Mobile Application Development for iOS – Part 1: Building Your First iOS App

Prof. Xiaoping Jia, School of Computing, DePaul University

January 2016

Contact the author:
Email: xjia@cdm.depaul.edu
Web: http://venus.cs.depaul.edu/xjia

In this series, we will develop iOS applications using Xcode and iOS SDK. Development tools for iOS are currently only supported on MacOS X. A Mac computer running MacOS X 10.9.4 or later, i.e., OS X Mavericks (10.9.x), OS X Yosemite (10.10.x), or OS X El Capitan (10.11.x) is required. To develop applications for iOS 9, you must download and install Xcode 7.0 or later[1]. We will use Xcode 7.2 (released in December 2015) with support for iOS 9 SDK. It allows us to target the latest iOS devices running iOS 9, including *iPhone 6s*, *iPhone 6s Plus*, *iPad Pro*, *iPad Mini* and *iPad Air 2*.

[1] The latest version of Xcode can be downloaded from `https://developer.apple.com/xcode/`

Creating a New Project

Let's first launch Xcode. From the initial welcome screen,

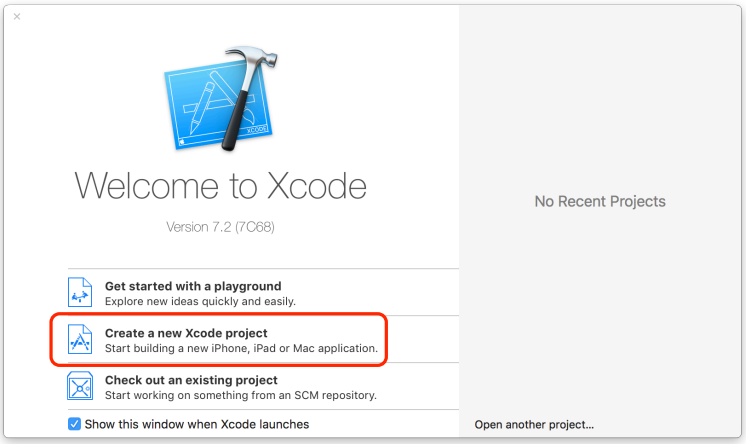


Figure 1: Xcode 7.2 welcome screen.

- Select "Create a new Xcode project"

XCODE WILL PRESENT YOU WITH A SERIES OF DIALOGS to collect the information and the options of the new project to be created. The first dialog will prompt you to choose a *project template*.

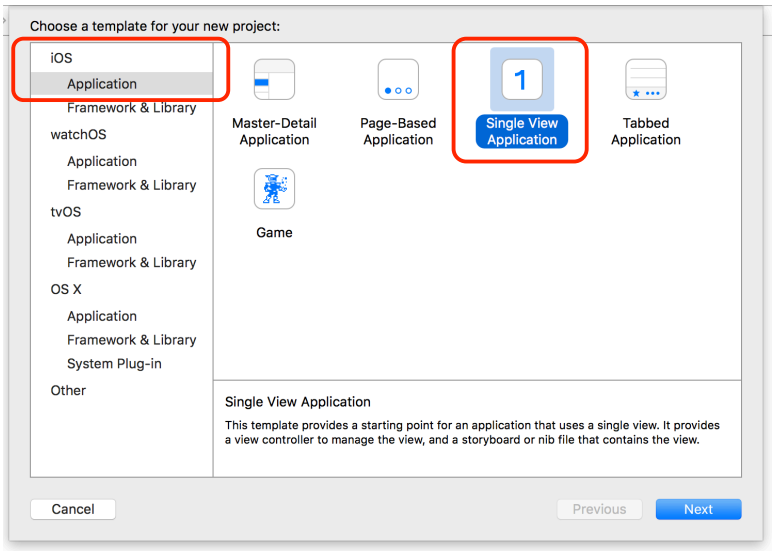


Figure 2: Xcode new project wizard, step 1: choose project template.

We choose the *Single View Application* template for an *iOS Application*. The template is a complete application with a single blank view that can be easily customized. This template offers an easy starting point for many simple applications.

- From the panel on the left side, select *Application* under *iOS*.
- From the panel on the right side, select *Single View Application*.
- Click *Next*

The next dialog is for project options.

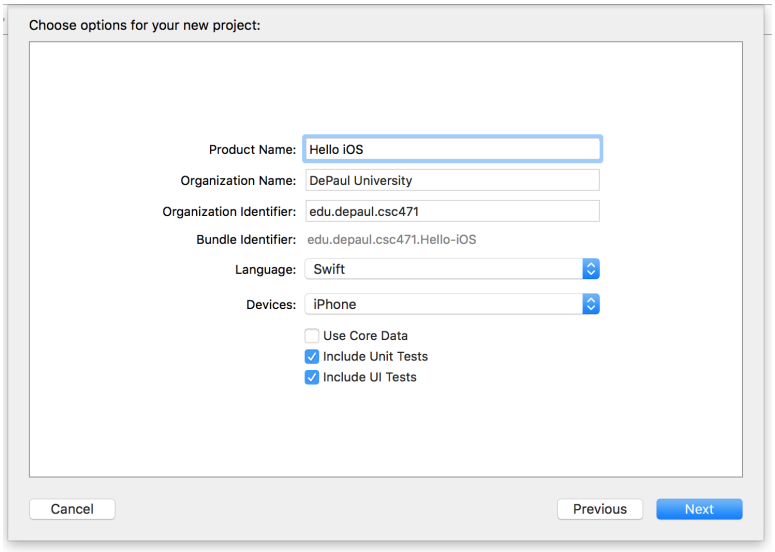


Figure 3: Xcode new project wizard, step 2: project options.

- *Project Name* and *Organization Name*: You may choose any appropriate name for these fields.
- *Organization Identifier*: A globally unique identifier for your organization. A common convention is to use the *reverse* Internet domain name of your organization, i.e., the components of the domain name in reverse order, e.g., `edu.depaul.csc471`
- *Bundle Identifier*: An identifier of your application, which must be globally unique. It is generated by Xcode, and it is derived from the organization identifier and the project name your have provided.
- *Language*: You have the options of *Swift* and *Objective-C*. Choose *Swift*.
- *Devices*: The device groups your application will target. The options are: *iPad*, *iPhone*, *Universal*. For this application, let's choose *iPhone*.
- *Use Core Data* check box: Whether your application will use the *Core Data* framework for on-device databases. For this application, we don't need a database, so leave the box unchecked.
- *Include Unit Tests* and *Include UI Tests* check boxes: Whether to include unit tests and/or UI tests in your project. Although it is not important in this simple project, you should include the tests in general. Keep the boxes checked.
- Click *Next*.

