Session 3

Introduction to Xcode

MODULE OBJECTIVES

At the end of this module, you will be able to:

- Create a new app in Xcode
- Work on layouts and editors in Xcode
- Run your app on a simulator or on a device

SESSION OBJECTIVES

At the end of this session, you will be able to:

- Identify the purpose of Xcode templates and bundle identifiers
- Identify the Xcode's layouts and editors
- Use Interface Builder to edit storyboards
- Run your app on a simulator
- Work on auto-layout and editors
- Work on various application settings
- Run your application on a device

INTRODUCTION

Now that you know the basics of Objective-C and have created a small application in Objective-C, it is time to start the development of your app. iOS application development relies on a single unified development tool called **Xcode**. Xcode provides everything developers need to create great apps for Mac, iPad, and iPhone. It includes the **Xcode Integrated Development Environment** (IDE), **Swift**, and **Objective-C compilers**. Swift is a new innovative programming language for Cocoa and Cocoa Touch. It is a free tool for the new and curious developers. Experienced and dedicated developers, however, can get advanced information about updates to iOS by paying a small annual fee to enroll in Apple's **iOS developer program**.

Xcode is a powerful package with many features. You do not need to master it all to get started, but you do need to have a basic understanding of its most obvious features. In addition to getting started with Xcode, you also need to start learning about iOS 8 and technical details of the app design.

In this session, you will start development of your new app, **Bands**, in Xcode.

INTRODUCTION TO Xcode

This section introduces Xcode, the IDE used to actually create both iOS and Mac OS X desktop applications. Xcode is similar to **Microsoft Visual Studio** or **Eclipse**. You start by creating a project, and then you edit the code and the user interface files within Xcode. However, it was not always this way. Just a few years ago, Xcode was strictly for code editing while you worked on your user interface files in Interface Builder. Today, Interface Builder is integrated within Xcode to make it more familiar to developers coming from other platforms.

You can create any app according to your idea using Xcode IDE. Xcode provides you the user interface for all types of screen sizes and resolutions.

Creating a New App in Xcode

The first step to any iOS application development is to create a new project in Xcode. This project will hold all the code files, art assets, settings, and configuration files used to compile and distribute the application.

To start developing the **Bands** app, you can create a new iOS project in Xcode:

1. Open Xcode and select the **File** ➪ **New** ➪ **Project** menu, as shown in **Figure 1**. This starts a wizard to create a new project.

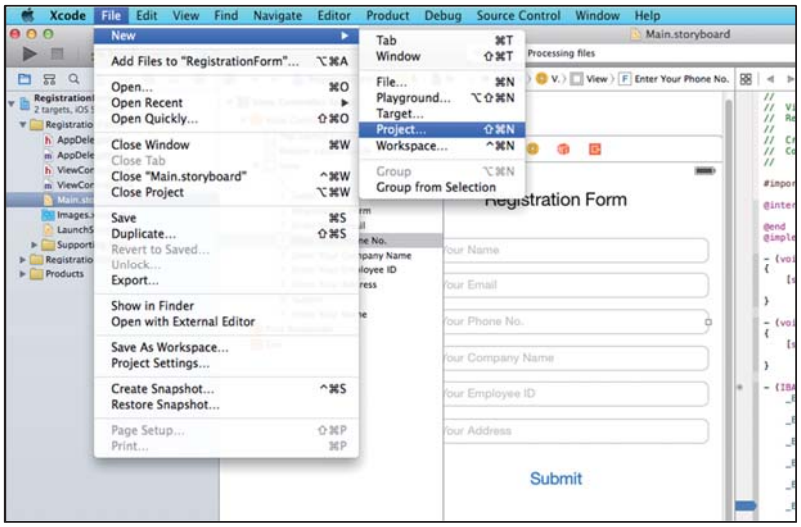

Figure 1: Select the File ➪ New ➪ Project menu in Xcode

2. On the first screen of the wizard (see **Figure 2**), select **Application** under **iOS** in the left panel. Then, select the **Single View Application** option in the main panel. The **Single View Application** template provides the code files and user interface files needed for an application with only one view. After selecting this template, click **Next**.

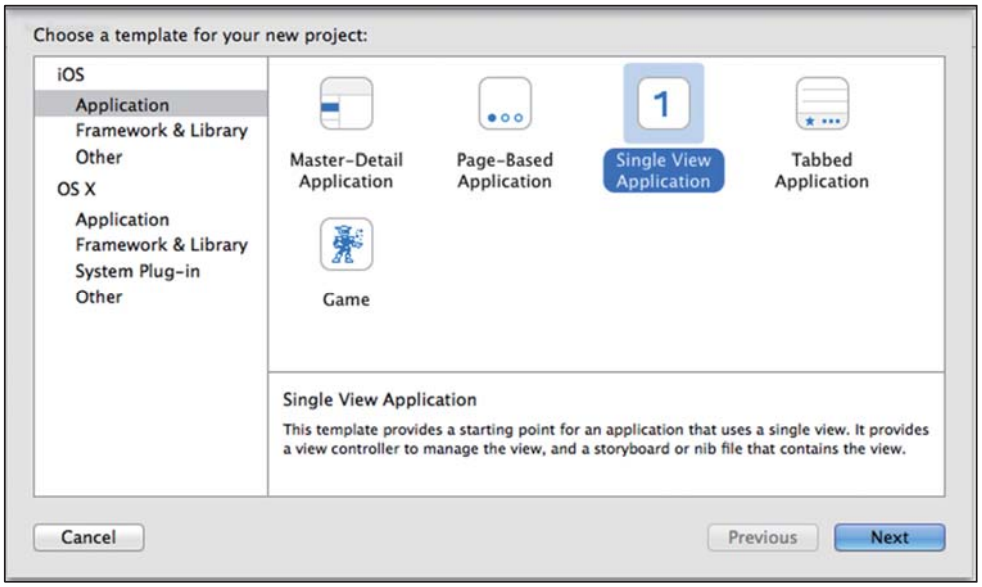

Figure 2: Choose the Single View Application Template

3. Choose options for your project on the next screen (see **Figure 3**). The name of a project is often the name of the application you are building, though it does not have to be. You can change the name of the application in the configuration file. Therefore, type **Bands** in the **Product Name** box. Select **Objective-C** from the **Language** drop-down list, select **iPhone** from the **Devices** list, and then click **Next**.

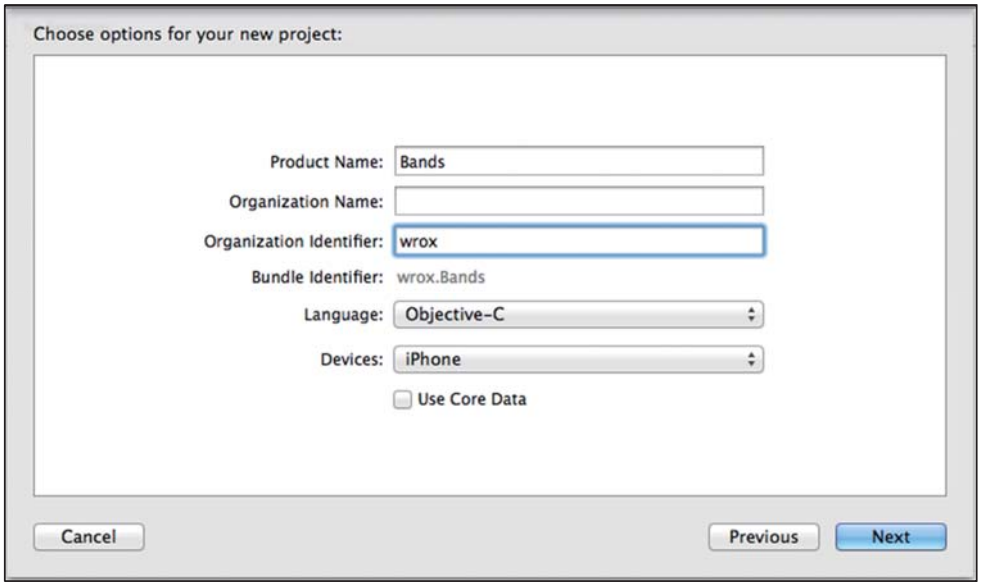

Figure 3: Choose Options for the New Project

4. On the next screen (see **Figure 4**), choose the location to save your project to a disk and click **Create.**

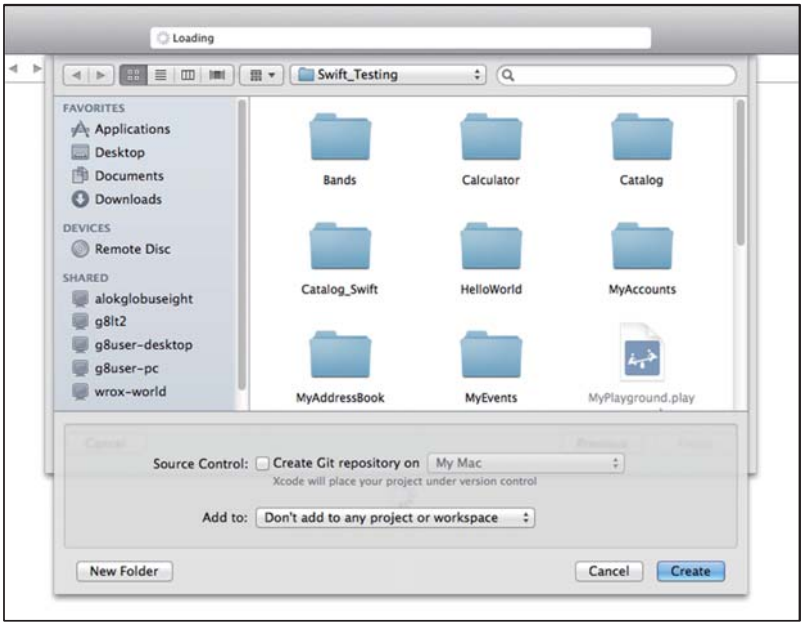

Figure 4: Save the Project and Click Create

5. The **Bands** project is created, as shown in **Figure 5**. This project is contained in a single directory with some subdirectories. The code files for the project are created in a subdirectory with the same name as the project, which is the **Bands** directory. Xcode projects are also created with default unit test files that are created in a directory with the project name followed by **Tests**, which is the **Bands Tests Directory**.

