iOS 8 App Development

Essentials

iOS 8 App Development Essentials

iOS 8 App Development Essentials – Second Edition

Rev 2.0

Table of Contents

Chapter 1

1. Start Here

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

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

1.1 For New iOS Developers

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

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

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

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

1.2 For iOS 7 Developers

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

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

Start Here

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

In addition the following changes have also been made:

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

1.3 **Source Code Download**

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

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

1.4 **Feedback**

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

1.5 **Errata**

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

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

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

Chapter 2

2. Joining the Apple iOS Developer Program

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

2.1 Registered Apple Developer

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

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

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

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

2.2 Downloading Xcode 6 and the iOS 8 SDK

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

2.3 iOS Developer Program

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

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

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

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

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

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

2.4 When to Enroll in the iOS Developer Program?

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

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

2.5 Enrolling in the iOS Developer Program

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

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

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

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

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

http://developer.apple.com/membercenter

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

Figure 2-1

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

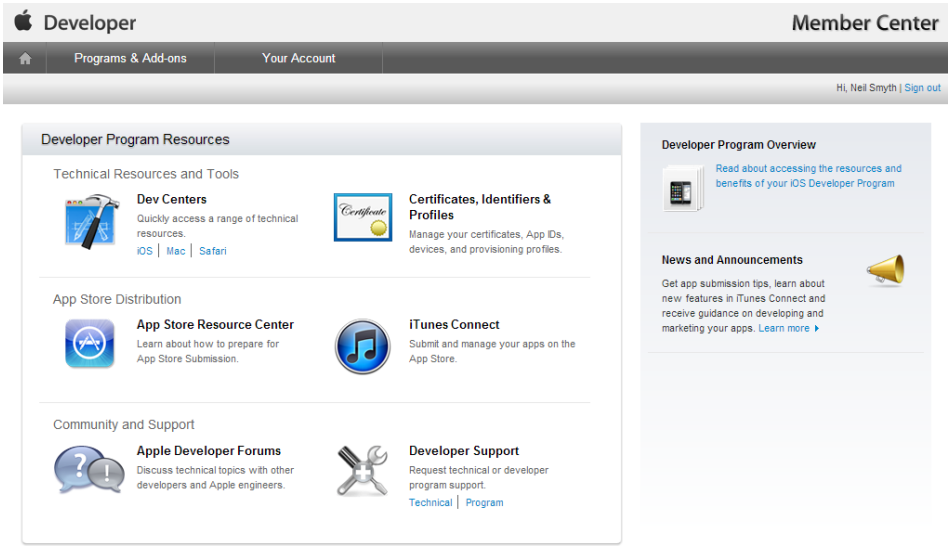

Figure 2-2

2.6 Summary

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