You will be prompted to choose a location for your project.

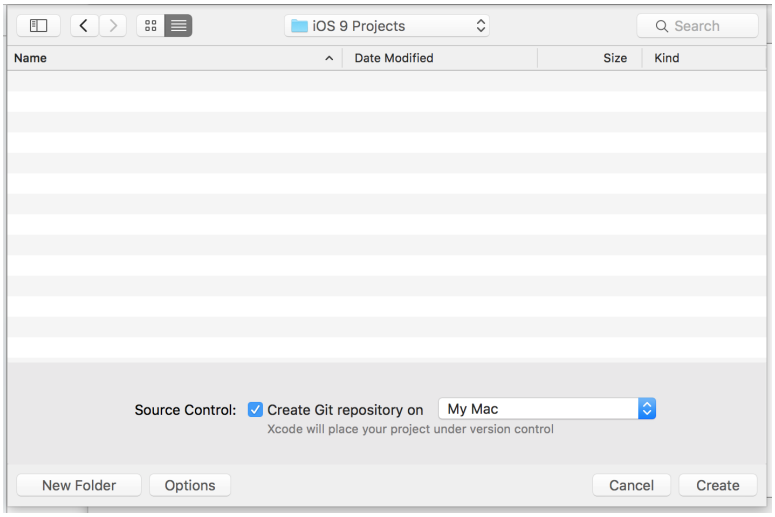


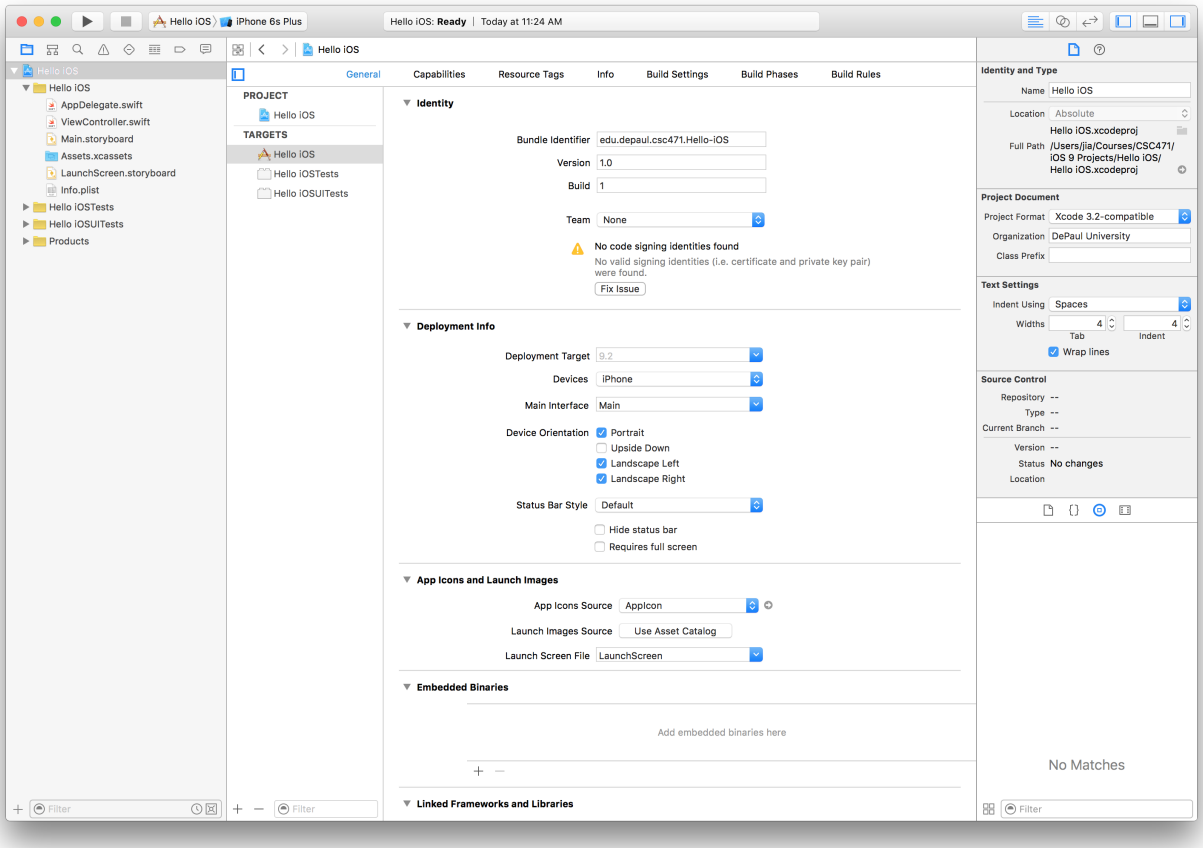
Figure 4: Xcode new project wizard, step 3: choose project folder location.

- Choose any folder or create a new folder. Xcode projects may reside anywhere on your computer.
- You have the option of creating a Git repository for source control. Although it is not necessary for your project to use a repository, it is generally a highly recommended practice to use a repository for any project other than throw-away toy projects. [2]
- Click *Create*

Your project will be saved in a new subfolder inside the folder you have chosen. The name of the project folder will be the same as the project name.

[2] Using a repository for source control offers two main benefits: a) version control of all source files, b) shared code base among the members of a development team. Git is a powerful and popular source control repository.

THE NEW PROJECT IS CREATED. The following is the initial view of the new project in Xcode workspace.



The newly created project is a complete project instantiated from the *Single View Application* template. It is ready to build and run. It will display a blank view. Of course, you will want to add some more interesting things to your app. But, before we do that, let's first get acquainted with some of the main features of Xcode.

Figure 5: A new project in the Xcode workspace.

If you notice a warning on "No code signing identities found," need not worry. This will only matter if you try to run your app on an iOS device. We will fix the issue when we get to that point.