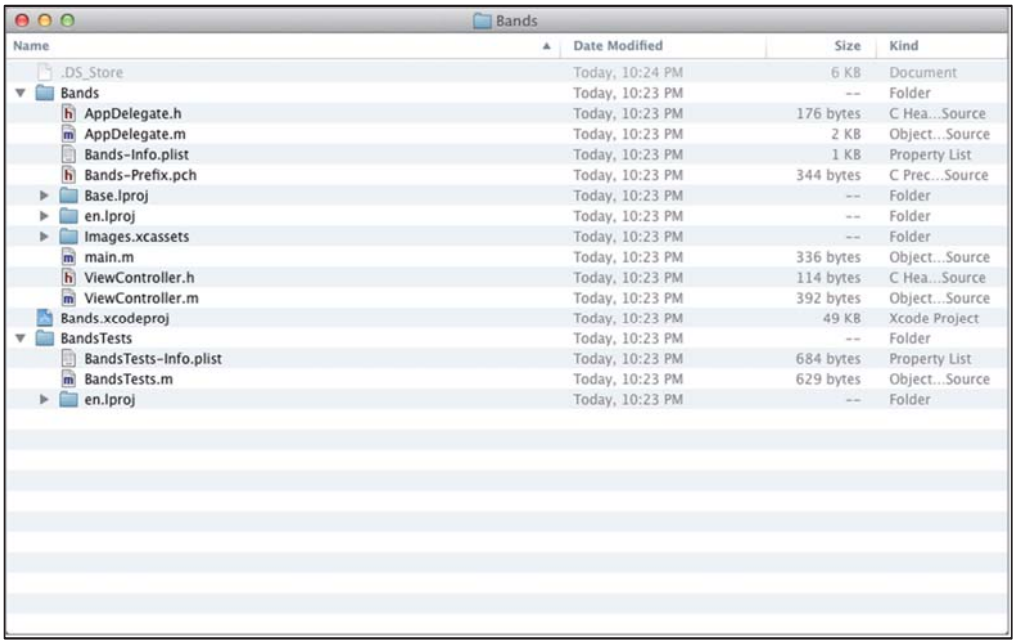

Figure 5: The Bands Project is Created

Discussing Xcode Templates

Xcode versions also offer a variety of project templates with which you can start building your app. In the previous section, you created the **Bands** project using the **Single View Application** template. The Calculator app is an example of a single view application.

Table 1 describes the various templates in Xcode.

Table 1: Xcode Templates

TEMPLATE NAME	DESCRIPTION	EXAMPLE APP
Master-Details Application	An application that typically uses a table view to list objects and a navigation controller to transition to a details view of the object.	Contacts
Page-Based Application	An application that contains different views and allows the user to transition between them by swiping to the left or right. These apps have a series of dots along the bottom so the user knows how many views there are and which one they are on.	Compass
Tabbed Application	An application that has a tab bar along the bottom that is used to switch between the different views.	Music
Utility Application	An application with a main view and a secondary view with an info button to switch between the two.	iOS 6 Weather
OpenGL Game	A game application that uses OpenGL for drawing.	Infinity Blade
SpriteKit Game	A game application that uses SpriteKit for drawing.	Disco Bees
Empty	A project that contains only a window and an application delegate file.	N/A

When you first create a project, it is important to think about what type of template makes sense for your application because it gives you a head start. However, that does not mean that you are stuck with that application architecture. Though you started the **Bands** app with the **Single View Application** template, you will modify it to be more of a **Master-Details Application** as you add new features. You can also modify the default templates or even add your own. If you are creating new projects often, you may want the default template to add files with your file naming conventions. This is an advanced topic and not recommended for beginners, but it is nice to know the option is there as you become an expert yourself.

Learning about Bundle Identifiers

The **Bundle Identifier** is a unique identifier for your application. This identifier is used throughout the Apple system, so you need to know it.

The Bundle Identifier is typically reverse-domain style with the company identifier you entered followed by your product name. An example of this using **Wrox** and **Bands** would be **wrox.Bands**. Though Xcode does not enable you to set this in the new project steps, you can edit it after you create the project.

EXPLORING THE Xcode PROJECT LAYOUT

After you have created your project, you can see the Xcode Workspace window. The layout is similar to other IDEs. **Figure 6** shows the Xcode IDE.

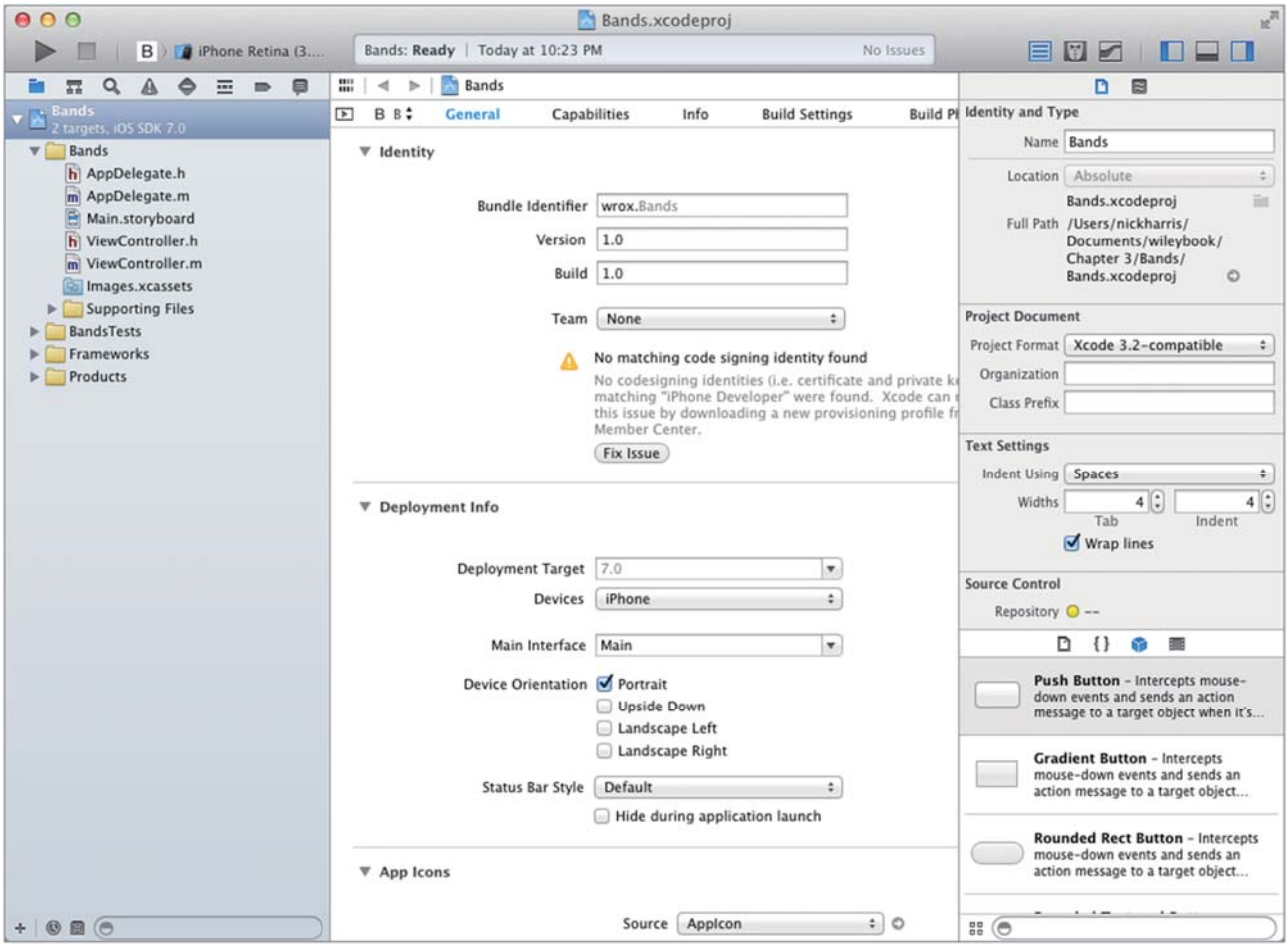

Figure 6: Xcode IDE

Now, let's discuss the various features of the Xcode IDE:

- **Navigation Pane:** This pane is on the left side of the IDE. The default view for this pane is the Project Navigator, which shows your project and its files, as well as any groups or folders you create. Yellow folders represent **groups**. They are used to group files together within the project, but they do not correspond to folders on disk. You can add folders, which map to folders on disk, which are shown as blue. Typically, groups are used instead of folders, but that is more of a developer preference. This pane also enables you to navigate your project using symbols as well as searching all files within the project. It also shows you all your breakpoints.
- **Editor:** This is the central pane of the IDE. Depending on the type of file selected, you can see different editors:
  - **Settings Editor:** This is displayed by selecting a project in the Navigation pane.
  - **Text Editor:** This is displayed by selecting a file in the Navigation pane.
  - **Interface Builder:** This is displayed by selecting the user interface files. The left side of Interface Builder shows all the user interface objects and their hierarchy. Selecting objects in the hierarchy selects them in the scene shown in the main portion of the editor.
- **Utility Pane:** The right pane is the Utility pane. Here, you see additional information that supplements the editor pane. You can see how this pane is used as you continue this session.

- **Debug Area:** The last pane is the Debug area. It is typically hidden while you use the editor and is shown while you debug the application. It contains your console as well as buttons to step through code along with variable information.

From the Utility pane, there are different utility tools available such as button, slider, and text, by which you can create different applications according to their usage requirement.

Discussing the `UIKit` Framework

Before you start building a user interface (UI), it is important to understand the components and frameworks you will use. All iOS applications are built using the `UIKit` framework that is part of Cocoa Touch. Apple uses a naming convention to help you know the framework used by a class or protocol. The convention is to prepend the name of the class with the frameworks abbreviation. With the `UIKit` framework, all classes and protocols start with UI.

The application itself is represented by the `UIApplication` object and its companion protocol `UIApplicationDelegate`. Every project you create using one of Xcode's templates will include a class called `AppDelegate` that implements the `UIApplicationDelegate` protocol. You use the methods of this protocol to determine the important events in the life cycle of the application. This includes when the application launches, becomes active, or is about to be terminated. The **Bands** app does not do anything with these events, but it is important to understand why the file is included when the project is created.

All of the user interface objects are also part of `UIKit`. You use them to build your application in a way that is visually familiar to other applications so that the user knows how to interact with your application. With that knowledge, it is time to start working on the user interface of the **Bands** app.

Discussing the Main Storyboard

`Main.storyboard` is the user interface file for your application. Storyboards were introduced to Xcode with the iOS 5 SDK. Storyboards enable you to build and view the entire flow of the application. The two main components of the storyboard are:

- **Scenes:** They are views in your application's user interface. Typically, they have their own `UIViewController` subclass in your project.
- **Segues:** They represent how your app navigates from view to view.

