# iOS®

# Developer

## Notes for Professionals

### Chapter 13: Cut a UIImage into a circle

### Chapter 4: attributedText in UILabel

### Chapter 107: Push Notifications

## 800+ pages

of professional hints and tricks

# Contents

# About

# Chapter 1: Getting started with iOS

| Version | Release Date |
|---|---|
| iPhone OS 2 | 2008-07-11 |
| iPhone OS 3 | 2009-06-17 |
| iOS 4 | 2010-06-08 |
| iOS 5 | 2011-10-12 |
| iOS 6 | 2012-09-19 |
| iOS 7 | 2013-09-18 |
| iOS 8 | 2014-09-17 |
| iOS 8.1 | 2014-10-20 |
| iOS 8.2 | 2015-03-09 |
| iOS 8.3 | 2015-04-08 |
| iOS 8.4 | 2015-06-30 |
| iOS 9 | 2015-09-16 |
| iOS 9.1 | 2015-10-22 |
| iOS 9.2 | 2015-12-08 |
| iOS 9.3 | 2016-03-21 |
| iOS 10.0.1 | 2016-09-13 |
| iOS 10.1 | 2016-10-24 |
| iOS 10.2 | 2016-12-12 |
| iOS 10.2.1 | 2017-01-23 |
| iOS 10.3 | 2017-03-27 |
| iOS 10.3.3 | 2017-07-19 |

## Section 1.1: Creating a default Single View Application

To develop an application for iOS, you should start with an application called Xcode. There are other alternative tools you can use, but Xcode is Apple's official tool. Note, however, that it only runs on macOS. The latest official version is Xcode 8.3.3 with Xcode 9 (currently in beta) due to be released later this year.

1. Boot up your Mac and install Xcode from the App Store if it's not already installed.

   (If you prefer not to use the App Store or have problems, you can also download Xcode from the Apple Developer website, but make sure that you select the latest release version and **not** a beta version.)

   

2. Open Xcode. The following window will open:
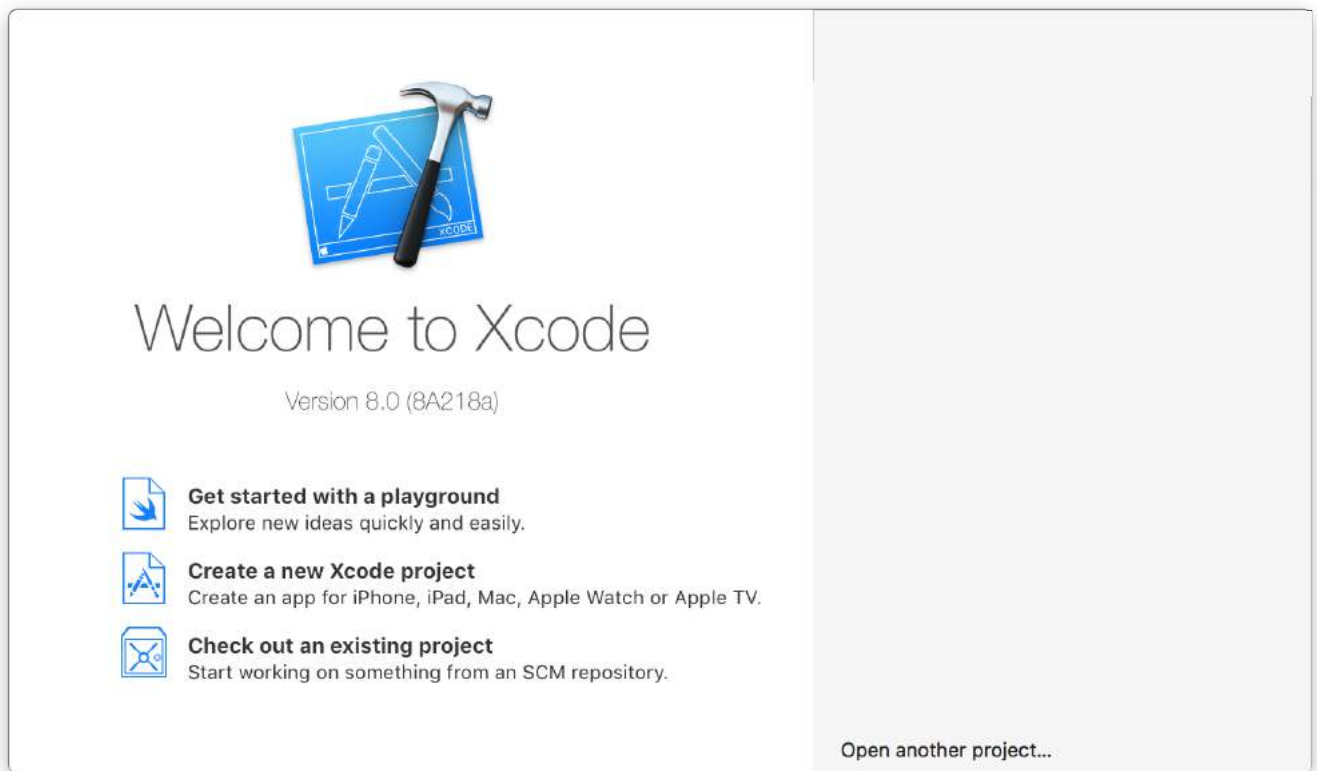
The window presents you with the following options:

- **Getting started with a playground:** This was introduced with the Swift language and Xcode 6. It's an interactive area which can be used to write small pieces of code to check runtime changes. It's a great way for Swift learners to be introduced to new Swift features.
- **Create a new Xcode project:** *Choose this option*, which creates a new project with default configuration.
- **Check out an existing project:** This is used to check out a project from a repository location, for example, check out a project from SVN.