Chapter 3

3. Installing Xcode 6 and the iOS 8 SDK

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

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

3.1 Identifying if you have an Intel or PowerPC based Mac

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

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

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

Figure 3-1

3.2 Installing Xcode 6 and the iOS 8 SDK

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

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

3.3 Starting Xcode

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

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

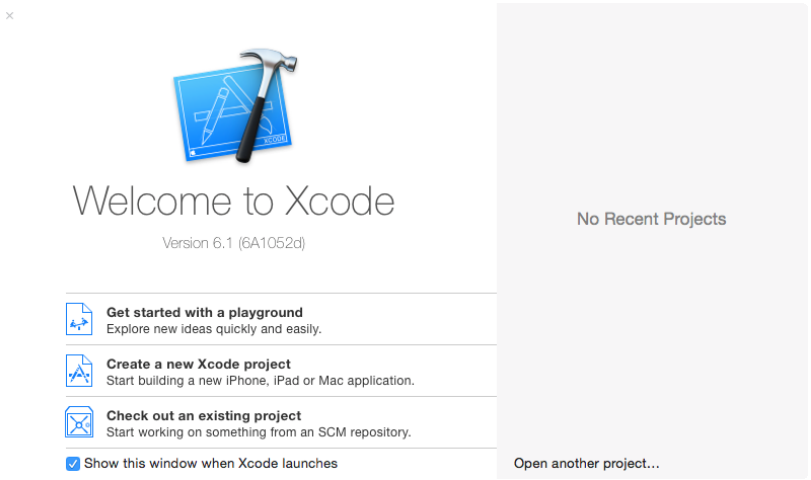

Figure 3-2

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

Chapter 4

4. A Guided Tour of Xcode 6

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

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

4.1 Starting Xcode 6

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

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

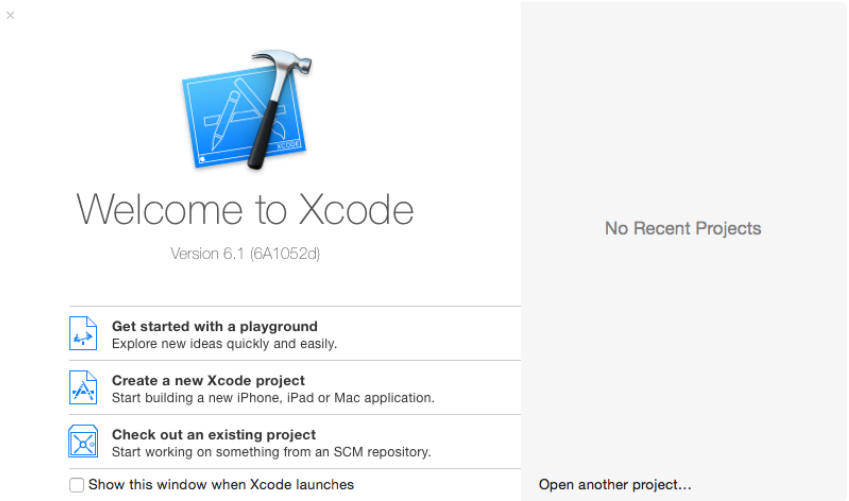

Figure 4-1

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

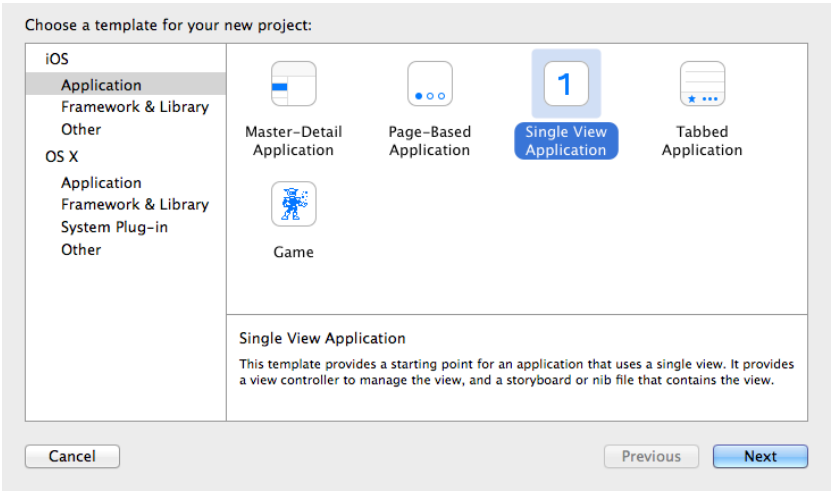

Figure 4-2

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

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

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

