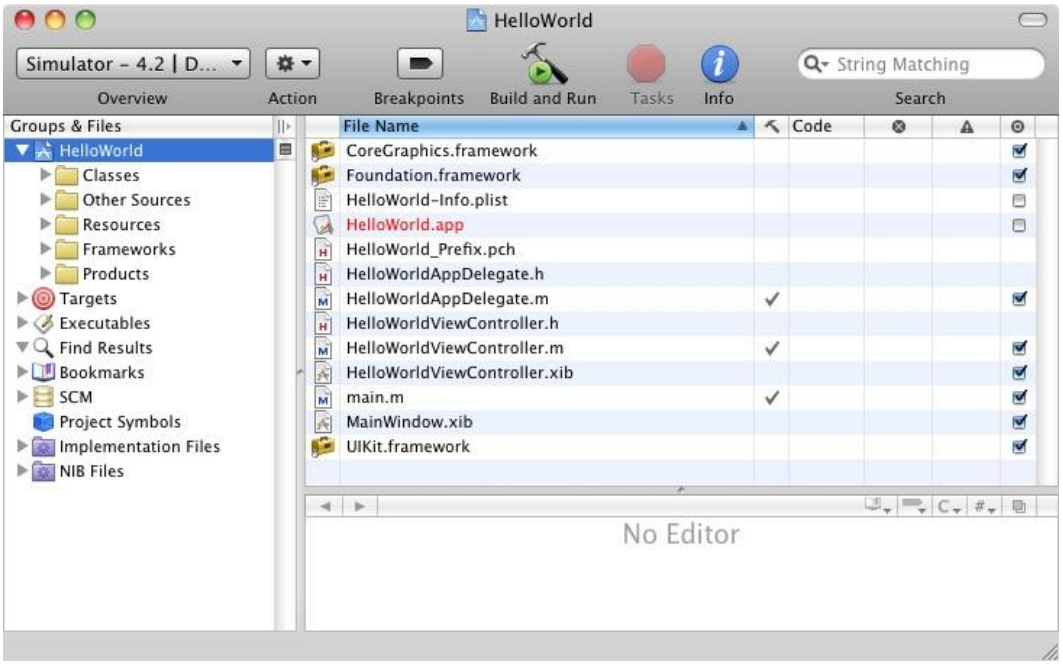
Begin by making sure that the *Application* option located beneath *iOS* is selected. The main panel contains a list of templates available to use as the basis for an application. The options available are as follows:

- **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. Select this option from the New Project window and make sure the *Product* menu selection is set to *iPhone*. Click on *Choose…* to drop down the file selection panel and enter the project name "HelloWorld" and select a suitable folder to hold to the project files (Xcode will default to your *Documents* folder unless you specify an alternative) and click *Save.*

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.

Also listed are some frameworks that Xcode believes we will need during the development of our app. Additional Frameworks can be added to the project by right clicking (or Ctrl-clicking) on the Frameworks folder located in the left hand *Groups & Files* panel and selecting *Add -> Existing Frameworks…* from the menu. A list of available frameworks will be displayed from which you may select the framework you need.