3. Select the second option **Create a new Xcode project** and Xcode will ask you to do some initial project setup:
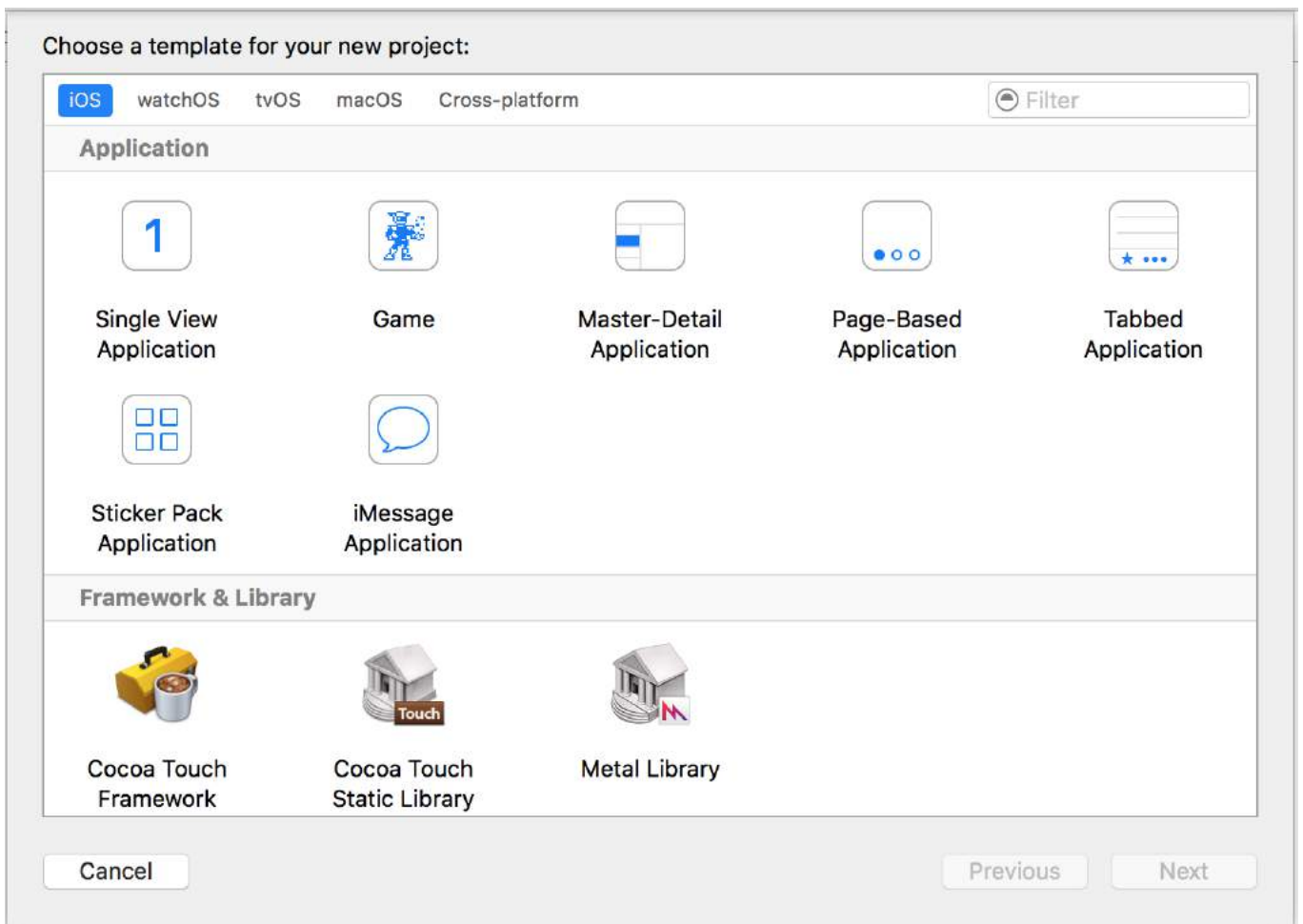
This wizard is used to select your project template. There are 5 options:

- **iOS:** Used to create iOS apps, libraries and frameworks
- **watchOS:** Used to create watchOS apps, libraries and frameworks
- **tvOS:** Used to create tvOS apps, libraries and frameworks
- **macOS:** Used to create macOS apps, libraries, frameworks, packages, AppleScripts, etc.
- **Cross-platform:** Used to create cross-platform apps, templates and In-App Purchase Contents

You can see that there are many different templates for your application. These templates are helpful to boost your development; they are pre-built with some basic project setups like UI interfaces and class files.

Here, we'll use the first option, **iOS**.