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

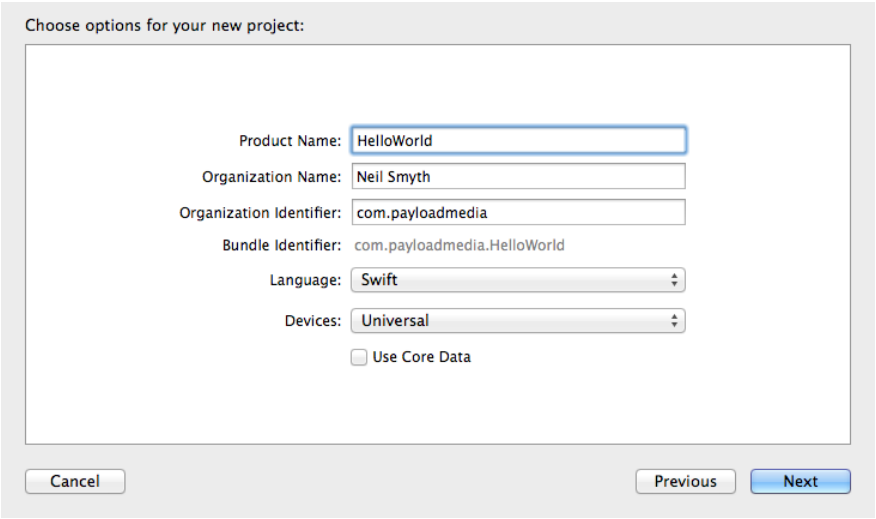

Figure 4-3

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

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

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

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

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

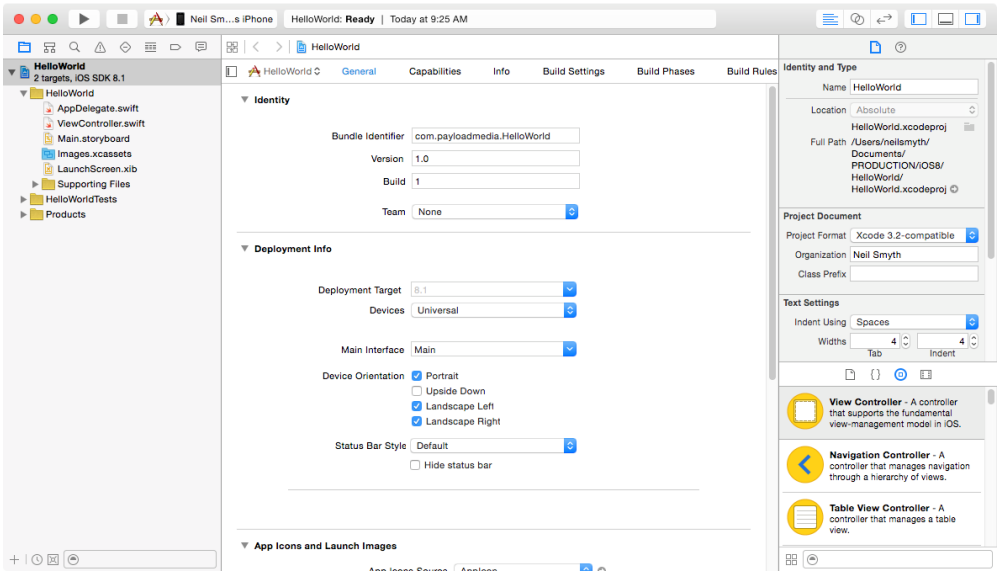

Figure 4-4

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

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

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

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

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

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

4.2 Creating the iOS App User Interface

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

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

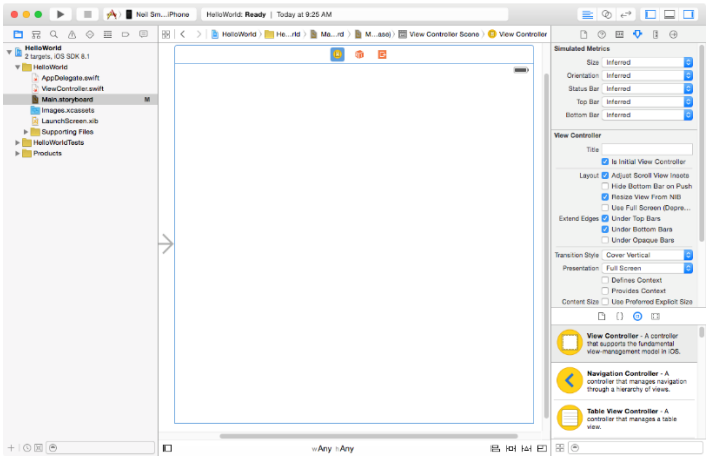

Figure 4-5

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

Figure 4-6

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

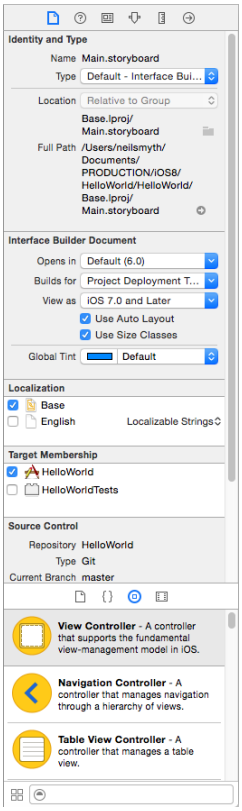

Figure 4-7

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

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

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

4.3 Changing Component Properties