The Xcode Workspace

THE XCODE WORKSPACE is generally divided into the following areas:

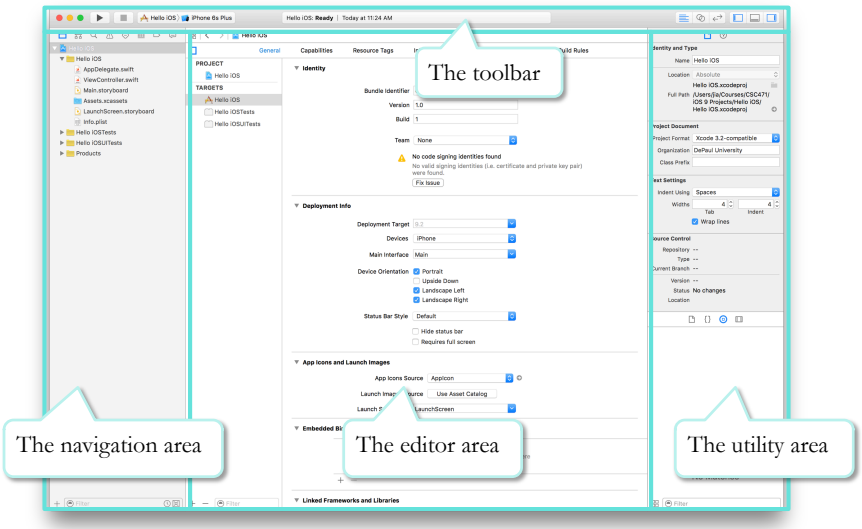


Figure 6: The layout of the Xcode workspace.

The *toolbar* is located at the top of the workspace. It contains a number of project-level controls, workspace configuration controls, and a message window in the middle.

THE MAIN AREA OF THE WORKSPACE below the toolbar is further divided into three areas.

The toolbar and the editor area are always present. The navigation area and the utility area are collapsable, i.e., they can be hidden away to make more room for the editor area when needed.

- The *navigation area* is located at the left side of the workspace. It contains the *Project Navigator*, which allows you to quickly access the components in your project. Additionally, the navigation area also contains several other overlapping views that are useful for various tasks of the project.
- The *editor area* is located in the center of the workspace. It is occupied by an editor for editing the contents of the component currently selected in the *Project Navigator*. The editor area can also be split into a side-by-side view with the main editor and an assistent editor.
- The *utility area* is located at the right side of the workspace. It contains a number of useful *inspectors* and *libraries* that are often used in conjunction with the editors.

Let's take a closer look at the main area of the Xcode workspace.

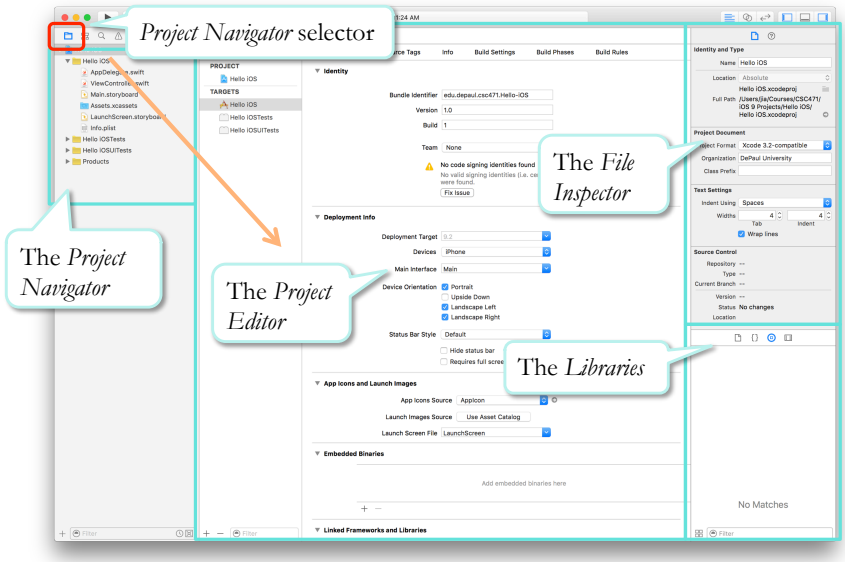


Figure 7: The *Project Navigator* and *Project Editor* in the workspace.

The most useful view in the navigation area is the *Project Navigator*, which provides a hierarchical overview of all the components in your project. It allows you to navigate and quickly select components in the project. When a component is selected, an editor that is capable of editing the contents of the selected component will be shown in the editor area. For example, when you select the root component, which representes the project-level configurations, the *Project Editor* appears in the editor area.

The utility area on the right side is vertically divided into an upper and a lower compartment.

- The upper compartment contains a number of overlapping *inspectors*, which usually display some detail information about an element of interest in the editor.
- The lower compartment contains a number of overlapping *libraries*, which usually contain elements that can be inserted into the artifact being edited.

Editors that may appear in the editor area include:

- The *Project Editor*, which allows you to edit the project-level properties and configurations of your project, including build options, target architectures, and app entitlements, etc.
- The *Interface Builder*, which allows you to graphically create and edit user interface files, i.e., storyboards.
- The *Source Editor*, which allows you to edit the contents of various types of text files, such as Swift source files.

An Anatomy of an iOS Project

EVEN A SIMPLE IOS PROJECT is composed of quite many files. It includes all the essentail ingredients of iOS projects.

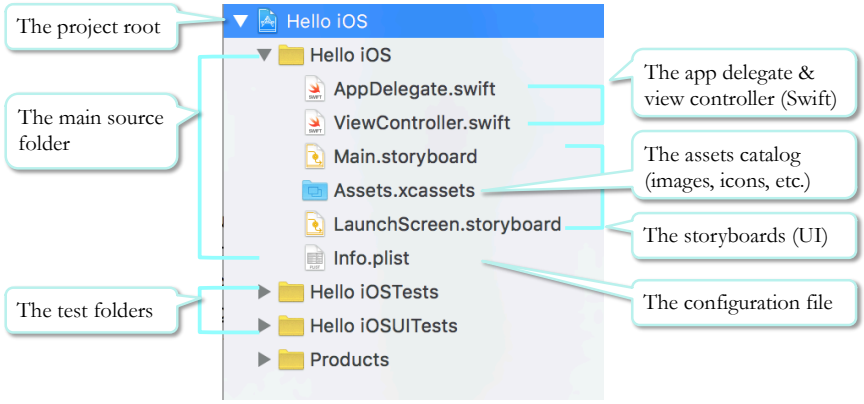


Figure 8: iOS project structure — the *Project Navigator* view

A simple project contains the following source folders:

- The *main source* folder, which has the same name as the project. It contains the source files of the project, including the user interface (UI) files, Swift source files, and resource files. These files are the main focus of the development.
- The *uint test* folder, which is named *Project Name*Tests and the *UI test* folder, which is named *Project Name*UITests. They contain the tests of your project.