1. Master-Detail Application:

   This template contains a combined master and detail interface: the master contains objects which are related to the detail interface. Selecting objects in the master will change the details interface. You can see this kind UI in the Settings, Notes and Contacts applications on the iPad.

2. Page-Based Application:

   This template is used to create the page-based application. Pages are different views held by one container.

3. Single View Application:

This is a normal application development template. This is good for beginners to learn application flow.
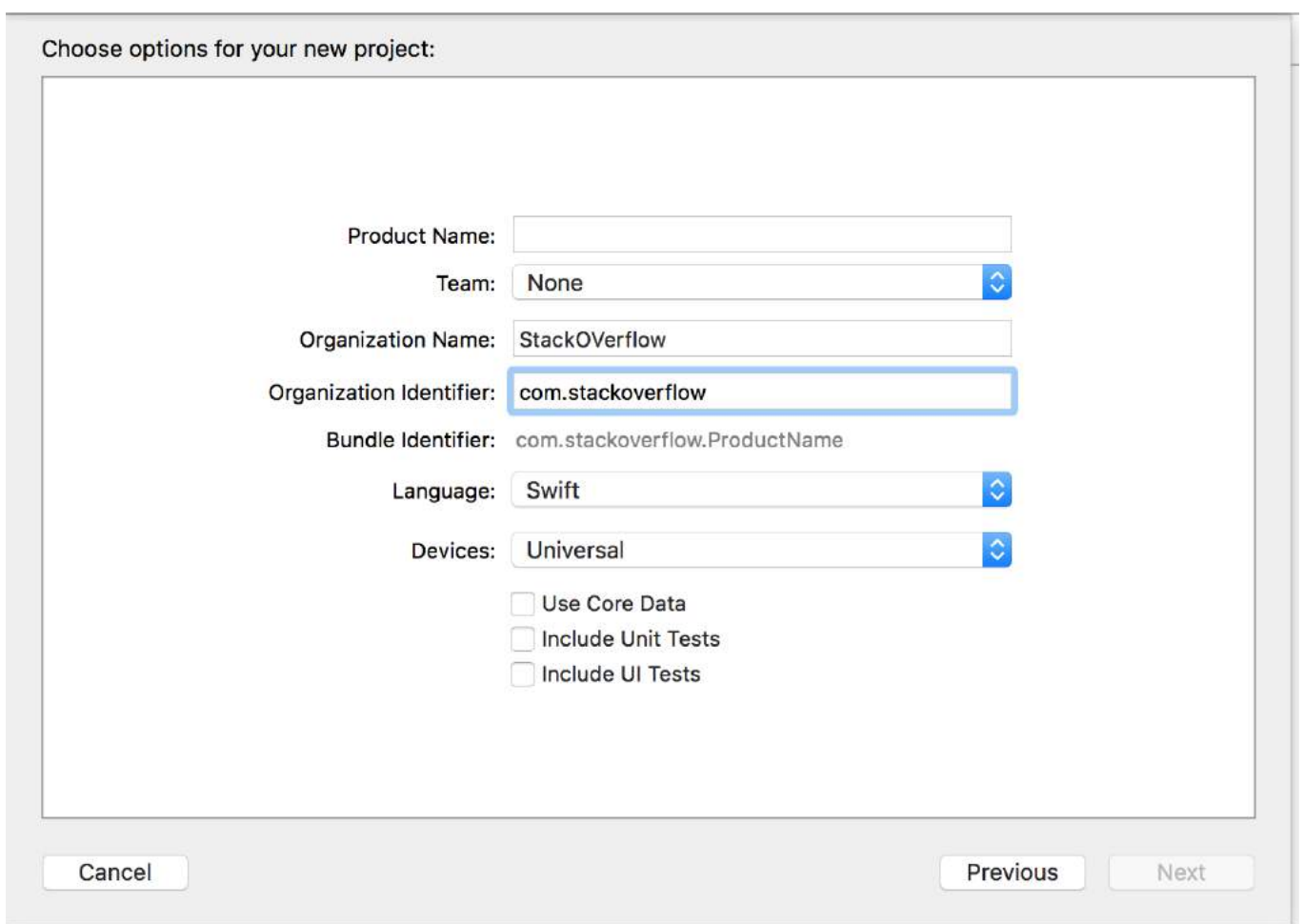
4. Tabbed Application:

   This template creates tabs at the bottom part of an application. Each tab has a different UI and a different navigation flow. You can see this template used in apps like Clock, iTunes Store, iBooks and App Store.

5. Game:

   This is a starting point for game development. You can go further with game technologies like SceneKit, SpriteKit, OpenGL ES and Metal.

4. In this example, we will start with **Single View Application**



The wizard helps you to define project properties:

- **Product Name:** The name of the project / application
- **Organization Name:** The name of the organization in which you are involved
- **Organization Identifier:** The unique organization identifier which is used in the bundle identifier. It is recommended to follow reverse domain name service notation.
- **Bundle Identifier:** *This field is very important.* It is based on your project name and organization identifier, choose wisely. The bundle identifier will be used in the future to install the application on a device and upload the app to iTunes Connect (which is the place we upload apps to be published on the App Store). It's a unique key to identify your application.