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

4.4 Adding Objects to the User Interface

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

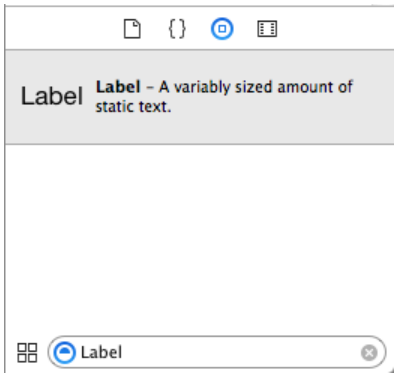

Figure 4-8

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

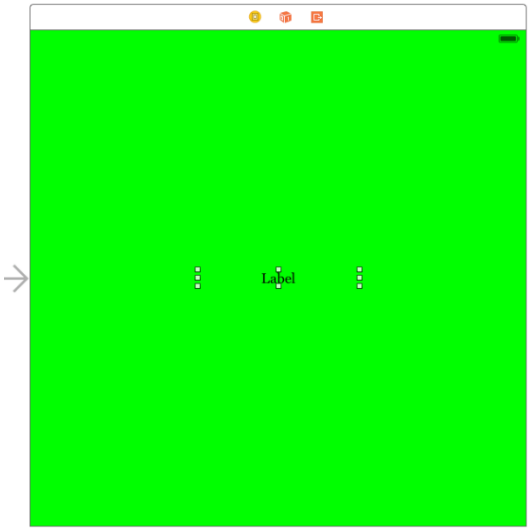

Figure 4-9

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

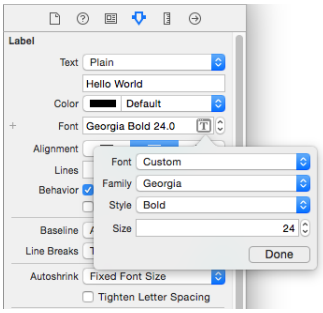

Figure 4-10

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

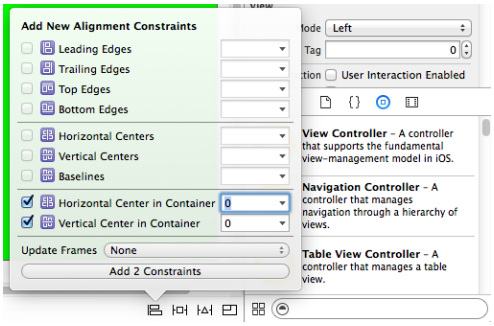

Figure 4-11

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

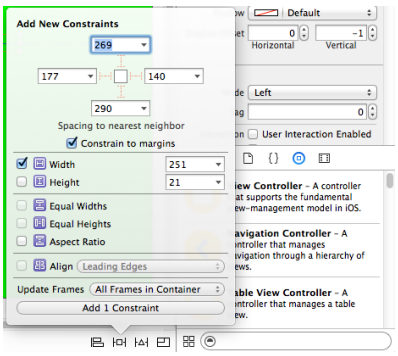

Figure 4-12

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

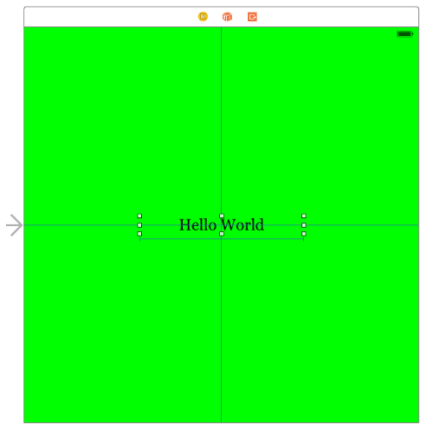

Figure 4-13

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

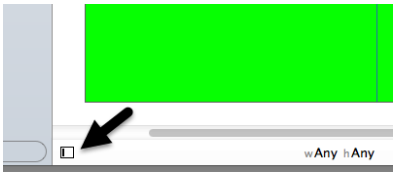

Figure 4-14

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

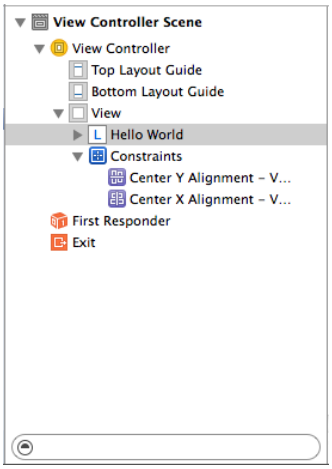

Figure 4-15

4.5 Building and Running an iOS 8 App in Xcode 6

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

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

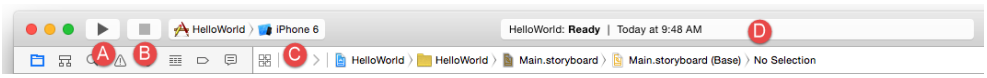

Figure 4-16

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

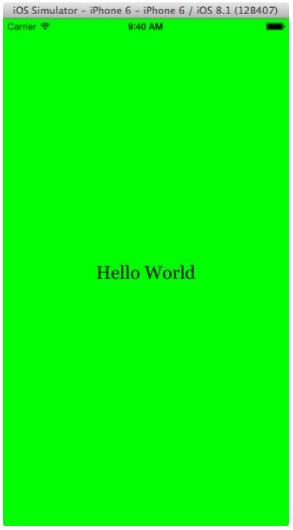

Figure 4-17

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

4.6 Dealing with Build Errors

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