Testing is a very important topic in mobile app development. Examples of unit tests and UI tests will be given later.

You will find several types of files in the main source folder.

- *.storyboard – The *Storyboard* files, which visually define the user interface of the application. A storyboard contains *scenes*, i.e., a screen or view, and *segues* that connect various scenes in the application.
- *.swift – The *Swift source* files, which define the key components and logic in applications. view controllers.
- *.xcassets – The *Asset Catalogs*, which manage the images and icons used by the application.
- *.plist – The *property list* files, which are commonly used for configurations and application data.

IT IS ALSO USEFUL TO TAKE A PEEK AT THE PROJECT under the file system view using the *Finder* on OS X.

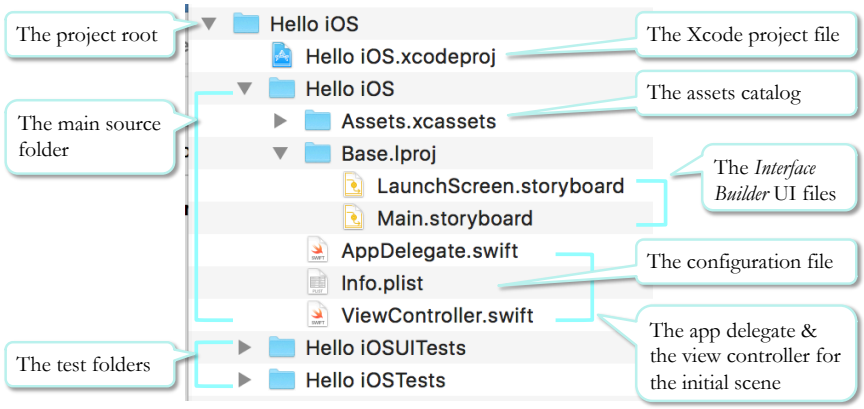


Figure 9: iOS project structure — the file system view

The structure under the project root folder is not identical to the structure in the *Project Navigator*. However, you will find the same items you see in the *Project Navigator* view in the file system view as well. The *Project Navigator* presents a *logical* view of the items in the project.

THE KEY PIECES IN A SIMPLE IOS APPLICATION are the following:

- The *Main* storyboard: Main.storyboard.

  The main storyboard captures the UI design of your application. A simple app typically contains a single main storyboard that captures the design of screens and the flows among screens application-wide. The storyboard files are edited visually using the *Interface Builder*.

  Starting Xcode 7, larger applications may divide the user interface design into several storyboards.

- The *Launch Screen* storyboard: LaunchScreen.storyboard.
  The *launch screen* storyboard describes the design of a screen that is displayed briefly while the app is being initialized during the initial launch, before the actual initial screen of the app is ready. It is provided to improve the *perceived performance* of the app.
- The *App Delegate*: AppDelegate.swift.
  For each application, there is always one *app delegate*. It is a Swift file generated from the template. For simple applications, there is no need to modify the the app delegate. It can be left untouched.
- The *View Controller*: ViewController.swift.
  This is the view controller associated with the only view of the single view application we have created. In general, each view controller is associated with a corresponding view or screen in the UI, and defines the logic and behavior of the view. An application may contain many view controllers.

The Interface Builder

EVERY APPLICATION HAS A *storyboard* named Main.storyboard, which visually defines the user interface of the views in the application and the flow of the screens. Let's jump right into the project by selecting the Main.storyboard from the Project Navigator. This will bring up in the editor area one of the most interesting component of Xcode – the *interface builder* – for editing the storyboard.

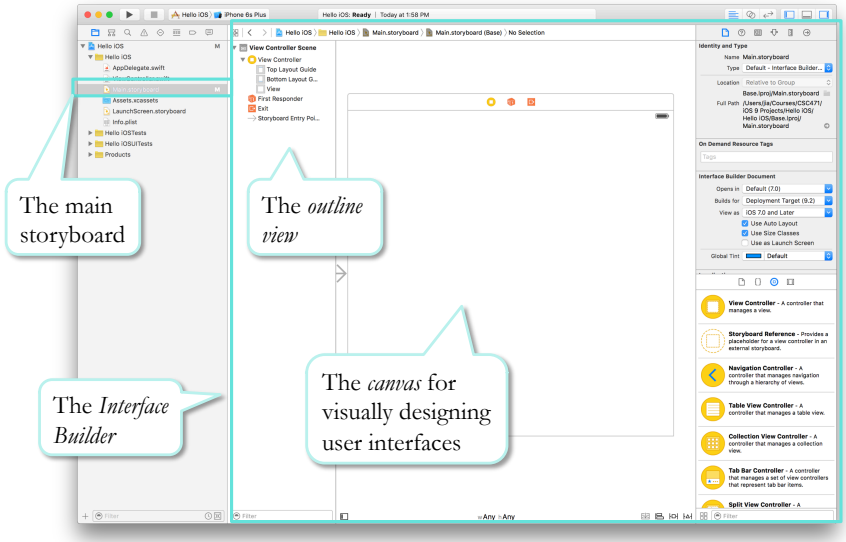


Figure 10: The *Interface Builder*

THE INTERFACE BUILDER (IB) IS A VISUAL EDITOR that allows you to design the user interface of your app *visually*. The interface builder divides the editor area into the *outline view* area on the left side, the *canvas* area that occupies most of the editor area for visual editing. The outline view is a hierarchical overview of all components in the storyboard. The canvas area shows the same components in the storyboard visually. The items in the outline view are selectable, and the item corresponding to the selected item will be highlighted in the canvas.

A STORYBOARD CONSISTS OF one or more *scenes*. Each *scene* represents the visual design of a unit of presentation in the user interface, typically a screen or a view. The main storyboard initially contains a single scene, which represents the first screen of your app.