- **Language:** The programming language which you would like to use. Here you can change Objective-C to Swift if it's not selected.
- **Devices:** Supported devices for your application that can be changed later. It shows iPhone, iPad, and Universal. Universal applications support iPhone and iPad devices, and it's recommended to select this option when it's not necessary to run the app on only one kind of device.
- **Use Core Data:** If you would like to use Core Data Model in your project then mark it as selected, and it will create a file for the `.xcdatamodel`. You can also add this file later on if you don't know in advance.
- **Include Unit Tests:** This configures the unit test target and creates classes for unit testing
- **Include UI test:** This configures the UI test target and creates classes for UI testing

Click on **Next** and it will ask you for a location where you want to create project directory.
Click on **Create** and you will see the Xcode UI with an already defined project setup. You can see some classes and Storyboard files.

This is a basic template for a Single View Application.

At the top left of the window, check that a simulator is selected (e.g. "iPhone 6" as shown here) and then press the triangular RUN button.



5. A new application will open—Simulator (this may take some time the first time you run it and you may need to try twice if you see an error the first time). This application provides us with device simulation for created applications. It almost looks like a real device! It contains some applications like a real device. You can simulate orientations, location, shake gesture, memory warnings, In-Call Status bar, finger touch, lock, reboot, home etc.

You will see a plain white application because we have not made any changes to the template yet.

So start your own. it's a long run and there are lots of new opportunities waiting for you!

If you are not sure where to go next, try out Apple's '[Jump Right In](#)' tutorial. You have already performed the first few steps so are off to a head start.

# Section 1.2: Hello World

After setting up Xcode, it is not difficult to get your first iOS up and running.
In the following example we will:

- Start a new project
- Add a label
- Printing message to console.
- Run in the simulator

**Starting a new project**

When the Xcode welcome screen comes up, choose **Create a new Xcode project**. Alternatively, you could do **File >**

**New > Project...** from the Xcode menu if you already have it open.
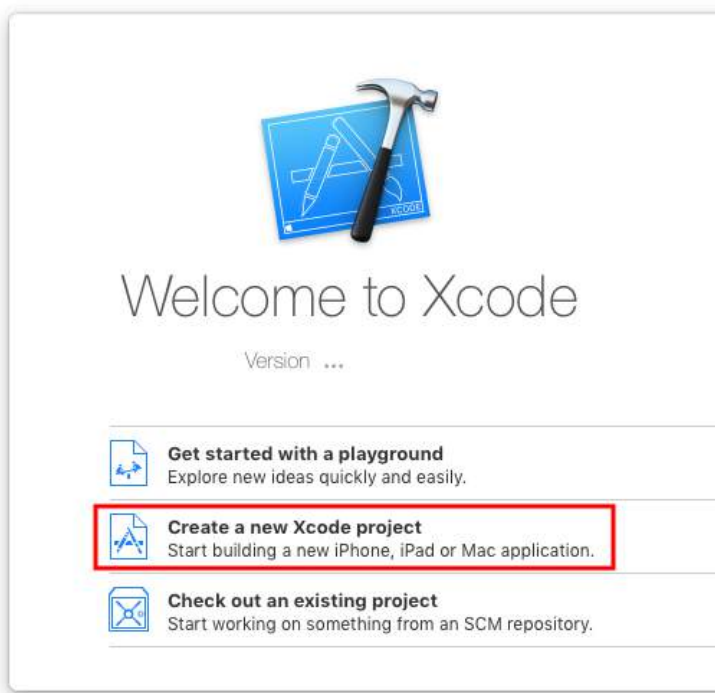


Choose a **Single View Application** and click **Next**.
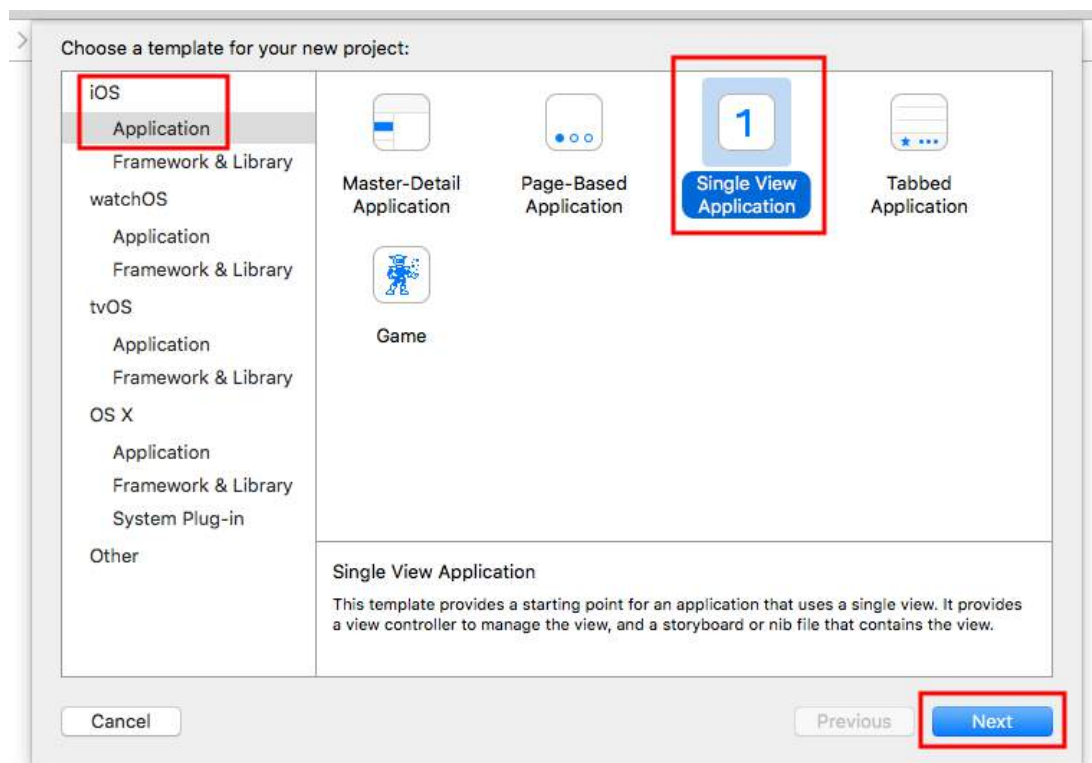