Adding a Label to a Storyboard

The most basic user interface object in any language is a text label. In iOS, it is called a `UILabel`. The steps to add a `UILabel` to a scene are as follows:

1. Select the `Main.storyboard` file from the Navigation pane.
2. At the bottom of the Utility pane, select the Objects tab represented by the cube icon.
3. In the search box at the bottom of the screen, type **Label** to filter the objects in the list.
4. Drag the **Label** onto your scene in Interface Builder, as shown in **Figure 7**. This will add the `UILabel` object to the base `UIView`.

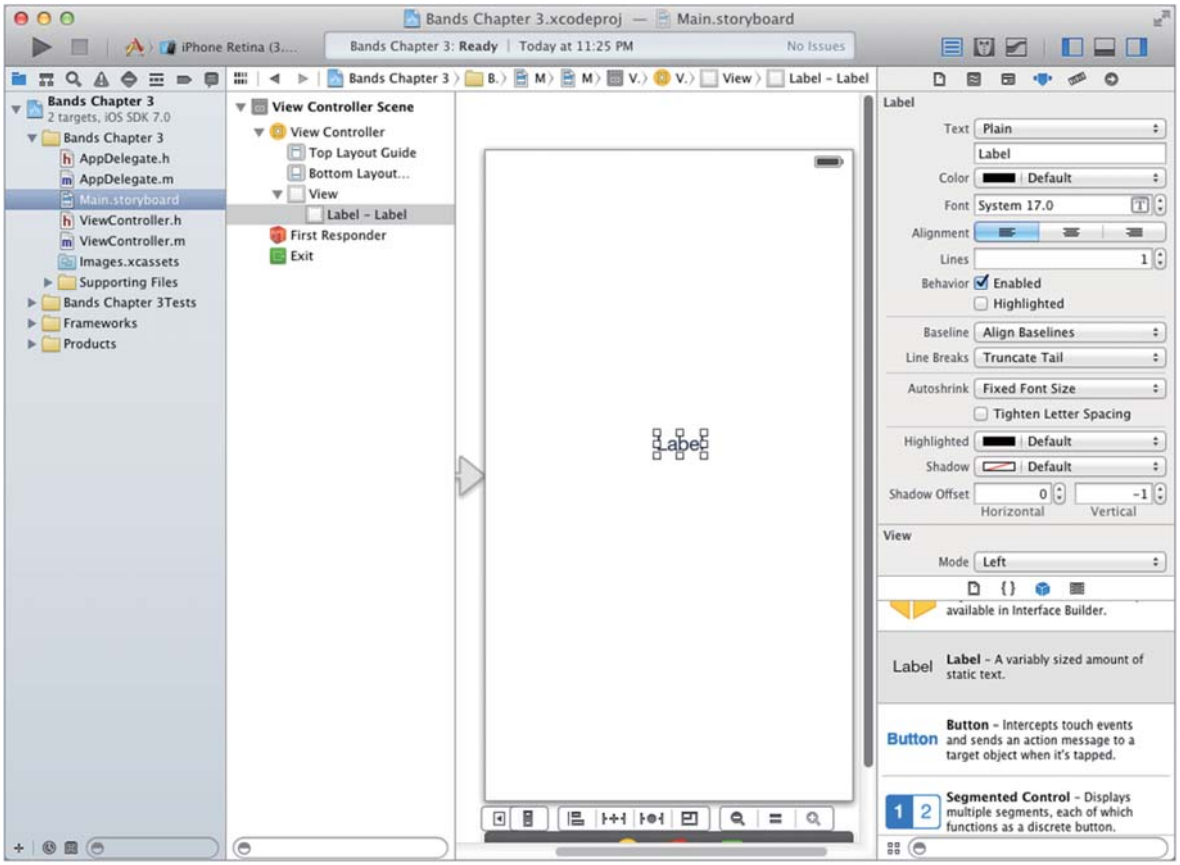

Figure 7: Drag Label onto Interface Builder

The storyboard is the user interface file for the project. In the Model-View-Controller design pattern, it is the "view" role. The project right now has only one scene that is represented by a single `UIView` object of the `UIKit` framework. You build the user interface of an application by adding other `UIKit` objects to the storyboard.

In Xcode these objects are all listed in the Utility pane. You can use the search bar at the bottom of the pane to filter the list and quickly find the object you are looking for. In the above steps, you have added a `UILabel` object to the base `UIView`. This is how you will build the user interface for the **Bands** app.

Exploring Interface Builder

Interface Builder is the user interface editor in Xcode.

The left side of the editor shows all the user interface objects and their hierarchy. In **Figure 7**, notice that the **Label** is listed under the **View** because it is a subview. Selecting objects in the hierarchy selects them in the scene shown in the main portion of the Editor. The **Utility pane** in Interface Builder has a series of inspectors on top and the collection libraries on bottom. The most commonly used library is the **Objects** library, which lists all the `UIKit` objects used to build an iOS application.

Setting Attributes

After you have added a UI object to your view, you need to set its attributes.

1. Select the required label.
2. On the Utility pane, select the Attributes inspector represented by the slider icon.

3. Change the text of the label by replacing **Label** with **Bands**.
4. In the view, drag the side boundary of the label to fit the new text, as shown in **Figure 8**.

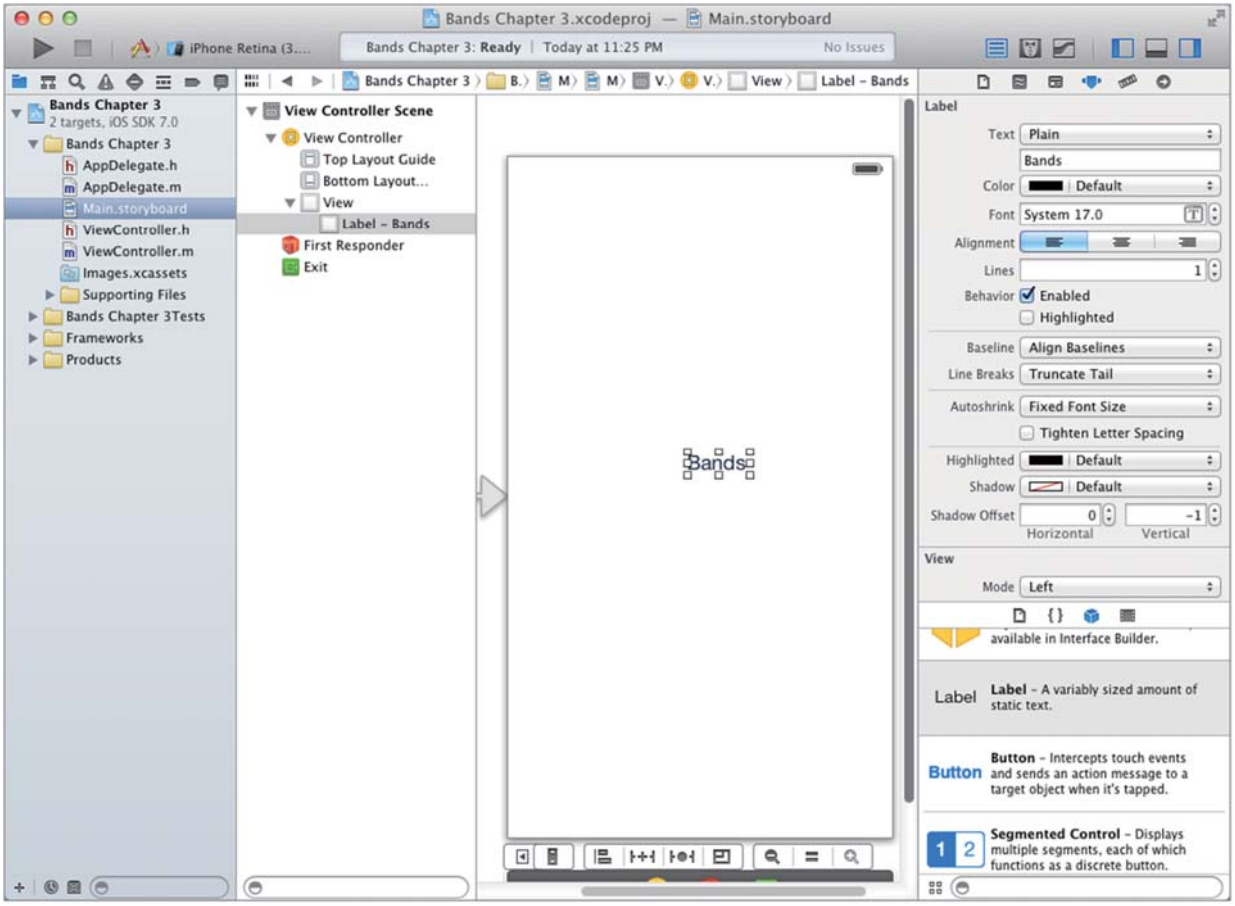

Figure 8: Setting the Label Text Attribute

All `UIKit` objects have attributes. These attributes can be set either at runtime or at design time. Changing the text from **Label** to **Bands** sets the text attribute of the label. Other attributes you can set for a label are all your typical text attributes, such as color, font, and alignment.

Exploring Inspectors

There are six inspectors on the Utility pane:

- Attributes inspector
- Quick Help inspector
- Size inspector
- File inspector
- Identity inspector
- Connections inspector

Let's consider each inspector below.

Attributes Inspector

The Attributes inspector (**Figure 9**) enables you to set the font style, color, size, and alignment of a `UIObject`.

File Inspector

The File inspector (**Figure 10**) shows the attributes of a file.

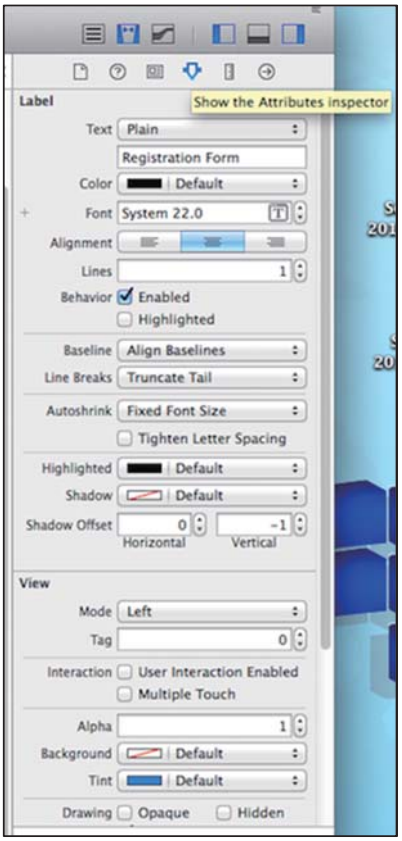

Figure 9: Attributes Inspector

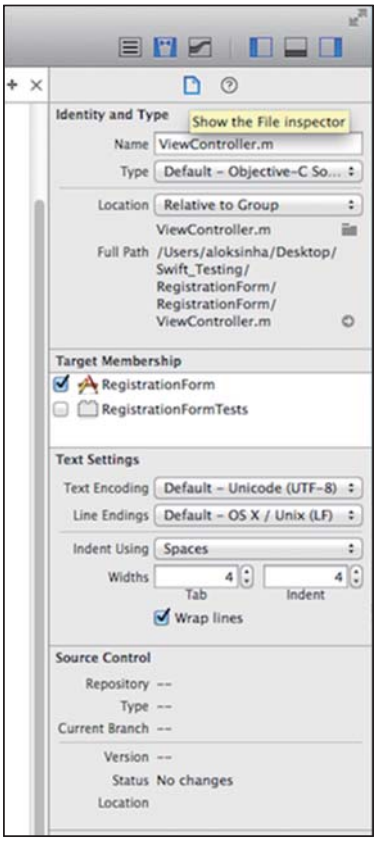

Figure 10: File Inspector

Quick Help Inspector

The Quick Help inspector (**Figure 11**) shows the documentation of a selected object.