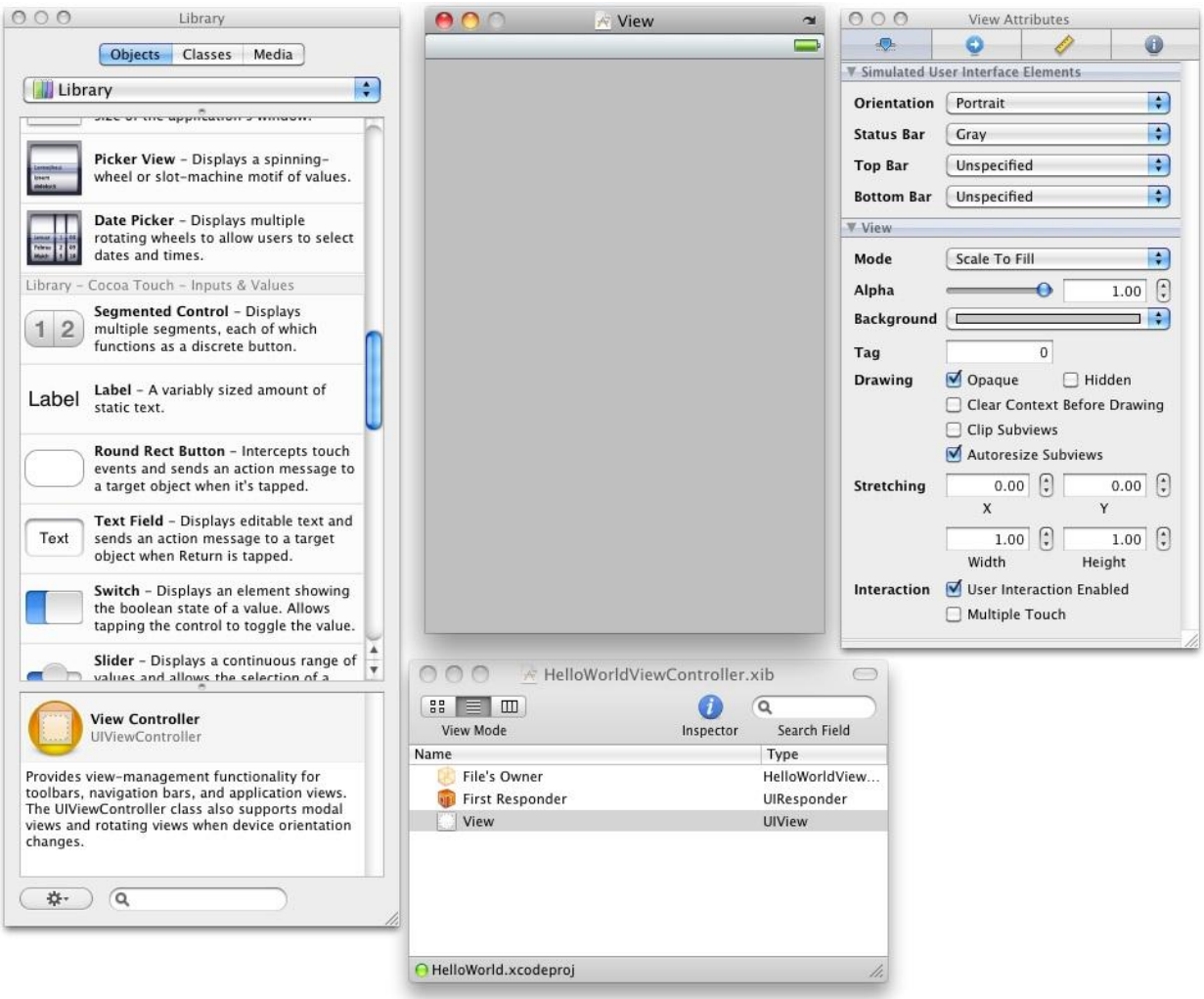
When a source file is selected from the list in the main panel, the contents of that file will appear in the panel below where it may then be edited. To open the file in a separate editing window, simply double click on the file in the list.

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 double click on the file name in the list. Interface Builder will subsequently start up and display a number of windows as shown in the following figure:



The smaller dialog entitled *HelloWorldViewController.xib* contains the key objects that make up the view controller for our user interface. We will be looking at the other objects in the window in later chapters. For now, the item of most interest to us is the *View.* This corresponds to the view canvas onto which our user interface is going to be constructed and is represented by the gray dialog entitled *View.* If the view dialog is not visible at any time simply double click on the View icon to display it.

The gray dialog entitled *View* provides a representation of the *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. The objects available to us as we develop our app are located in the *Library* window. The objects are categorized into groups and additional information about each object can be obtained by selecting the item and reading the corresponding description.

Finally, the *Attribute Inspector* dialog allows the properties of the currently selected user interface component to be viewed and configured. The attributes available will change depending on the type of component selected in the View dialog. Presently, the attributes for the *UIView* object will be displayed since this the only component in our interface design at this point. If the Attribute Inspector dialog is not visible it may be displayed by selecting the *Tools -> Attribute Inspector* menu option or by pressing Command + 1. The various settings in the inspector are divided into categories (attributes, connections, size and identity) and grouped on different screens. Each screen may be accessed by clicking on the corresponding toolbar button at the top of the Inspector window. Inspector windows with the required category pre-selected may also be opened by making appropriate selections from the Interface Builder *Tools* menu.