Write "HelloWorld" for the **Product Name** (or whatever you want really) and under **Language**, make sure **Swift** is selected.

- **Universal** means that your app will run on both the iPhone and iPad.
- **Use Core Data** refers to persistent data storage, which is not needed in our Hello World app.
- We will not be doing **Unit Tests** or **UI Tests** in this example, but it doesn't hurt to get into the habit of adding them.
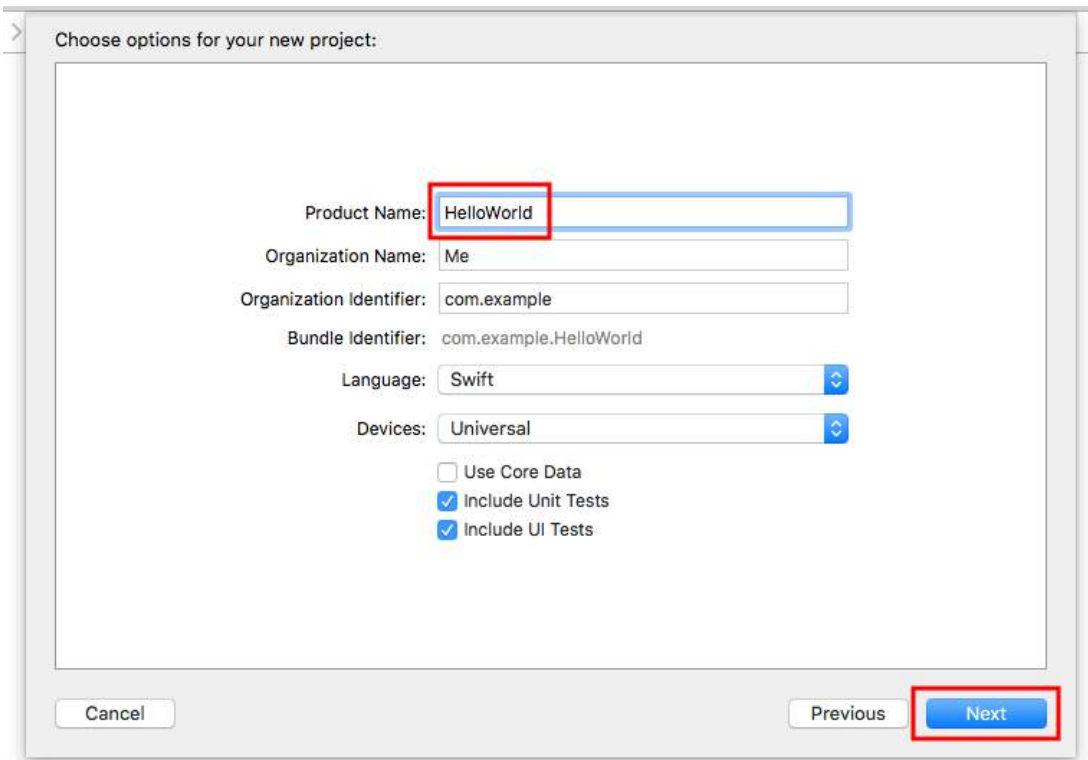
Choose an existing folder or create a new one where you will save your Xcode projects. This will be the default in the future. We created one here called "Xcode Projects". Then click **Create**. You can select Source Control if you like (used when syncing to sites like [GitHub](#)), but we won't be needing it in this example.
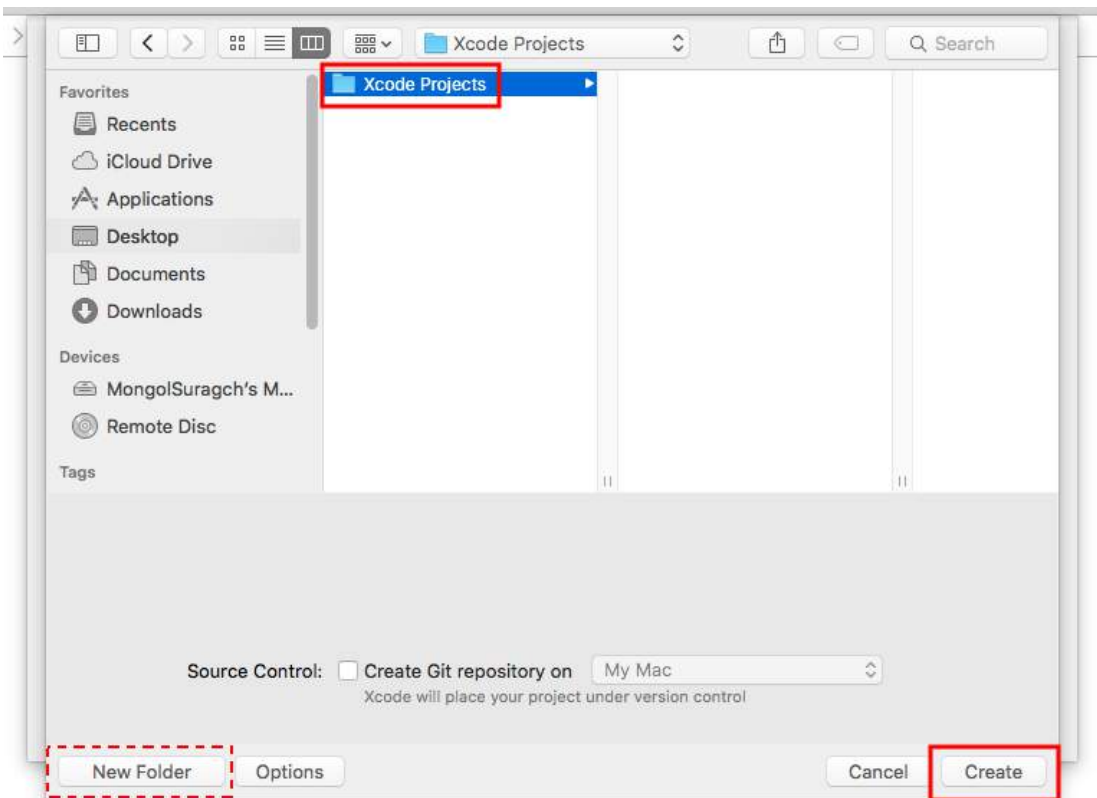