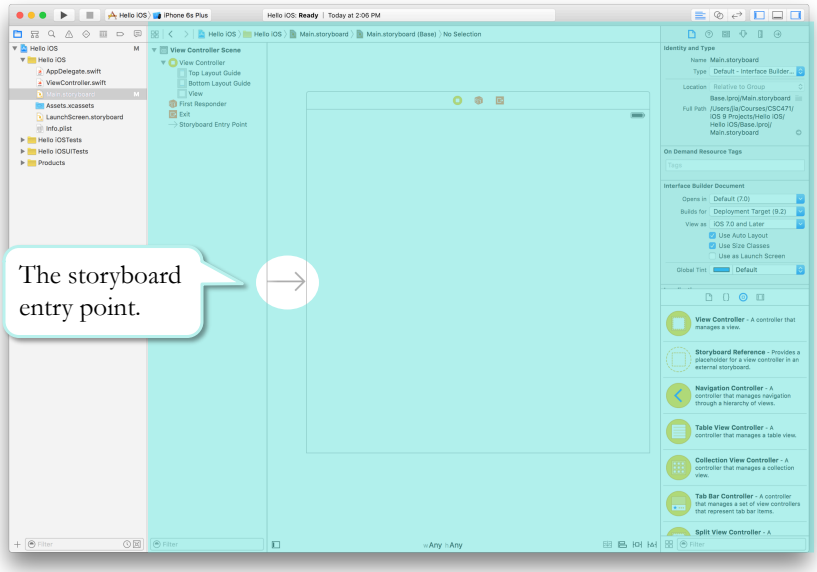


Figure 11: The *entry point* of the main storyboard.

THE ARROW AT THE LEFT SIDE OF THE SCENE is the *entry point* of the storyboard. Each storyboard can have only one entry point. The entry point arrow points to the scene that will appear first after the app is launched.

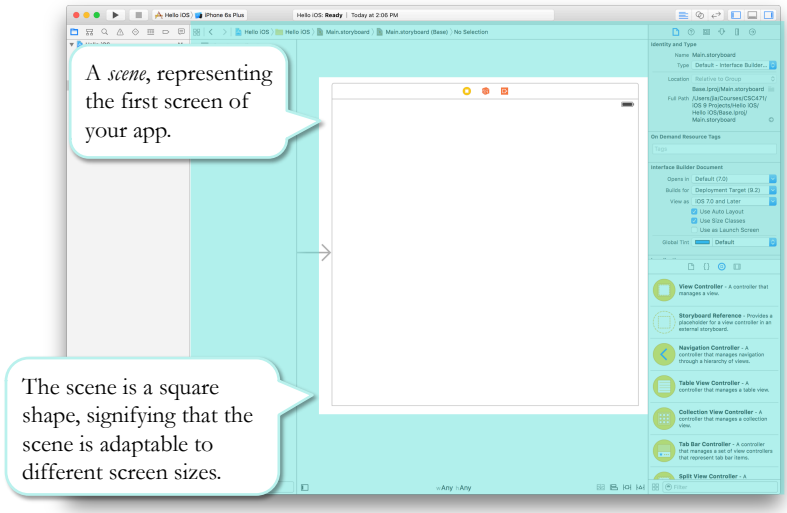


Figure 12: The *initial scene* of the main storyboard.

THE SCENE IN THIS STORYBOARD appears as a square shape, which does not correspond to the shape of the screen of any iOS device. This is the default shape of scenes in Xcode 6, to indicate that the scene is not designed for a particular type of devices, iPhone or iPad, rather it is intended to be *adaptable* to different screen sizes.

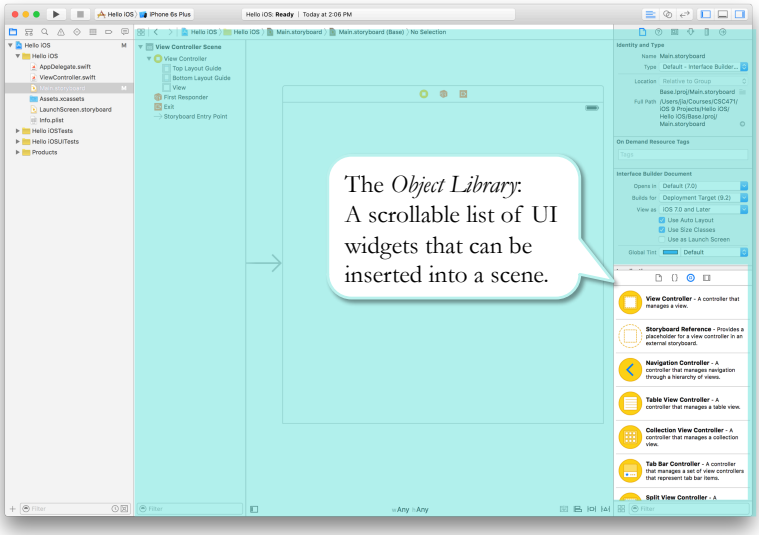


Figure 13: The *Object Library* of the Interface Builder.

THE *Object Library* at the lower compartment of the utility area contains a scrollable list of user interface widgets and other components that can be inserted into a scene. To insert a widget, select the widget the from the Object Library, and drag it into the scene.

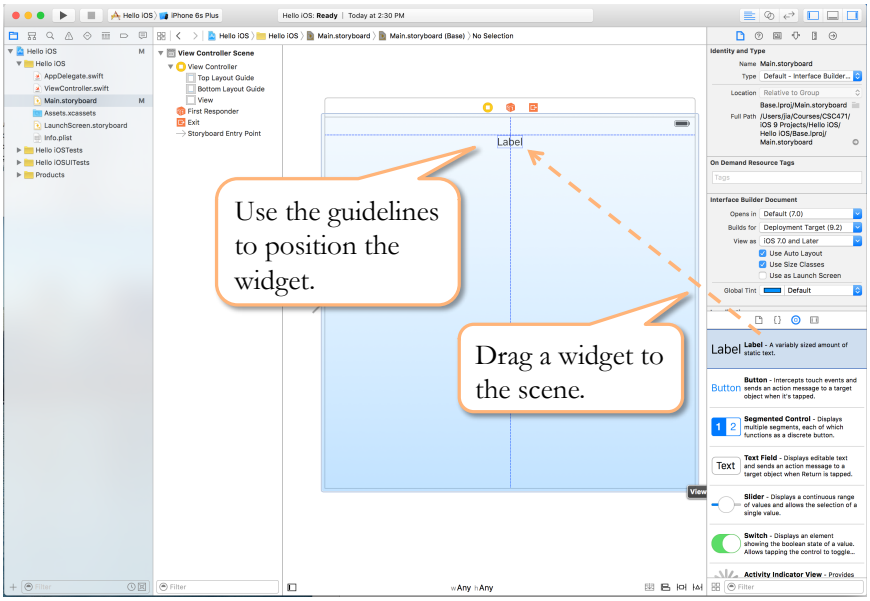
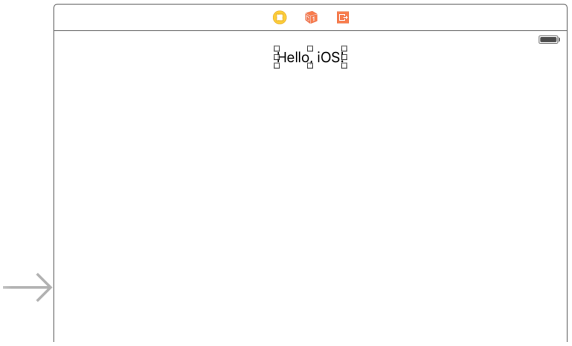