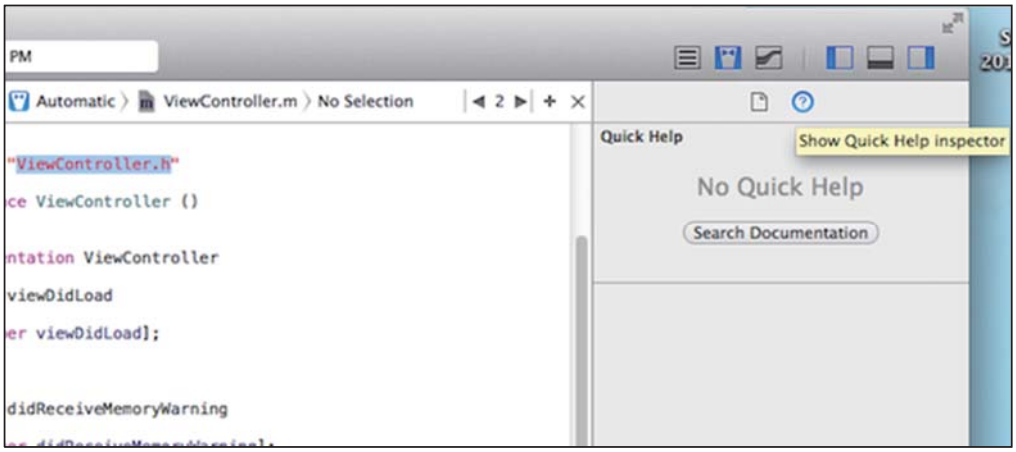

Figure 11: Quick Help Inspector

Identity Inspector

The Identity inspector (**Figure 12**) enables you to set the parent class of an object.

Size Inspector

The Size inspector (**Figure 13**) enables you to change the size and origin of an object as well as its auto layout constraints.

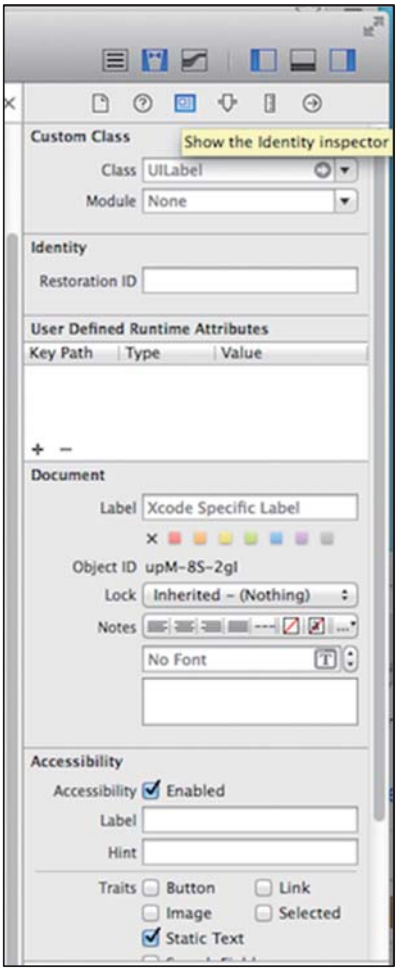

Figure 12: Identity Inspector

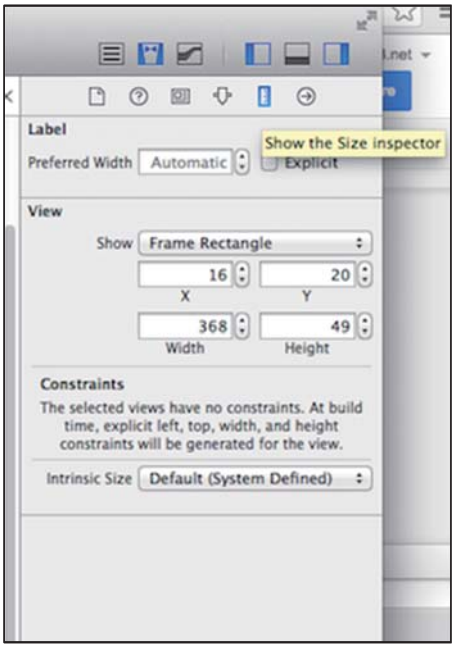

Figure 13: Size Inspector

Connections Inspector

The Connections inspector (**Figure 14**) enables you to connect actions and outlets.

Figure 14: Connections Inspector

Aligning `UIObjects`

Interface Builder shows various guidelines as you add **subviews**. These guidelines, shown as dashed lines, are designed by Apple to recommend spacing between objects and their alignment to the boundaries of the view.

Centering a `UIObject`

1. Select the label in the view.
2. Drag the label to the left side of the view until you see the left guideline appear.
3. Drag the label to the right side of the view until you see the right guideline appear.
4. Drag the label to the center of the screen until you see the horizontal and vertical centering guidelines.

Interface Builder shows you various guidelines as you move `UIObjects` around in the view. You can use these to place your objects in context to other objects and boundaries.

Only one application can run on an iPhone at any time (except for some built-in applications by Apple). Therefore, when you press the **Home** button on your iPhone, your application exits. Tapping an application icon starts the application all over again.

RUNNING AN APP IN THE SIMULATOR

Xcode includes the iOS simulator to help you quickly develop and prototype your app without needing to run on a device. Debugging with the simulator is faster than on a device and enables you to access some Xcode tools that are not available to the device. The simulator runs in its own window on your Mac. You can simulate iPhone 3.5-inch and 4-inch devices as well as iPads both in standard or retina display. You can also change the version of the OS the simulator runs.

Running an App in the iPhone Retina (4-Inch) Simulator

To launch the simulator, you first need to select the device with which you would like to test. The steps to run an app in the iPhone Retina (4-inch) simulator are as follows:

1. In Xcode, locate the scheme selector next to the **Run** button.
2. From the **iOS simulator** drop-down list, select **iPhone Retina (4-inch)** in the simulator section.
3. Click the **Run** button. The simulator launches and runs your application, as shown in **Figure 15**.

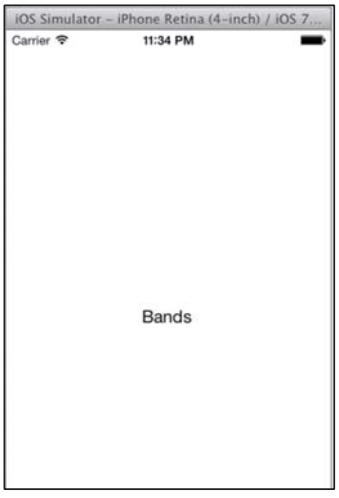

Figure 15: Run an App in iPhone Retina (4-inch) Simulator

Using the scheme selector, you selected to run the application in the simulator on an iPhone Retina (4-inch) device. The app was then compiled, installed, and launched in the simulator using your selection.

Running the App in the iPhone Retina (3.5-Inch) Simulator

You must test your application on all devices you plan to support because of different screen sizes. You also need to keep these different sizes in mind while designing your user interface.

The following steps demonstrate a layout issue found only by testing with multiple devices:

1. In Xcode, select the **iPhone Retina (3.5-inch)** device in the simulator section.
2. Click the **Run** button. The simulator switches to the iPhone Retina (3.5-inch) device, as shown in **Figure 16**.

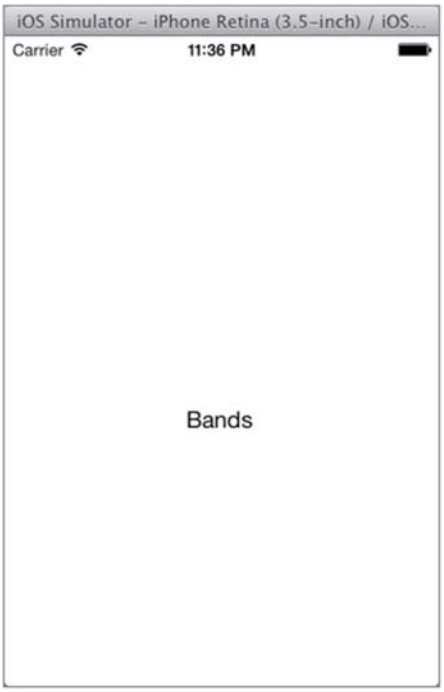

Figure 16: Run an App in iPhone Retina (3.5-Inch) Simulator

What you have done here is to switch the device type the simulator should use. As a result, the label you centered in Interface Builder is no longer centered in the simulated application. This is because the iPhone Retina (3.5-inch) has less screen space. To keep the label centered, you need to use auto layout.

Testing Rotation on the Simulator

Rotating a device also changes the screen size of your app. The simulator enables you to test rotation of your app along with the device.

1. Run your app in the iPhone (3.5-inch) simulator.
2. Select **Rotate Left** from the **Hardware** menu. The simulator rotates to the left and moves to the Landscape orientation, as shown in **Figure 17**.

Figure 17: Rotating the Simulator Left

3. Now, select **Rotate Right** from the **Hardware** menu. The simulator rotates to the right and returns to the Portrait orientation.

The simulator enables you to test rotation in your app by giving you hardware commands to rotate the simulated device. You can also use the keyboard instead of the menu using the Command-arrow key combination. It is important to test your application in all orientations you plan on supporting.

It should be noted that supporting rotation in your application is an optional feature. You set which orientations you support in the application settings.

ADDITIONAL KNOWHOW

The iPhone simulator is a simulator, and not an emulator. Often, an emulator is a complete re-implementation of a particular device or platform. The emulator acts exactly like the real device would. A simulator is a partial implementation of a device/platform. It tries to mimic the behavior of a real iPhone device. It uses the various libraries installed on Mac to perform its rendering so that the effect looks the same as on an actual iPhone. Also, applications tested on the simulator are compiled into x86 code, which is in the byte-code understood by the simulator.