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

4.7 Monitoring Application Performance

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

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

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

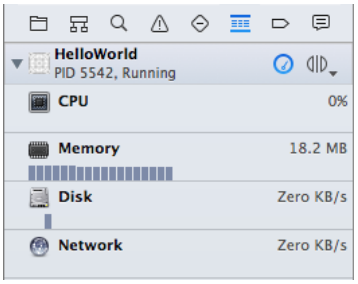

Figure 4-18

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

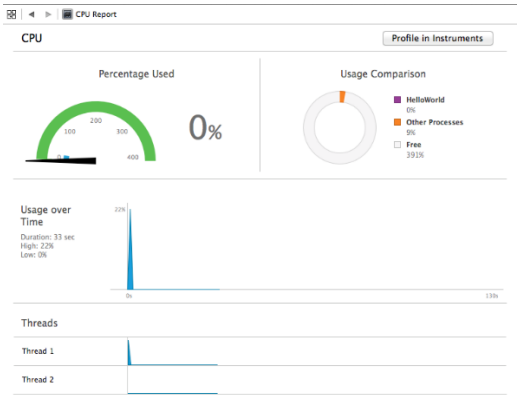

Figure 4-19

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

4.8 An Exploded View of the User Interface Layout Hierarchy

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

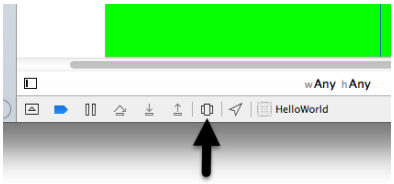

Figure 4-20

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

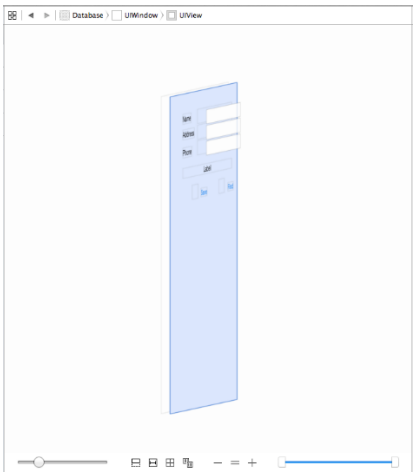

Figure 4-21

4.9 Summary

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

Chapter 5

5. Testing Apps on iOS 8 Devices with Xcode 6

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

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

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

5.1 Configuring Xcode with Apple IDs

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

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

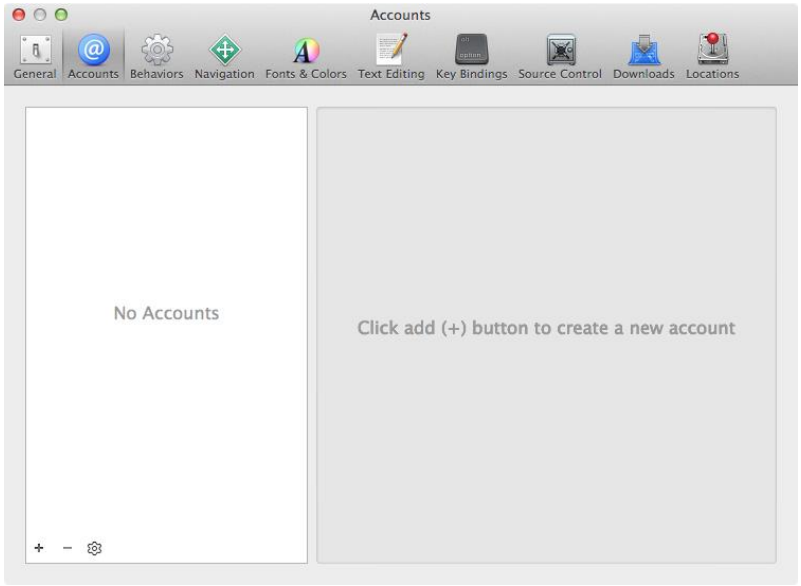

Figure 5-1

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

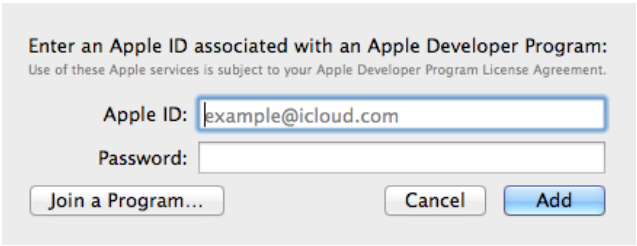

Figure 5-2