Figure 14: Inserting a *Label* widget into a *scene* in a storyboard.

Let's select a *Label*, and drag it to the scene. As we drag a widget over the scene, dashed blue guidelines will appear to assis the positioning of the widget. Let's use the guidelines to position the label just below the top guideline, i.e., just below the status bar, and centered it horizontally around the vertical center guideline.

A *Label* is one of the simplest UI widget from the Object Library. It contains a piece of static text.

Next, click the label in the scene to select it, and use the keyboard to type the text of the label – *Hello iOS*. The edited label will appear in place, WYSIWYG (what-you-see-is-what-you-get).



Running Apps in Simulator

NOW, LET'S GIVE THE APP A TEST RUN in an *iOS Simulator*, which is part of the iOS SDK. The controls for selecting the targeting device or simulator and running the app are located at the left side of the toolbar.

A project can be executed using different configurations. Each configuration for execution is knows as a *scheme*. To select a scheme, you need to select a *build target* and a *run destination*. The build target can be the *application* itself, the *unit test*, or the *UI test*. The run destination can be a device or a simulator. At the moment, we have only one target, the application itself. Let's select the desired simulator to run the app, using the drop-down menu (shown in Figure 15 with the options of the available iOS devices and simulators.). For example, we choose the simulator for *iPhone 6s*. Click the *Run* button. The simulator window appears shortly and the result is shown in Figure 16.

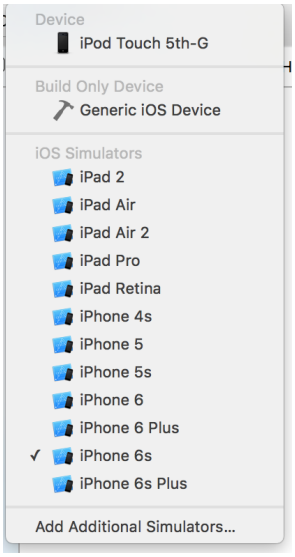


Figure 15: The options of *run destinations*: the available iOS devices and simulators

The unit test and UI test build targets can be added later.

YOU NOTICE THAT the appearance of the app running on an iPhone 6s is different from the design of the scene in our storyboard. The label *Hello, iOS!* appears just under the status bar but it is not centered horizontally. This is because Xcode supports and encourages the use of *auto layout*, which is a constraint based automatic mechanism that layout the screens based on the actual screen size of the device on which the app is running. Apps using auto layout will be able to automatically adapt to different screen sizes of iOS devices. This is why by default the scenes are square-shaped in storyboards, to indicate that the design is not specific to a particular dimension or shape of device screens. Xcode discourages the use of absolute positions and sizes in UI design.

Auto layout is a powerful but slightly complicated mechanism, which we will discuss in detail later. For now, let's keep things as simple as possible. We will not use auto layout for the first a few apps. Instead, we will design the UI for a specific size of iPhone screen only. Rest assured that we will use auto layout in our UI design very shortly.

Figure 16: The *Hello iOS* app running in the iPhone 6s simulator. Notice that the position of the label "Hello, iOS!" is off center.

Setting the Screen Size

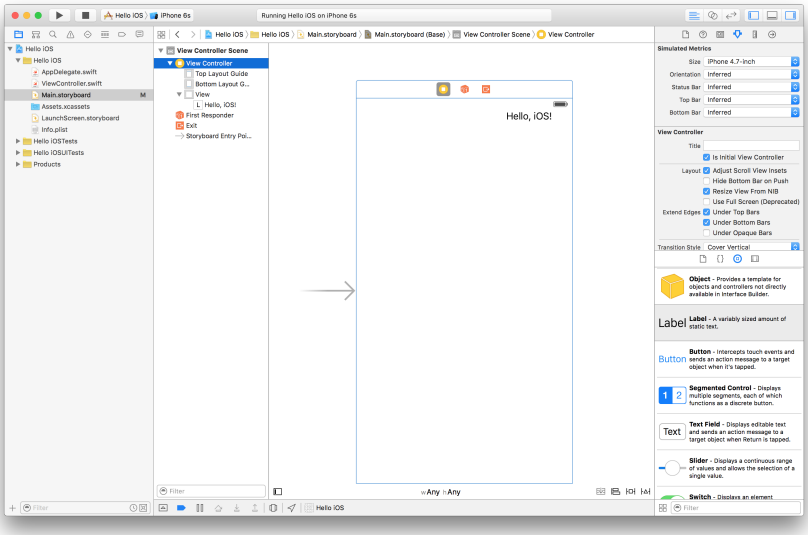
To design the UI for a specific size of iPhone screen only, we need to set the size of the scene in our storyboard to match the size of the screen. For this, we will use one of the most useful inspectors in Xcode – the *Attribute Inspector*. We first select the *View Controller* from the *outline view* panel of the storyboard. The *View Controller* represents a rectangular region occupying the entire screen area of an iPhone. We then use the *Attribute Inspector* selector at the top of the inspector panel to bring up the *Attribute Inspector*, as illustrated in Figure 17. The *Attribute Inspector* displays the attributes of the selected item in the storyboard, in this case the *View Controller*.

Figure 17: The *Attribute Inspector* of the *View Controller*.