**Adding a label**

This is the file structure of an Xcode project.

Select *Main.storyboard* in the Project Navigator.
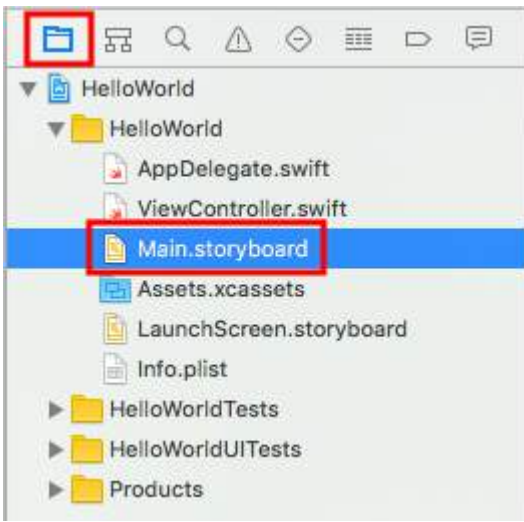
Type "label" in the search field of the Object Library in the bottom right of Xcode. Then drag the `UILabel` onto the storyboard View Controller. Place it generally in the region of the top left corner.
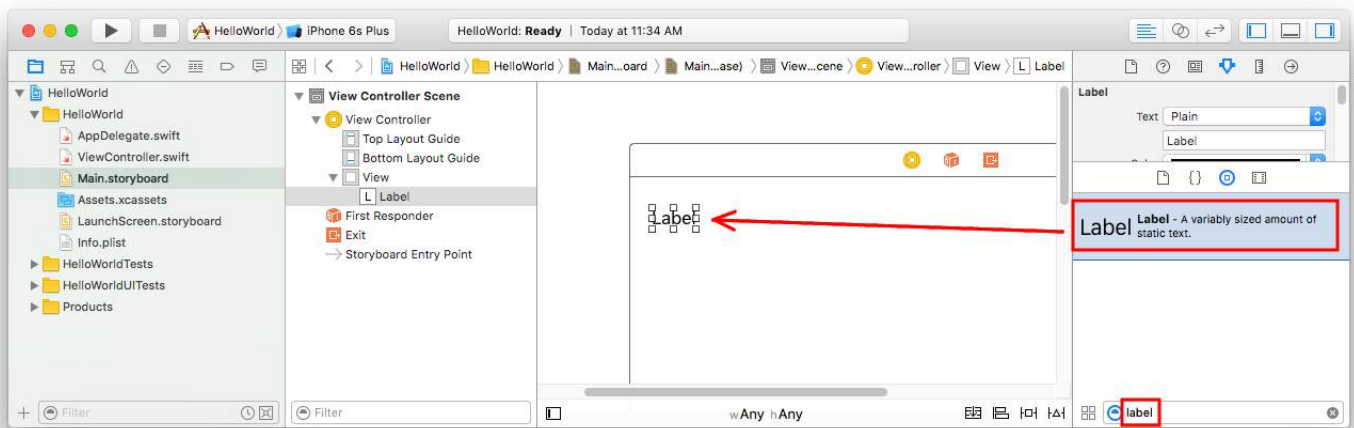