5.3 Changing Component Properties

With the property panel for the View selected we will begin our design work by changing the background color of the view. Begin by making sure the View is selected and that the Attribute Inspector window is visible. 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 scroll down the list of objects in the Library window until you reach the *Inputs and Values* section. 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 appears. With the Label still selected, click on the centered alignment button in the *Layout* attribute section of the Attribute Inspector window 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 36.

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* and then exit via the *Interface Builder -> Quit Interface Builder* menu option.

5.5 Building and Running an iOS App in Xcode

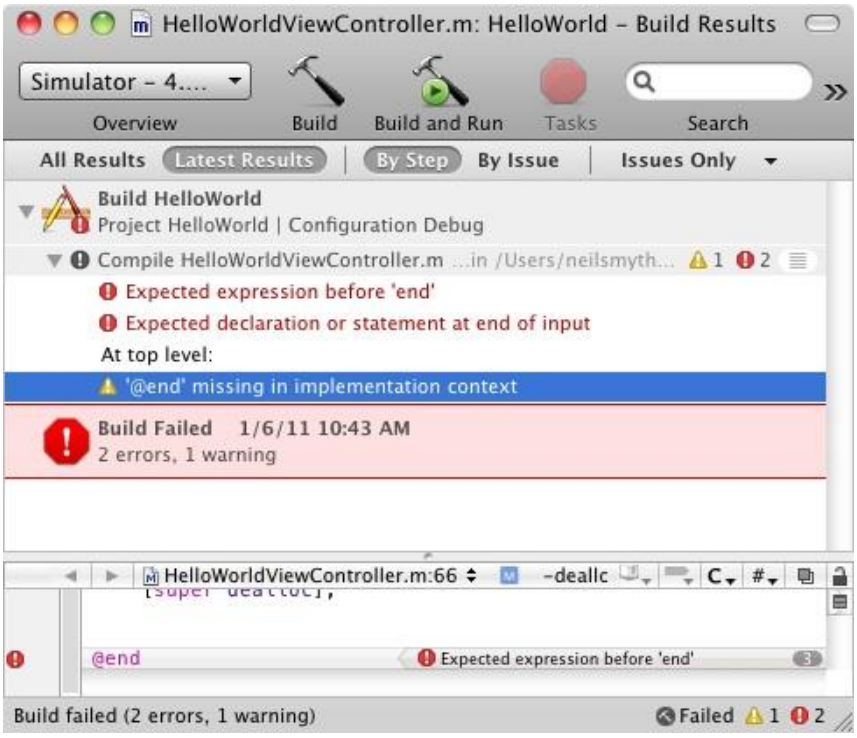
Before an app can be run it must first be compiled. Once successfully compiled it may be run either within a simulator or on a physical iPhone, iPad or iPod Touch device. The process for testing an app on a physical device requires some addition steps to be performed involving developer certificates 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 project window make sure that the menu located in the top left hand corner of the window has the *Simulator* option selected and then click on the *Build and Run* toolbar button to compile the code and run the app in the simulator. The gray status bar at the bottom of the Xcode window 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 bar at the bottom of the screen will report that an error has been detected by displaying "Build failed" together with the number of errors detected and on the right hand side a "Failed" link. When this link is clicked, Xcode will display the *Build Results* window providing detailed information on the nature of the problem or problems detected:



If the problem lies in a line of source code (as is the case in the above screenshot), double clicking on the line reference in the results window will open the editor and take you directly to the offending line so that the appropriate corrective action may 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 iPhone simulator 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.