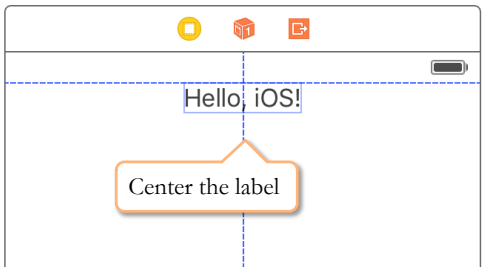
In the *Simulated Metrics* section of the *Attribute Inspector*, it lists a number of attributes all indicated as being *inferred*, which means the values of these attributes are determined bases on the metrics of the screen of the device running the app at runtime. Let's change the *Size* attribute from *inferred* to *iPhone 4.7-inch*. Now, we are designing the UI layout for a specific screen size, namely the 4.7 inch iPhone (iPhone 6 and 6s), only. This also means that our UI layout may look awful on iOS devices with other than 4.7 inch screens.

Figure 18: Set the *Size* attribute of the *View Controller*

After changing the screen size, the scene in the storyboard becomes a tall rectangular shape, which represents the dimension of the 4.7 inch screen of iPhone 6 and 6s.



Now you can reposition the label to center it horizontally.



Run the app again, and the result is shown in Figure 19.

Figure 19: The *Hello iOS* app running in iPhone 6s simulator. Notice that the label is perfectly centered horizontally.

Editing Attributes of Widgets

NOW LET'S TRY TO MAKE THE LABEL a little more interesting. We will try to change the font and the color of the label To set or change the attributes of a widget, we again use the *Attribute Inspector*. We first select the widgets in the storyboard. A widget can be selected directly from the scene or from the *outline view* panel. We then select the *Attribute Inspector*, which will display intuitive editable fields corresponding to the attributes of the selected widget (Figure 20).

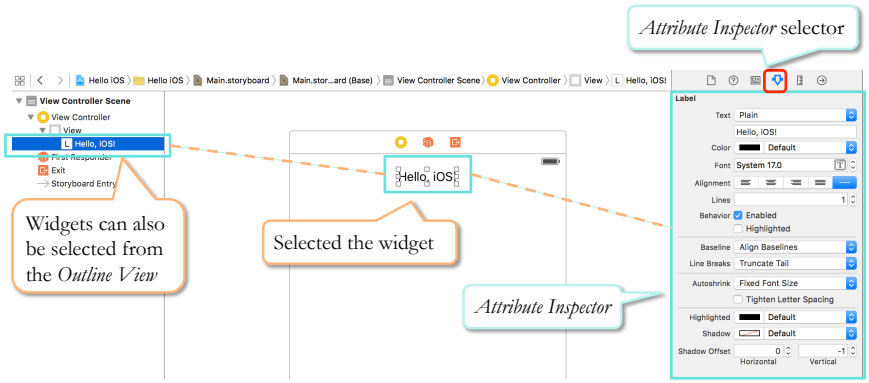
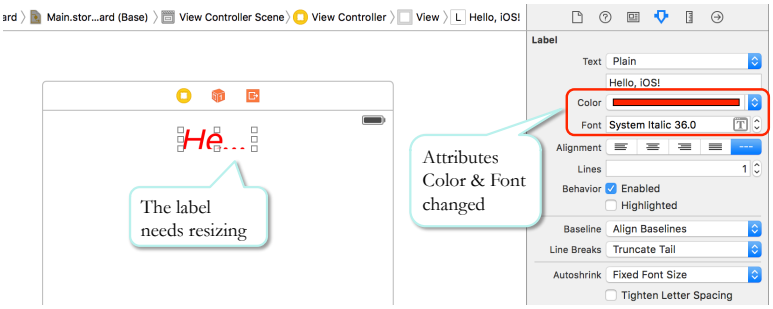


Figure 20: The *Attribute Inspector* for the Label widget.

Let's find the fields labeled *Color* and *Font* and change their values. You will notice that changes in the attributes of the widget are immediately reflected in the storyboard. In other words, the storyboard provides instant preview of any attribute changes.



After changing the font size, the text of the label no long fits the current size of the label. Now go back to the storyboard, resize the label and reposition it.

Run the app again, and the result is shown in Figure 21

Figure 21: The *Hello iOS* app running in the iPhone 6s simulator

App Icons

Finally, let's try to spruce up our *Hello iOS* app by adding an app icon. Every app has an app icon. If no app icon is specified, a default icon is used (see Figure 22).

The app icon appears in several contexts and in different dimensions, as shown in Figure 23 below:

- On the *Home* screen, dimension: 60×60 (points)
- In the results of the *Search* (formerly *Spotlight*), dimension: 40×40 (points)
- In the *Settings* app, dimension: 29×29 (points)

Notice that the dimensions of the icons are measured in *points* (pt), not in pixels. This is because the pixel densities, typically measured in *pixels-per-inch (ppi)*, of the screens of difference iOS device families, vary significantly. If we were to use images of identical dimensions in pixels on different iOS devices, their actual physical size could vary by a factor of 2-3. The *point* is introduced as a logical unit for measuring dimensions of on-screen objects, to ease the pain of dealing with different screen pixel densities and to maintain a relatively consistent physical dimension of images on devices with different screen pixel densities.

Figure 22: The *Home* screen shows the default placeholder app icon for *Hello iOS*

Figure 23: App icons on the *Home* screen (left), in the results of the *Search* (center), and in the *Settings* app (right).

Screen Resolutions of iOS Devices

To achieve the best results in appearance and performance when displaying images on screens, one must take into account the resolutions and pixel densities of screens of different iOS device families. An image of a given dimension in points will be rendered on different devices in different dimensions in pixels. The relationship between dimensions in points and dimensions in pixels for a given device is known as the *scale factor* of that device. Table 1 and Table 2 summarize the the sizes, resolutions, pixel densities, and scale factors of screens of all major iOS device families.

An icon or an image should be the same size in points across all devices, which means the sizes will be different in pixels. Although iOS will scale an image at runtime if necessary, for best results in appearance and performance, you should provide icons and images in different pixel sizes for all scale factors of the devices you intend to support.

For example, if you intend to support devices running iOS 7 - 9, and your app is designed for iPhone only, then you need to supply images for scale factors 2x and 3x. If your app is designed for iPhone and iPad, then you need to supply images for scale factors 1x, 2x, and 3x.