Make sure the label is selected on the storyboard and then in the **Attributes Inspector**, change the text to "Hello, World!" You will then have to resize and reposition the label on the storyboard since the text length is longer now.
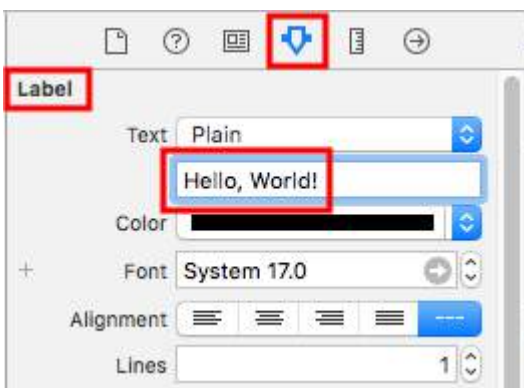


Alternatively, double-click the label on the storyboard to edit it to be "Hello, World!". At any rate, the storyboard should look something like this:



**Adding Code**

Select *ViewController.swift* in the Project Navigator.
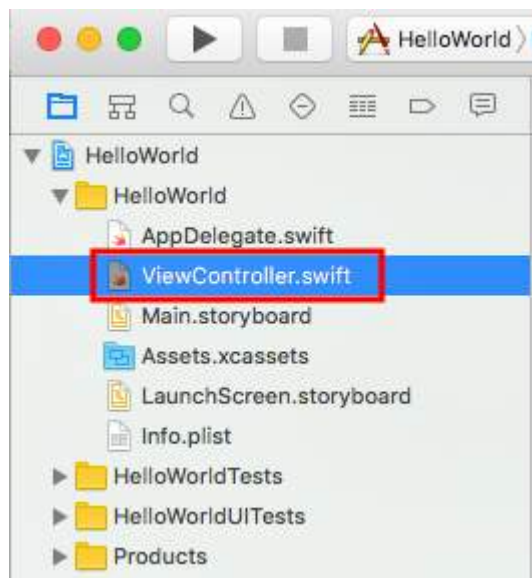


Add `print("Successfully created my first iOS application.")` to the `viewDidLoad()` method. It should look something like this.

```swift
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // print to the console when app is run
        print("Successfully created my first iOS application.")
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

**Running the app in the simulator**



Press the Run button to build and run the app. In this example the current simulator device (referred to as a "scheme") defaulted to the iPhone 6s Plus. Newer versions of Xcode will default to newer schemes. You can also choose other schemes by clicking the name. We will just stick with the default.

The simulator will take some time to start on the first run. Once running, it should look like this:



In the simulator menu, you can choose **Window > Scale** to make it smaller, or press ?cmd + 1/2/3/4/5 for 100% /

75% / 50% / 33% / 25% scale respectively..

The Xcode debug area (at the bottom) should have also printed "Successfully created my first iOS application." to the console. "Successfully created my first iOS application." message is the string you printed programmatically in the **Add code** part.



**Going on**

You should learn about Auto Layout constraints next. These help you to position your controls on the storyboard so that they look good on any device size and orientation.

# Section 1.3: Xcode Interface

In the Xcode, you have three separate areas of working - Navigators (in red), Debug area(in green) and Utilities(in blue).
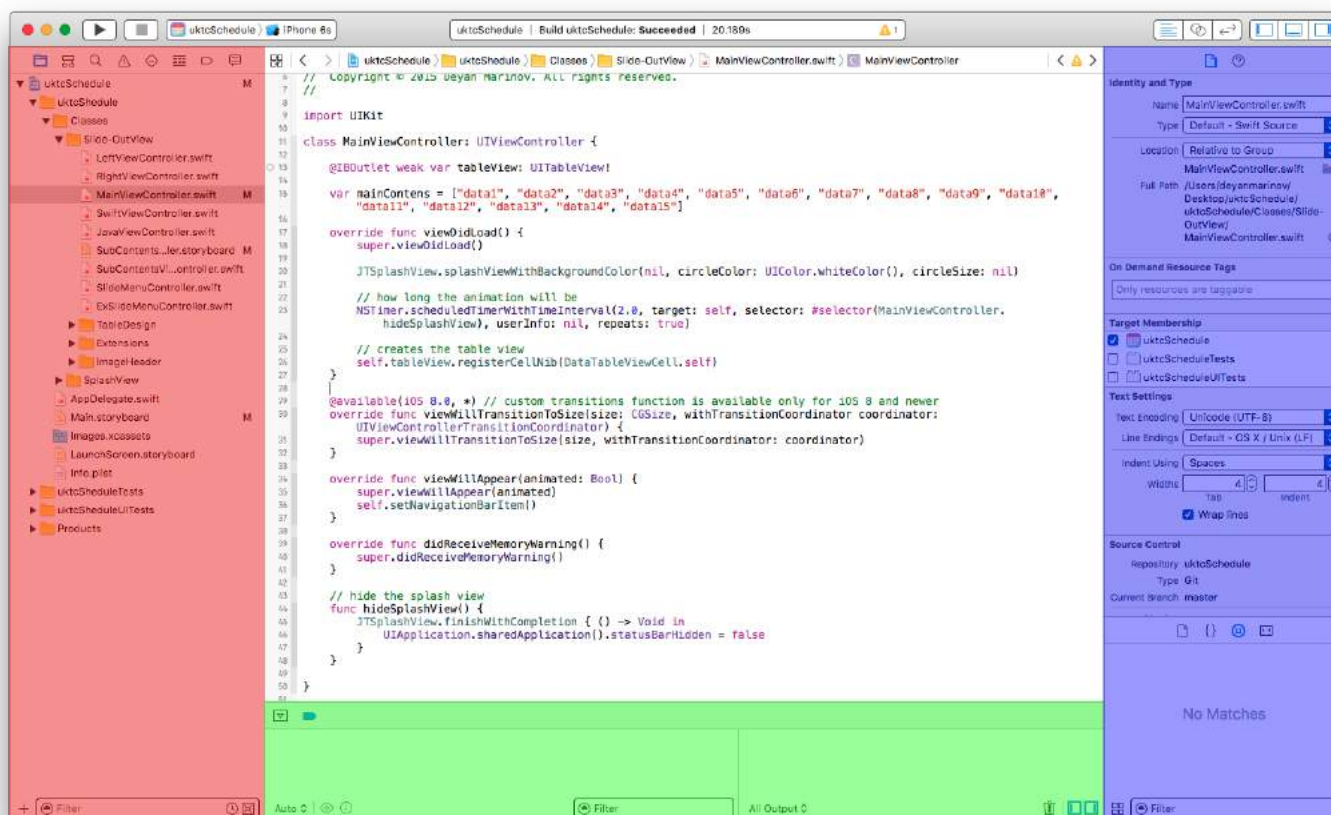


The workspace window always includes the editor area. When you select a file in your project, its contents appear in the editor area, where Xcode opens the file in an appropriate editor. For example, in the image above, the editor area MainViewController.swift, a swift code file that is selected in the Navigator area on the left of the workspace

window.

**Navigator Area**
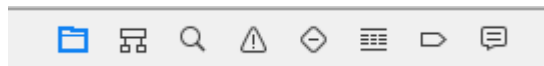
The navigator window contains the following eight options:

- **Project navigator.** Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.
- **Symbol navigator.** Browse the symbols in your project as a list or hierarchy. Buttons on the left of the filter bar let you limit the shown symbols to a combination of only classes and protocols, only symbols in your project, or only containers.
- **Find navigator** Use search options and filters to quickly find any string within your project.
- **Issue navigator.** View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.
- **Test navigator.** Create, manage, run, and review unit tests.
- **Debug navigator.** Examine the running threads and associated stack information at a specified point or time during program execution.
- **Breakpoint navigator.** Fine-tune breakpoints by specifying characteristics such as triggering conditions.
- **Report navigator.** View the history of your build, run, debug, continuous integration, and source control tasks.

**The Editors**

Most development work in Xcode occurs in the editor area, the main area that is always visible within the workspace window. The editors you use most often are:

- **Source editor.** Write and edit source code.