LEARNING ABOUT AUTO LAYOUT

Auto layout is a feature of Xcode that helps you define the location of your user interface objects, irrespective of the size of the screen on which your app runs. iPhones with a 3.5-inch display have less screen space than iPhones with a 4-inch display. The size of the screen can also change if the device is rotated. The auto layout feature enables you to ensure that your label remains centered on the screen.

Setting Auto Layout Constraints

Auto layout works by adding different constraints to your user interface objects. Constraints can be set between an object and its container. There are two methods to set auto layout constraints:

- Using the Alignment Constraints options
- Using the Control-drag method

Setting Auto Layout Constraints using the Alignment Constraint Options

1. In Xcode, select `Main.storyboard` in the Project Navigator.
2. Select your label in Interface Builder.
3. At the bottom of the screen, click the **Align** button to display the **Add New Alignment Constraints** options, as shown in **Figure 18**.

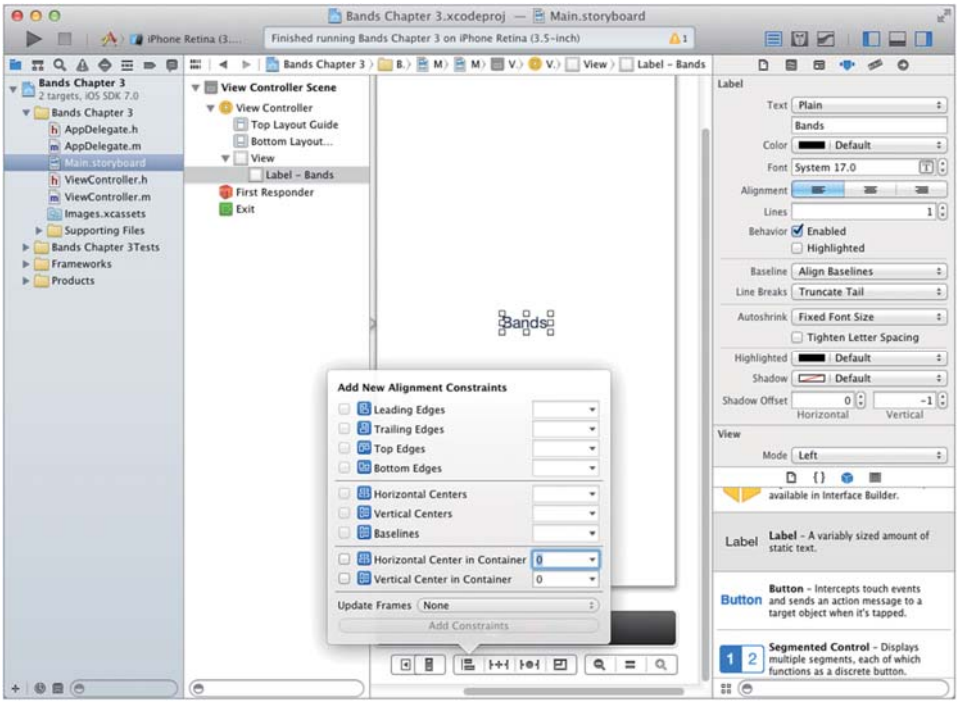

Figure 18: Add New Alignment Constraints Options

4. Select the **Horizontal Center in Container** check box and the **Vertical Center in Container** check box, and then click the **Add Constraints** button. Xcode adds lines, which represent your constraints, as shown in **Figure 19**.

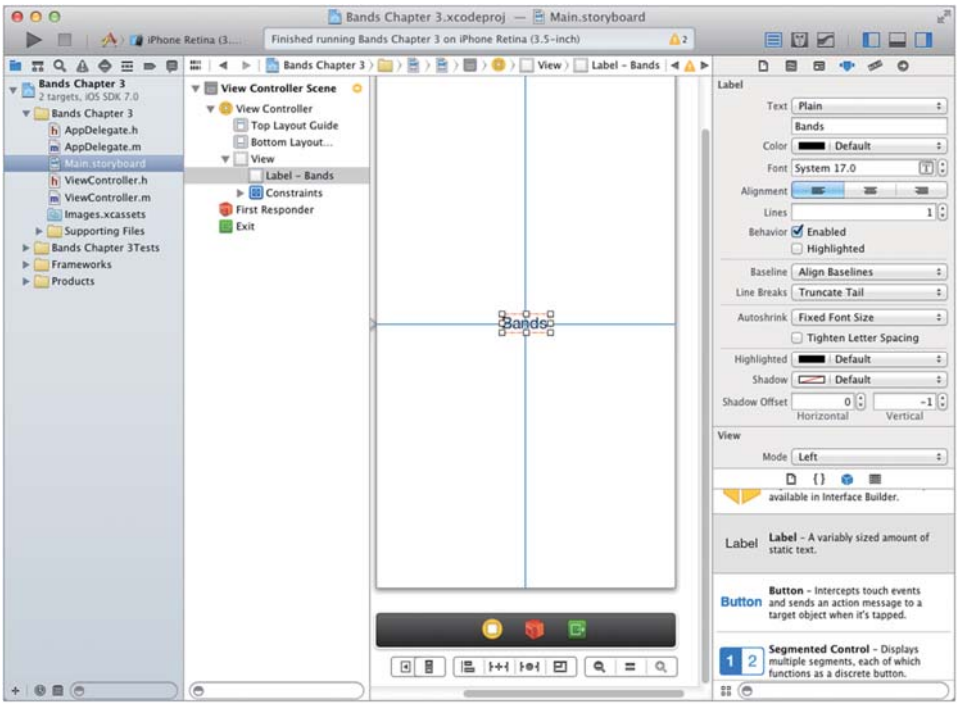

Figure 19: Adding Lines as Constraints

5. Run the app in the iPhone Retina (3.5-inch) simulator. Your label remains centered, as shown in **Figure 20**.

Figure 20: Running an App in iPhone Retina (3.5-Inch) Simulator

The **center horizontally** and **center vertically** constraints enable you to set your label to be always centered in the view, irrespective of the size of the screen.

Setting Auto Layout Constraints by the Control-Drag Method

1. Select the object to which you want to add a constraint.
2. Drag a line to the other object you want in the constraint while keeping the `<Ctrl>` key pressed.
3. Highlight it as shown in **Figure 21**. When you release the mouse button, a dialog box with the available constraints will be displayed.

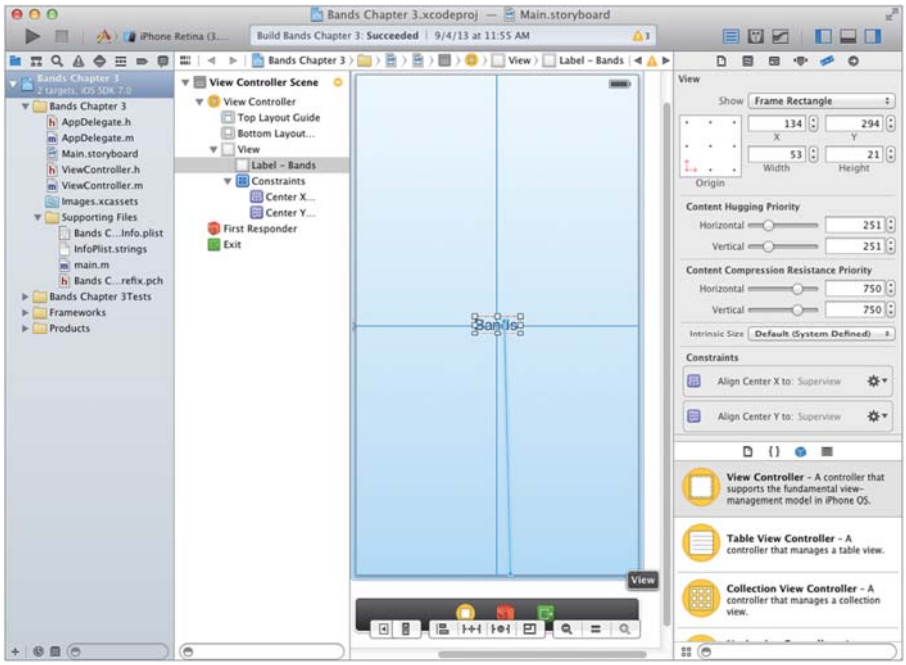

Figure 21: Auto Layout Constraints by Dragging the Mouse

4. You can view the constraints of each object in the storyboard hierarchy on the left side of Interface Builder or by selecting the Size inspector in the Utility pane, as shown in **Figure 22**.

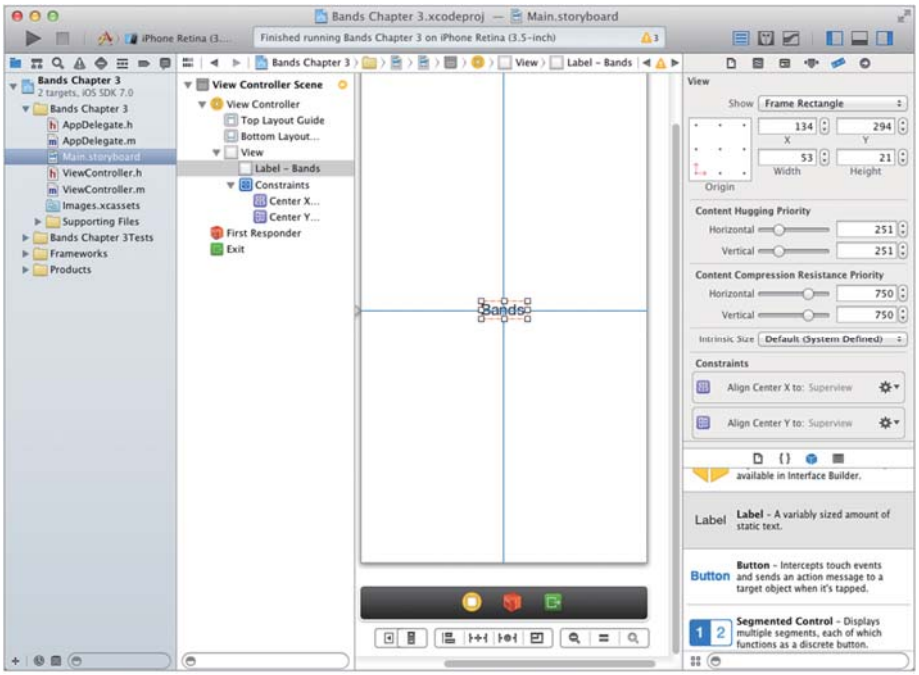

Figure 22: Displayed Constraints of Each Object

To learn more about auto layout in Interface Builder, you can check Apple's help documentation at *https://developer.apple.com/library/ios/documentation/userExperience/Conceptual/AutolayoutPG/Introduction/Introduction.html.*

EXPLORING APPLICATION SETTINGS

Now that you have your application running, it is time to set its version number, icon, and other settings to make it complete. Xcode uses property list files, known as `plists`, to store this information. You can view your application's `info plist` by expanding the **Supporting Files** group in the Project Navigator and selecting the `Bands-info.plist` file. Though you can edit your settings using the plist Editor, you may find it easier to use Xcode's Settings Editor instead.