Devices that are compatible with iOS 9:

- iPhone 4, 4s, 5, 5c, 5s, 6, 6 plus, 6s, 6s plus
- iPad 2, 3, 4
- iPad Air, Air 2
- iPad Mini, 2, 3, 4
- iPad Pro
- iPod Touch 5th, 6th generation

Devices that are compatible with iOS 8:

- iPhone 4, 4s, 5, 5c, 5s, 6, 6 plus
- iPad 2, 3, 4
- iPad Air, Air 2
- iPad Mini, 2, 3, 4
- iPod Touch 5th generation

Devices that are compatible with iOS 7:

- iPhone 4, 4s, 5, 5c, 5s
- iPad 2, 3, 4
- iPad Air
- iPad Mini, 2, 3, 4
- iPod Touch 5th generation

	iPhone iPhone 3G, 3GS	iPhone 4, 4s	iPhone 5, 5c, 5s iPod Touch 5/6th gen	iPhon 6, 6s	iPhone 6 plus iPhone 6s plus
Size (diagonal)	3.5″	3.5″	4″	4.7″	5.5″
Display type	Classic	Retina	Retina	Retina (HD)	Retina HD
Pixels (px)	320×480	640×960	640×1136	750×1,334	1,080×1,920
Density (ppi)	163	326	326	326	401
Points (pt)	320×480	320×480	320×568	375×667	414×736
Aspect ratio	3:2	3:2	~16:9	16:9	16: 9
Scale factor	1x	2x	2x	2x	3x

Table 1: The screen sizes and resolutions of iPhones and iPod Touch

	iPad iPad 2	iPad 3, 4 * iPad Air, Air 2	iPad Mini	iPad Mini 2,3,4 *	iPad Pro
Size (diagonal)	9.7″	9.7″	7.9″	7.9″	12.9″
Display type	Classic	Retina	Classic	Retina	Retina
Pixels (px)	768×1,024	1,536×2,048	768×1,024	1,536×2,048	2,048×2,732
Density (ppi)	132	264	163	326	264
Points (pt)	768×1,024	768×1,024	768×1,024	768×1,024	1,024×1,366
Aspect ratio	4:3	4:3	4:3	4:3	4:3
Scale factor	1x	2x	1x	2x	2x

Table 2: The screen sizes and resolutions of iPads (*: with Retina Display)

Assest Catalogs

A TYPICAL MOBILE APP contains many images and icons. It is a tedious task to manually organize and manage multiple image files of different resolutions for each image. Xcode uses a tool known as *Asset Catalogs* to simplify the management of icons and images. Each asset in a catalog consists of

- a *name*, which is used in your app to refer to the asset.
- a *set of images* of different dimensions (in pixels).

At runtime, iOS will load the image from the set that is most appropriate for the scale factor of the current device. An image will be scaled if necessary. However, this would incur a runtime cost and result in a reduced quality of the image.

To use the *Asset Catalogs* to manage the app icon, you should first prepare a set of images of the app icon for the different contexts and scale factors[3]. Assume that our app is designed for iPhone only, we need to support scale factors of 2x and 3x. Images of the app icon in the following resolutions (in pixels) will be needed:

- For the *Home* screen (60pt): 120×120 (2x), 180×180 (3x)
- For the *Search* results (40pt): 80×80 (2x), 120×120 (3x)

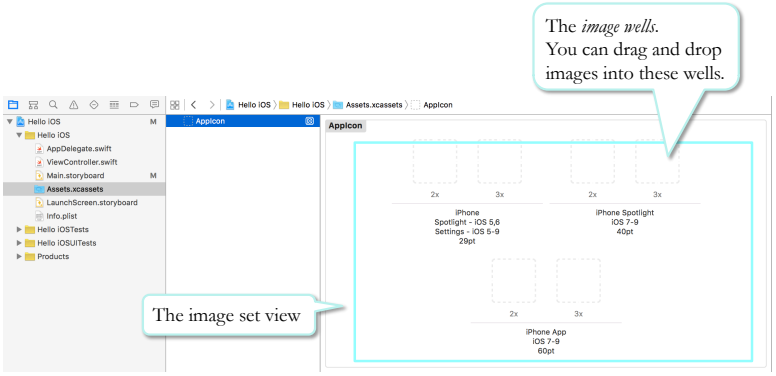
[3] It is recommended to use the PNG (Portable Network Graphics) format for the images. Xcode also supports vector PDF format.

- For the *Settings* app (29pt): 58×58 (2x), 87×87 (3x)

An asset named *AppIcon* already exists in the *Asset Catalog* of the project. To open the *Asset Catalog*, from the *Project Navigator*,

- Select the item labeled Assets.xcassets

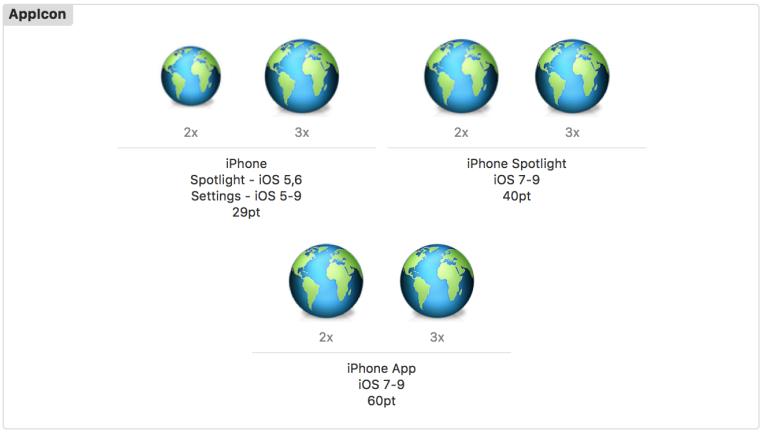
The *Asset Catalog* editor appears in the editor area:



From the overview panel of the *Asset Catalog* editor

- Select the item named AppIcon, the asset for the app icon.

The image set view of the *Asset Catalog* editor shows a number of empty *image wells* initially, indicating no image has been specified. You can drag and drop image files from a *Finder* window to the image wells.



After filling all the image wells with images of appropriate resolutions, run the app again. Then return to the *Home* screen of the simulator[4]. You will see the updated app icon of the *Hello iOS* app (Figure 24).

Figure 24: The Home screen of the iPhone 6s simulator with the icon of the *Hello iOS* app.

[4] To show the *Home* screen of the simulator, from the menu bar
Hardware ▶ Home
Or use keyboard shortcut: ⇧⌘H
This will simulate the pressing of the *Home* button on the simulator.