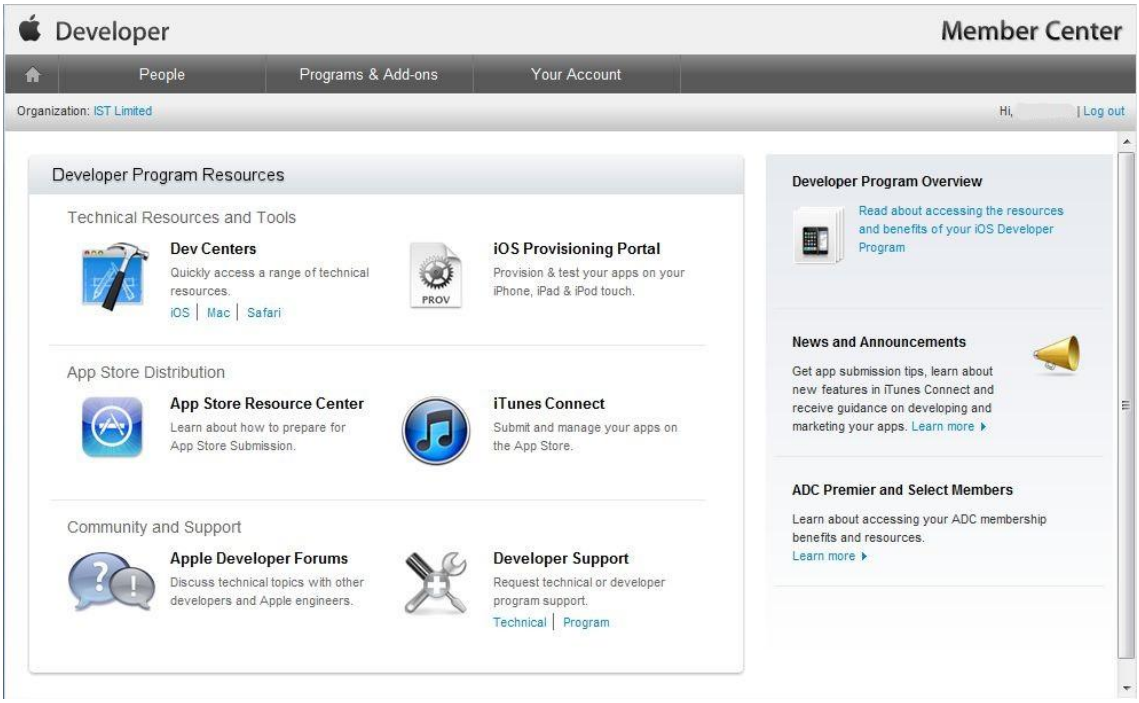
6.1 Joining the iOS Developer Program

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/

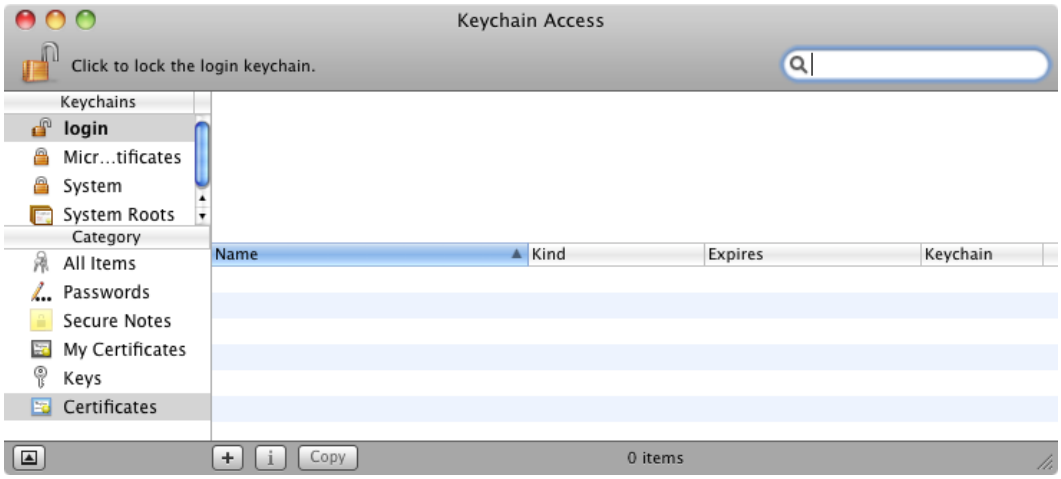
Having completed the enrollment process, navigate to http://developer.apple.com and click on the *Member Center* link located near the top right hand corner of the screen. On the resulting page enter the Apple ID and password associated with your iOS Developer Program membership to access the member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