In this section, you will learn how to set the following application settings:

- Version and build numbers
- App icons
- Supported rotation orientations
- Launch images

Setting Version and Build Numbers

Every iOS application, just like any desktop application, has a version and a build number used to identify the version of the app. This section walks you through setting these versions using Xcode's Settings Editor.

Setting Properties using the Information Property Editor

1. In the Project Navigator, select the project.
2. In the Editor, select the **General** tab, and display the Info Property Editor, as shown in **Figure 23.**

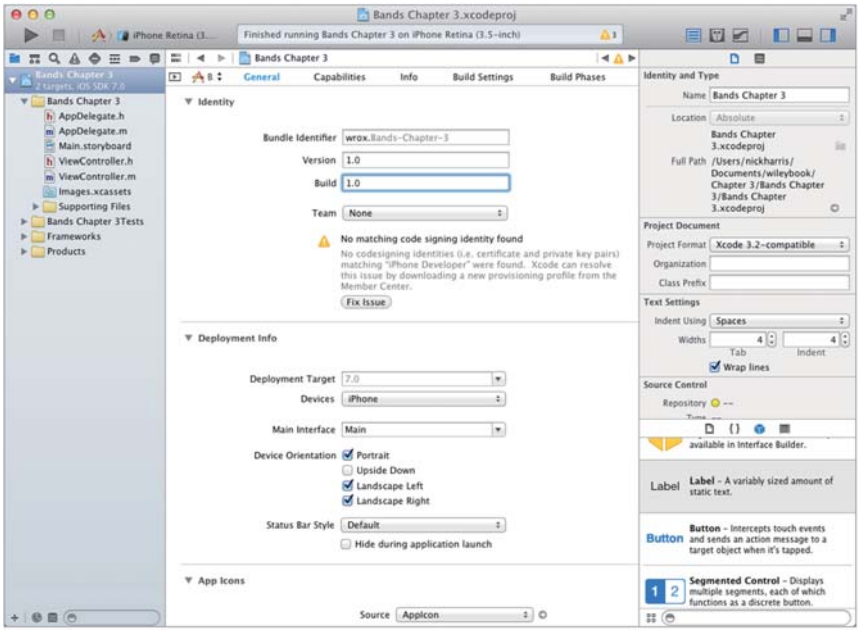

Figure 23: Info Property Editor

3. In the **Identity** section, set the **Version** to **1.0** and **Build** to **0.1** (see **Figure 23**).
4. Select the `Bands-info.plist` file in the Project Navigator. You can see the **Bundle version string,short** property set to **1.0** and the **Bundle version** property set to **0.1**, as shown in **Figure 24**.

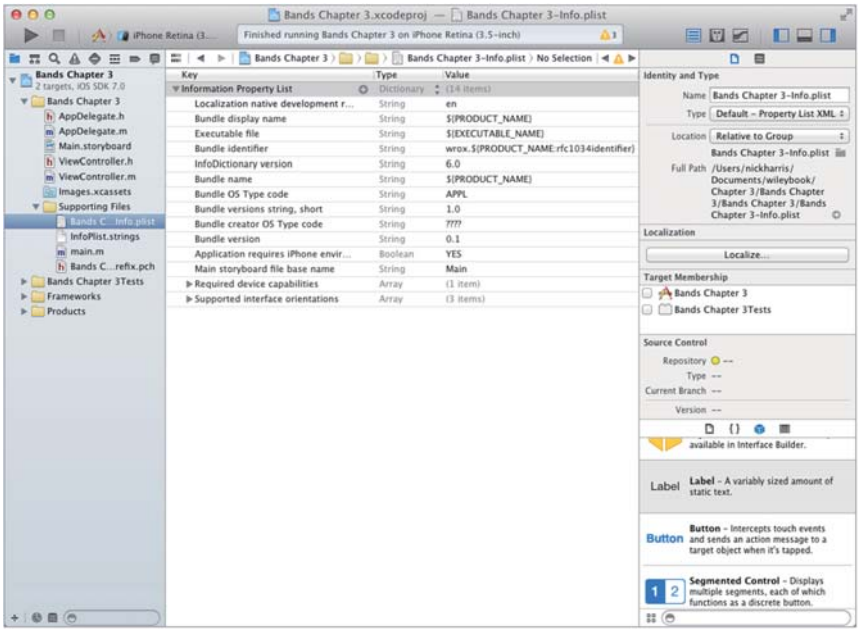

Figure 24: Xcode Information Property Editor

The Xcode Information Property Editor helps you change your application settings without needing to know the specific property names used in the underlying `Bands-info.plist` file.

Setting Supported Rotation Orientations

Supporting **rotation** in your application is optional. It might be a challenge to design an application that looks nice in both **Portrait** and **Landscape orientations**, which is not worth the effort. Therefore, you can choose to support either Portrait or Landscape orientation in your app.

1. In the Project Navigator, select your project.
2. In the Deployment Info section, select the **Portrait** check box and deselect the **Upside Down**, **Landscape Left**, and **Landscape Right** check boxes.
3. Run the application in the iPhone Retina (4-inch) Simulator.
4. Rotate the simulator to Landscape orientation. You can see that the app no longer rotates to this orientation, as shown in **Figure 25**.

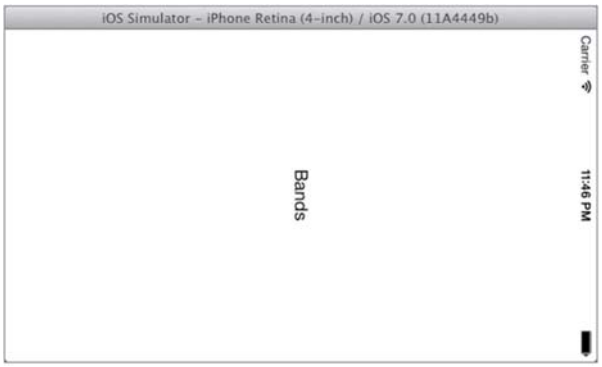

Figure 25: Setting Portrait Orientation as Supported Rotation

In your application settings, there is an array of supported orientations, which are listed under the Supported Interface Orientations property. By checking only Portrait in the Info Property Editor, you remove all other orientations, leaving just Portrait in the array.

Setting the App Icon

An application is not complete without an icon. The icon is one of the most important parts of your application. It is the first thing a user will see when browsing the App Store, so it needs to look nice as well as catchy. This course will not delve into what makes for a good icon, but Apple does layout some Human Interface Guidelines at *https://developer.apple.com/library/ios/documentation /userexperience /conceptual/mobilehig/ AppIcons.html*. If you can afford to, it is best to hire a designer to create your icon.

You will need icons of different sizes depending on the devices you plan to support.

- For iPhones and iPod touches with a retina display, you need an icon that is 120 × 120 pixels in size.
- For non-retina display in iPhones and iPod touches, you need an icon that is half that size, which is 60 × 60.
- For retina display in iPads and iPad minis, you need an icon that is 152 × 152 in size.
- For non-retina display in iPad and iPad minis, you need an icon that is 76 × 76 in size.

Since this course is primarily about coding an iOS application and not designing, you can use the simple 120 × 120 icon included with the sample code. The steps to set the app icon in Xcode are given in the next section.

Setting the Bands App Icon

1. Download the appropriate image, name it as `BandsIcon.png` and save it on your desktop.
2. In the Project Navigator, select the project.
3. In the Appicons section, click the arrow next to the **Source** drop-down list to bring up the Icon Settings Editor, as shown in **Figure 26**.

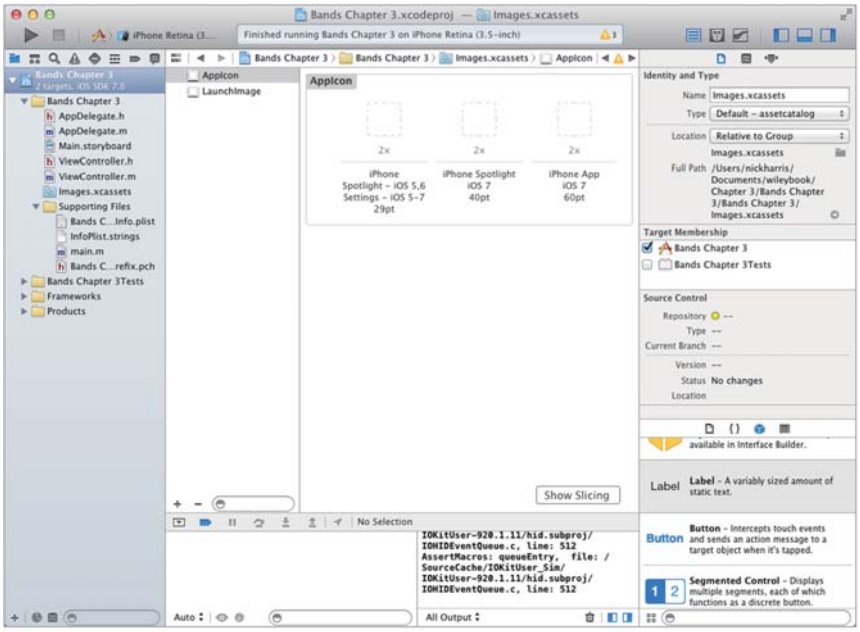

Figure 26: Icon Settings Editor

4. Drag the `BandsIcon.png` file onto Xcode, and drop it on the **iPhone App iOS 7** placeholder.
5. Run the application in the simulator.
6. Return to the Home screen in the simulator by selecting **Home** from the **Hardware** menu. You can now see the new icon in the simulator Home screen, as shown in **Figure 27**.

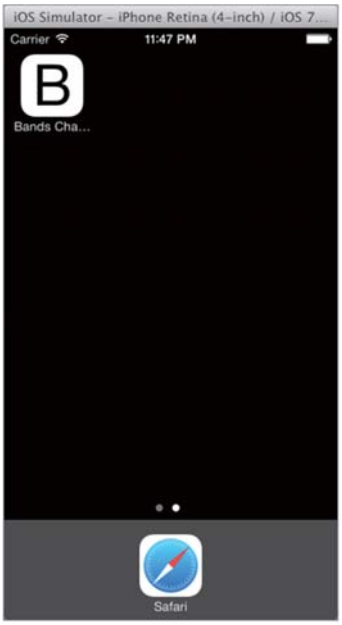

Figure 27: Bands Icon in the Simulator

Application icons have specific names that are used to identify them properly. Instead of needing to know these names, Xcode uses asset catalogs. By dragging a file onto the Xcode Icon Settings Editor, you have added the icon to the project. As a result, Xcode adds it to its proper asset catalog, so the system knows how to show your app icon on the Home screen.