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

5.2 Generating Signing Identities

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

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

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

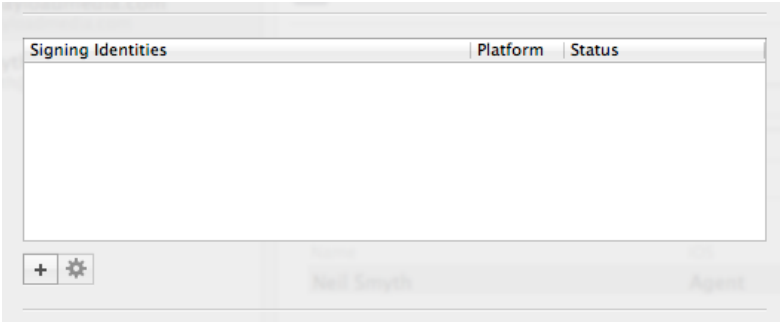

Figure 5-3

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

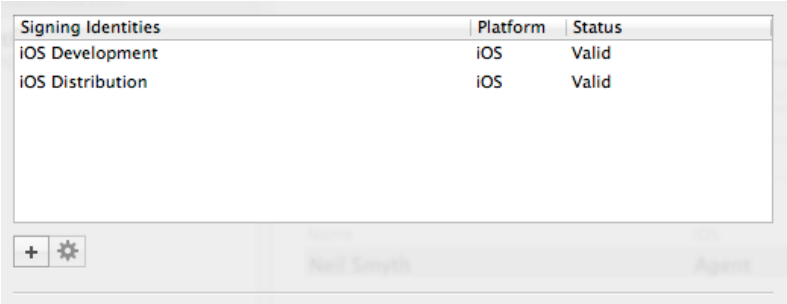

Figure 5-4

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

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

https://developer.apple.com/membercenter

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

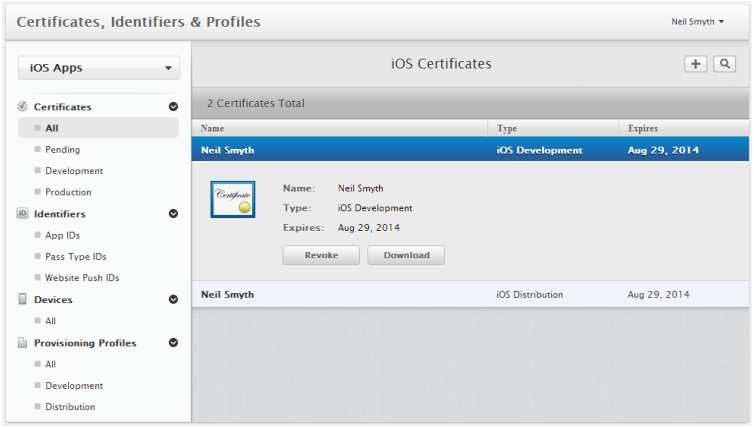

Figure 5-5

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

5.3 Device Registration

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

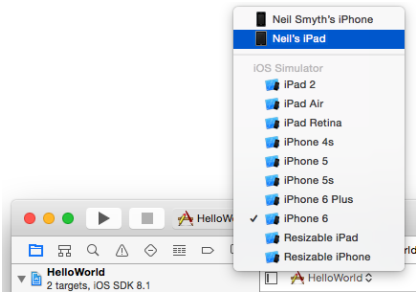

Figure 5-6

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

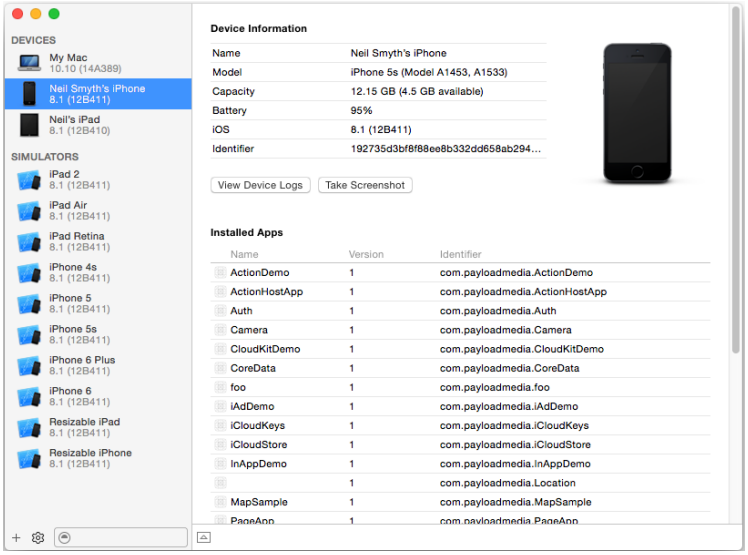

Figure 5-7

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

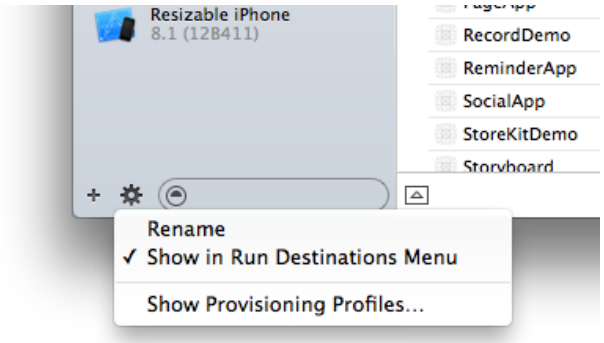

Figure 5-8

5.4 Manually Adding Test Devices

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

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

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

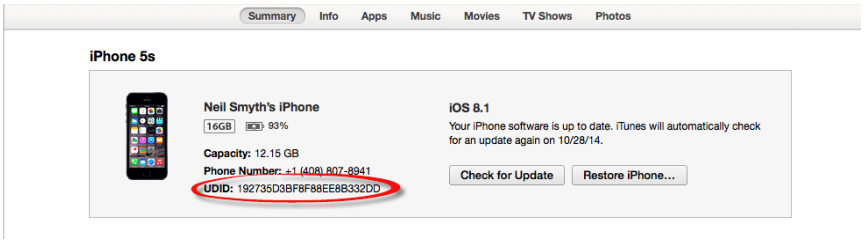

Figure 5-9

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