Icon picture and icon name play a great role in attracting users towards an application. The icon of your app should be a link with your app's action and should create the picture of your application in the user's mind.

Setting Launch Images

When iOS first launches your application, it may take a little bit of time before the app is ready to use. Instead of showing a blank screen, Apple requires you to supply a launch image. The image can be anything you want it to be, but it is recommended that you use an image that represents what your app will look like when it is ready to use. This creates a seamless visual experience to the user. You need to provide launch images for all device sizes and orientations your app supports.

The following section shows how to create launch images and set them in your project.

Creating and Setting Launch Images

1. Run your app in the iPhone Retina (4-inch) simulator.
2. Select the **Save Screen Shot** option from the **File** menu in the simulator. This creates a new PNG image on your desktop.
3. In Xcode, select your project from the Project Navigator.
4. In the Launchimage section, click the arrow next to the Source selector to bring up the Launch Image Editor, as shown in **Figure 28**.

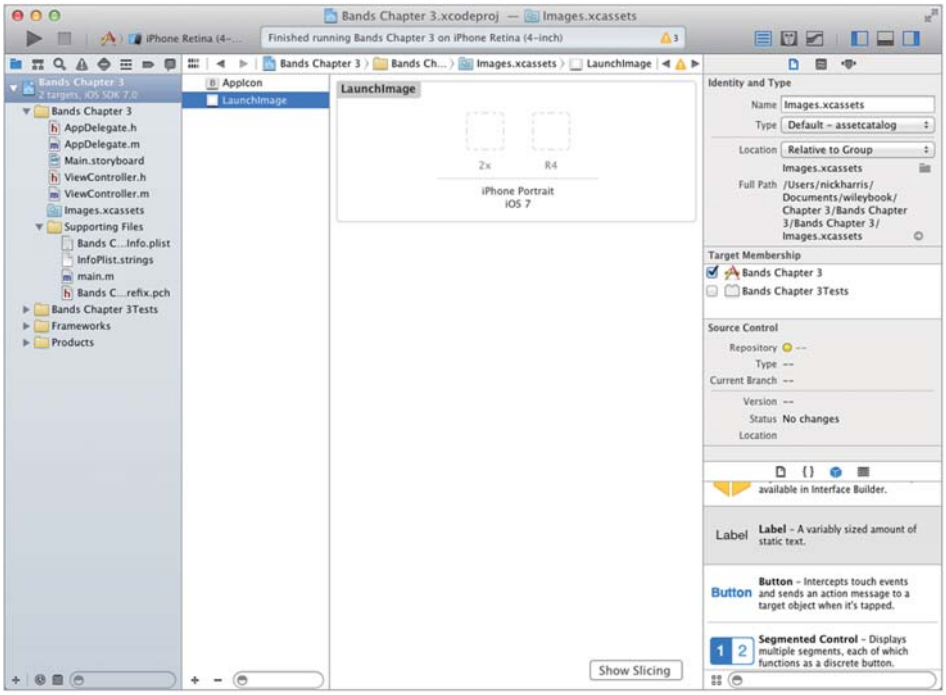

Figure 28: Launch Image Editor

5. Drag the screenshot file from your desktop to the second placeholder. The project will automatically be saved once the image is dropped.
6. Run your application again using the iPhone Retina (3.5-inch) simulator.

7. Repeat step 2 to create another screenshot image on your desktop.
8. Drag the new screenshot file from your desktop to the first placeholder. Again, the project will automatically be saved once the image is dropped.

Xcode determines if your application supports iPhone 4-inch displays by looking at what launch images are included. If no images are included, it assumes that the app supports the 4-inch display. By using the screenshot feature of the simulator, you have created image files that represent your user interface after the app is loaded and ready for use. Similar to icons, iOS uses specific names to know which launch images to use on different devices. Xcode saves you from having to know these names again by using the asset catalog, so that the system automatically knows which images to use without you memorizing the specific file names.

RUNNING AN APP ON A DEVICE

Running your application in the simulator enables you to quickly prototype and test your app. However, to actually get a feel for your application, you need to test it on an actual device.

To do this, you need to first enroll in the iOS developer program (*https://developer.apple.com/devcenter/ios/)*. This developer program costs **US $99 a** year at the time of this writing, but it is necessary to test on physical devices as well as getting your app into the App Store. This program also helps you to get access to all Apple's technical resources including the Apple Developer Forums.

All apps that run on an iOS device require a provisioning profile. The provisioning profile is a form of Digital Rights Management (DRM). When you buy an app from the App Store, it gets installed using an App Store provisioning profile. When you test your app during development, you need to install a developer provisioning profile. Both types of provisioning profiles require a certificate, which is used to sign the profile. Xcode can handle this for you, as shown in the following section.

Provisioning a Device for Testing

1. Connect your device to your Mac.
2. In Xcode, open the **Organizer** from the **Window** menu.
3. Select the **Devices** icon on the top of the **Organizer** window, as shown in **Figure 29**.

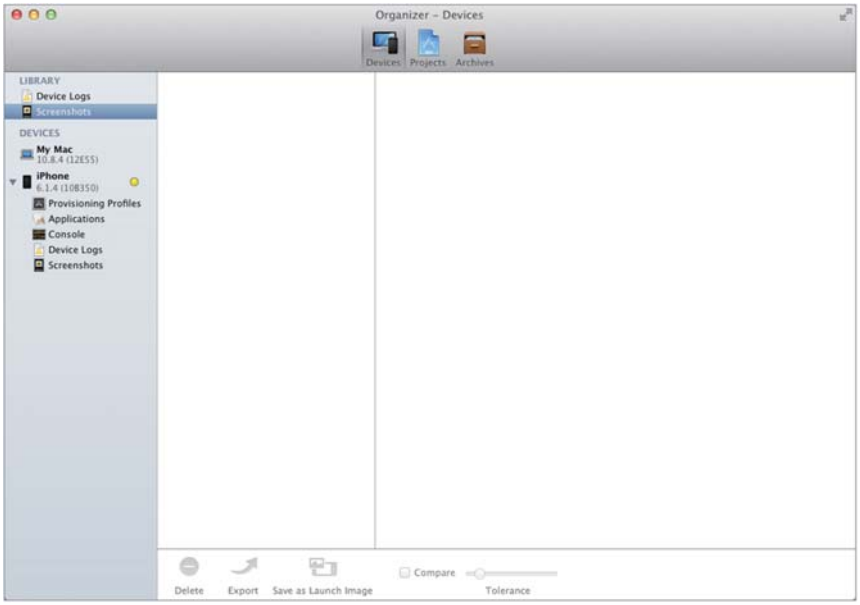

Figure 29: Select the Devices Icon

4. Select your connected device (see **Figure 30**) and then click the **Use for Development** button in the main part of the window.

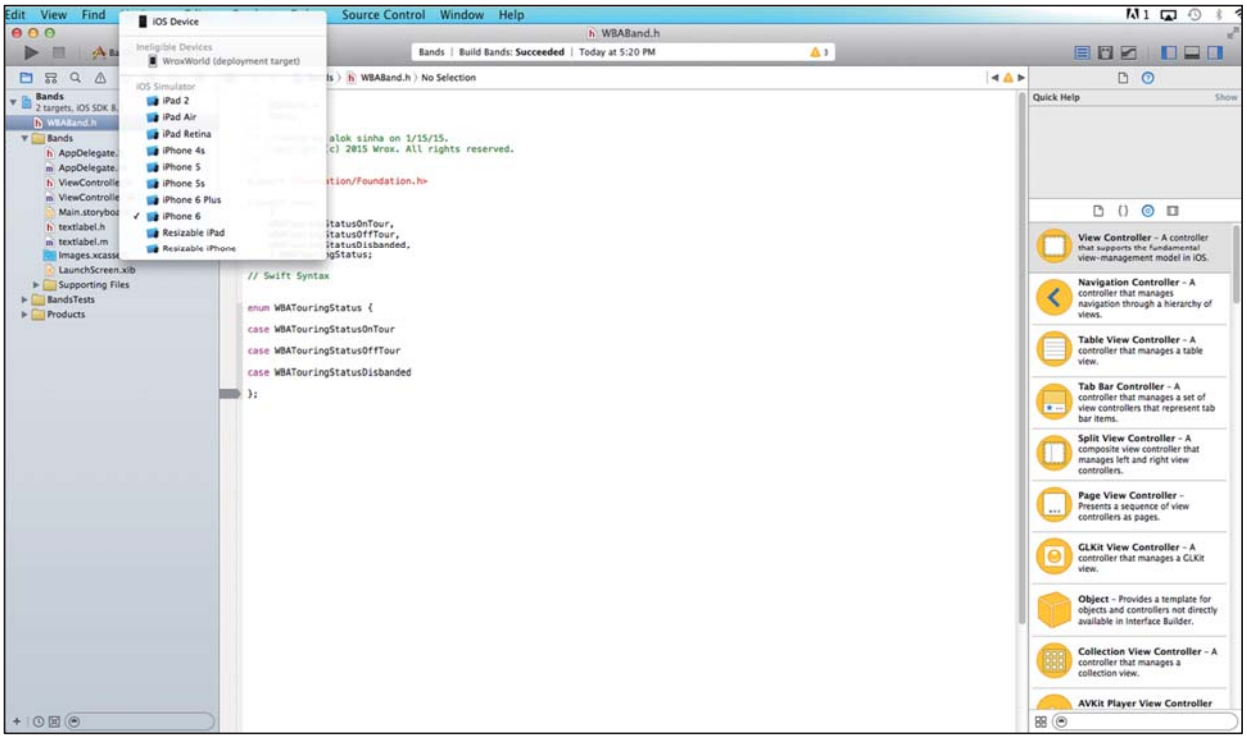

Figure 30: Select Your Connected Device

5. If you are already registered with an Apple Developer Program, add your Apple ID and password in the dialog box that appears (see **Figure 31**) and click **Add** to connect your device. If you are not registered, then click the **Join a Program** button.

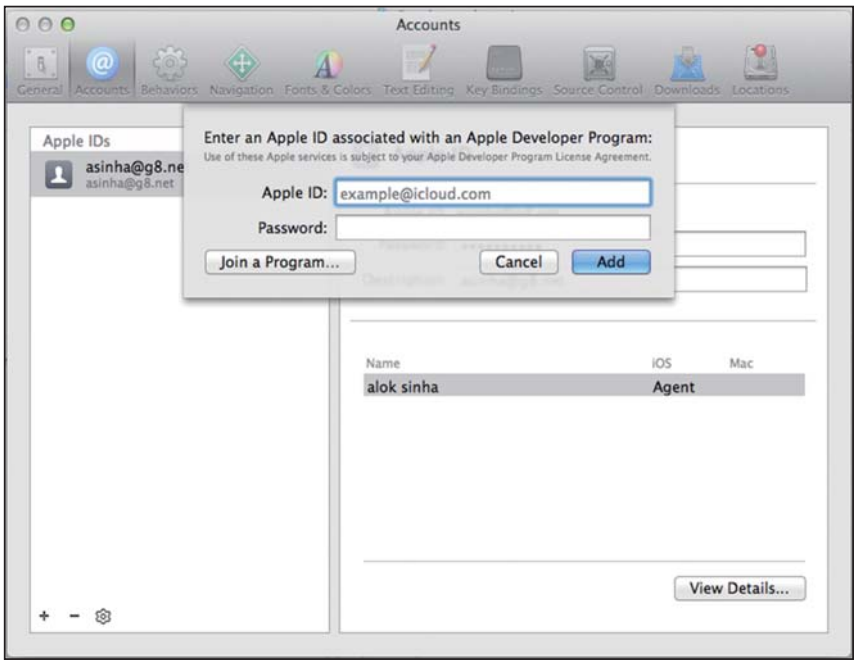

Figure 31: Enter Your Registered Apple ID for the Connected Device

6. This displays the **Certificate Not Found** dialog box (see **Figure 32**). Click **Request** to request a new certificate and wait for Xcode to complete the task.

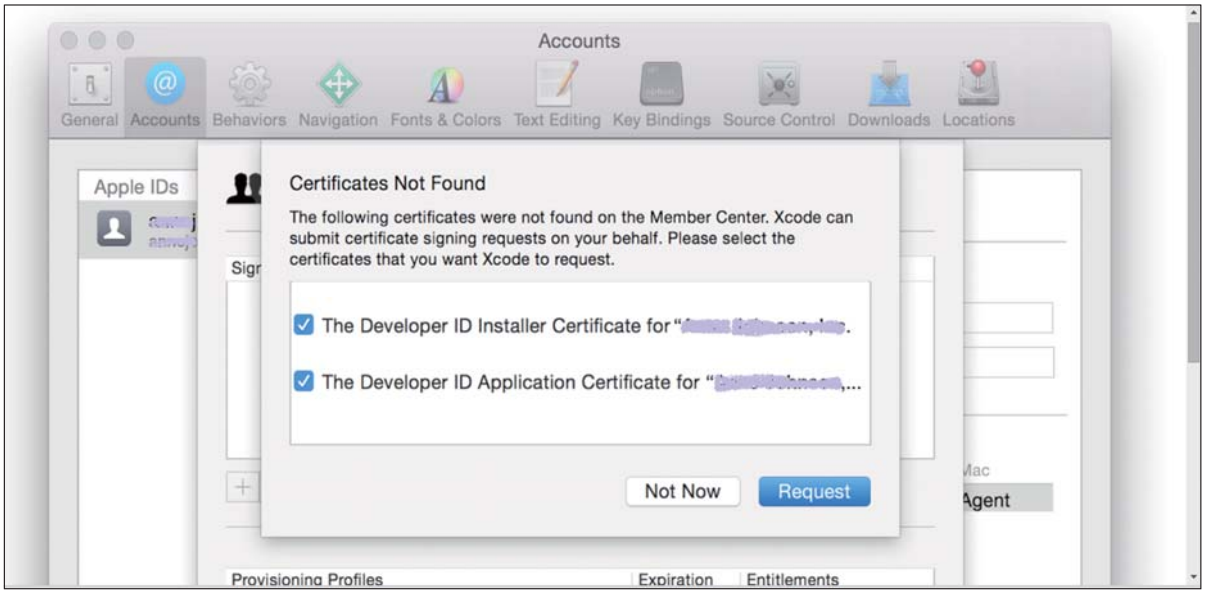

Figure 32: Request Certificates via Xcode

7. In Xcode, select the connected iOS device in the scheme selector (see **Figure 33**).

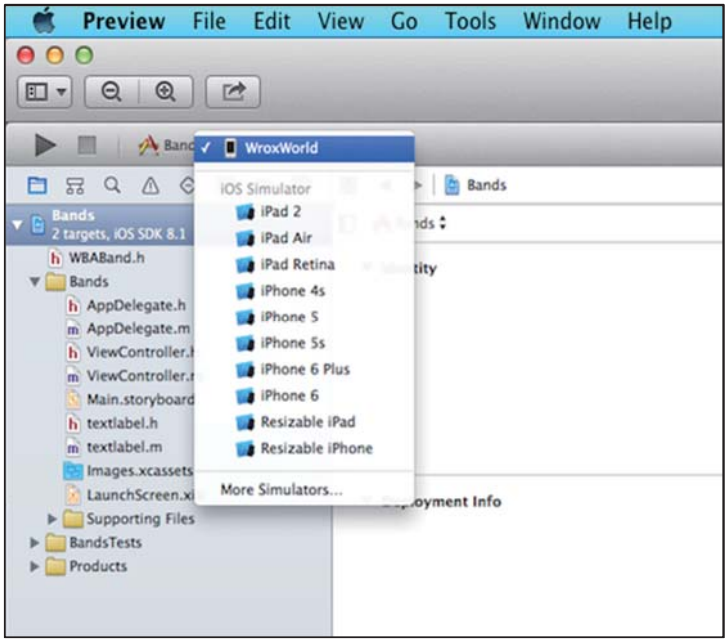

Figure 33: Select the Connected iOS device

8. Click **Run** to run your application on the connected device (see **Figure 34**). Xcode then installs the application on your device and runs it.

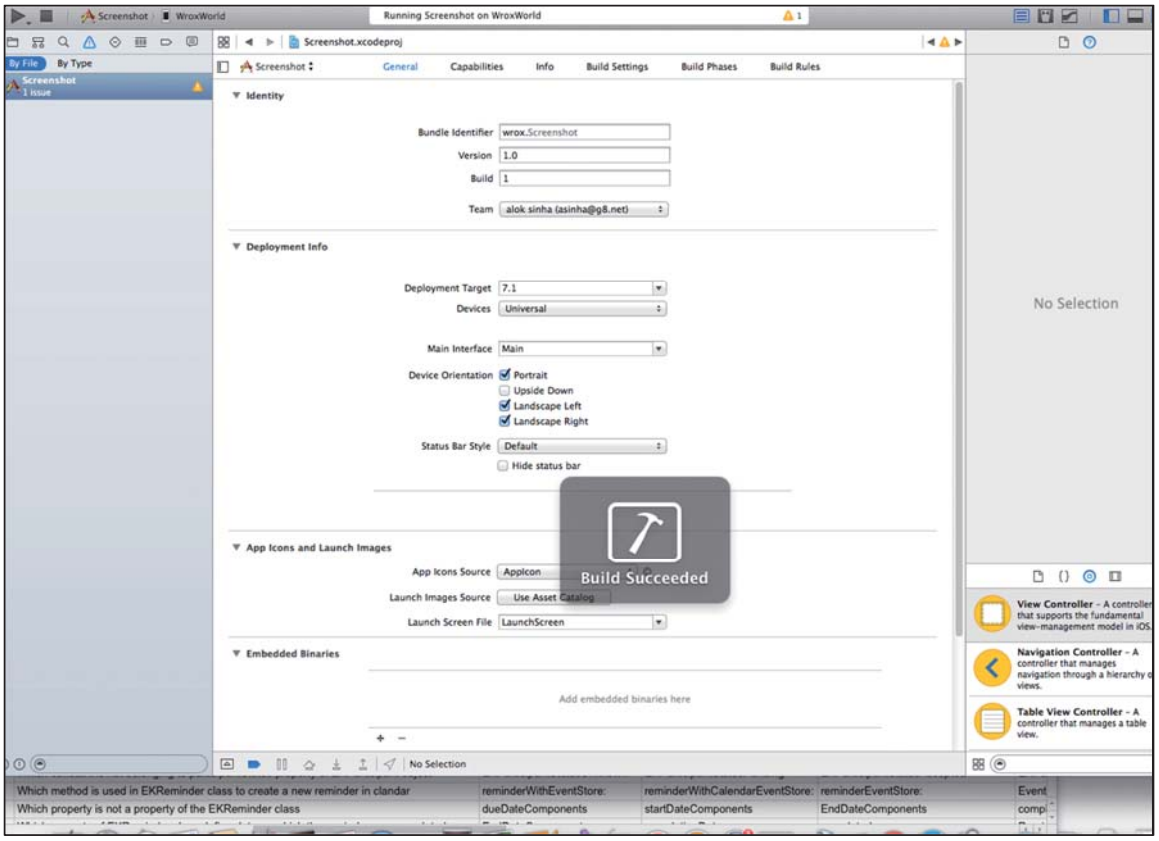

Figure 34: Click Run to Run App on the Connected Device

You have now registered your device with your developer account, created a certificate, and then created a team provisioning profile. This profile is managed by Xcode and can now be used to test your application on your device.

App development comes alive when you begin to build and test your own apps. Xcode includes a selection of simple app templates to help you get started. The official documentation includes hundreds of sample projects, and you can find many more projects online—all free. Building and running them in the simulator is easy, but getting them to run in a real device is only slightly more difficult and more rewarding.

During the lab hour of this session, you will set up the Xcode development environment and create a simple palindrome application in it.

Cheat Sheet

- All iOS applications are built in Xcode. The Xcode project organizes all of the code files, art assets, configuration files, and settings.
- The user interface is the “view” role in the Model-View-Controller design pattern. It is the part of the application the user interacts with. In Xcode, the user interface is built using storyboards with scenes and segues showing the flow of the application.
- iOS devices come in different sizes. They are also handheld devices that the user can rotate, which changes the screen dimensions. Apple has designed its auto layout feature in Xcode to ensure your user interface is displayed correctly, irrespective of the device or orientation in which it is being viewed.
- There are many settings to an iOS application. They are stored in property list files, also known as `plist` files. Xcode includes editors you can use to change these settings, so you do not need to know the keys and valid setting options, though you can edit the `plist` files directly if you choose.
- The iOS simulator is an essential tool in developing iOS applications. It allows you to test your application using any iOS device quickly without needing test devices connected to your development machine.
- To test your application on a physical iOS device, you need to register and provision the app with Apple through your developer account. Provisioning a test device can be done within Xcode.