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

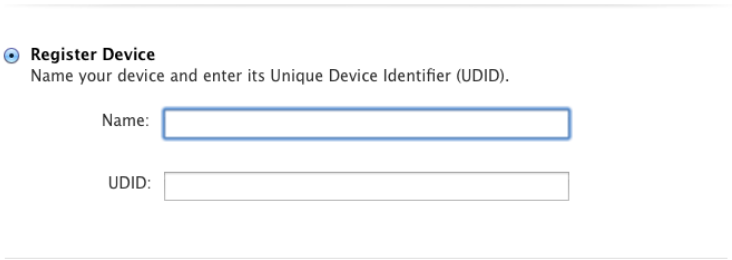

Figure 5-10

5.5 Running an Application on a Registered Device

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

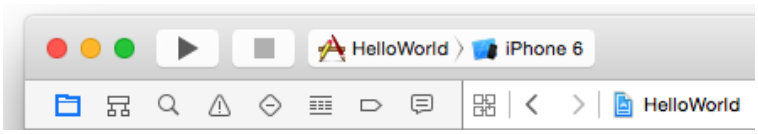

Figure 5-11

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

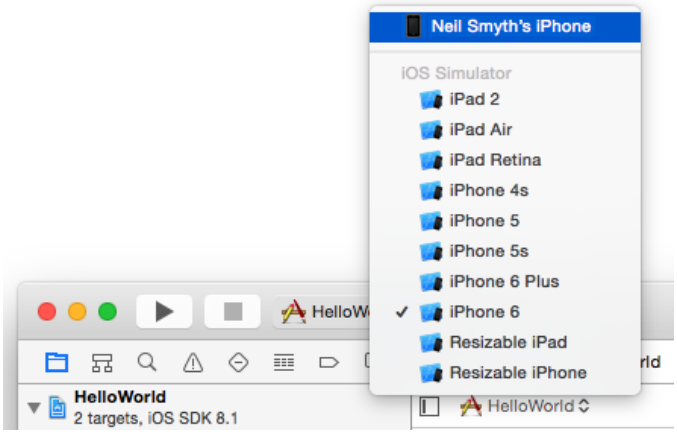

Figure 5-12

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

5.6 Summary

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

Chapter 6

6. An Introduction to Swift Playgrounds

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

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

6.1 What is a Swift Playground?

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

6.2 Creating a New Swift Playground

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

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

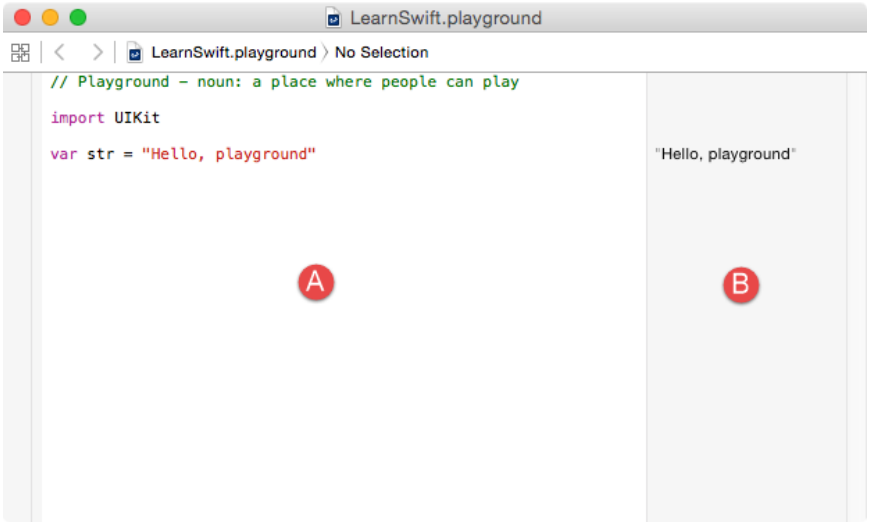

Figure 6-1

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

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

6.3 A Basic Swift Playground Example

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

```
var str = “Hello, playground”
```

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

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

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

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