```swift
//
//  LeftViewController.swift
//  uktcShedule
//
//  Created by Deyan Marinov on 10/9/15.
//  Copyright © 2015 Deyan Marinov. All rights reserved.
//

import UIKit

enum LeftMenu: Int {
    case Main = 0
    case Swift
    case Java
}

protocol LeftMenuProtocol : class {
    func changeViewController(menu: LeftMenu)
}

class LeftViewController : UIViewController, LeftMenuProtocol {

    @IBOutlet weak var tableView: UITableView!
    var menus = ["Main", "Swift", "Java"]
    var mainViewController: UIViewController!
    var swiftViewController: UIViewController!
    var javaViewController: UIViewController!
    var goViewController: UIViewController!
    var nonMenuViewController: UIViewController!
    var imageHeaderView: ImageHeaderView!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.tableView.separatorColor = UIColor(red: 224/255, green: 224/255, blue: 224/255, alpha: 1.0)

        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let swiftViewController = storyboard.instantiateViewControllerWithIdentifier("SwiftViewController") as!
            SwiftViewController
        self.swiftViewController = UINavigationController(rootViewController: swiftViewController)

        let javaViewController = storyboard.instantiateViewControllerWithIdentifier("JavaViewController") as!
            JavaViewController
        self.javaViewController = UINavigationController(rootViewController: javaViewController)
```

- **Interface Builder.** Graphically create and edit user interface files.

▼ 🗔 **Right View Controller Scene**
  ▶ 🟠 Right View Controller
    📦 First Responder
    📤 Exit

▼ 🗔 **Left View Controller Scene**
  ▶ 🟠 Left View Controller
    📦 First Responder
    📤 Exit

▼ 🗔 **Main View Controller Scene**
  ▶ 🟠 Main View Controller
    📦 First Responder
    📤 Exit
    → Storyboard Entry Point

▼ 🗔 **Swift View Controller Scene**
  ▶ 🟠 Swift View Controller
    📦 First Responder
    📤 Exit

▶ 🗔 **Java View Controller Scene**

▶ 🗔 **Non Menu Controller Scene**

Table View
Prototype Content

Table View
Prototype Content

Java View Controller

JavaViewController

Non Menu Controller

NonMenuViewController

To MainViewController

Swift View Controller

⊙ Filter                                        ▢            wAny hAny              ▦ ⊟ ⊢ ⊣

- **Project editor.** View and edit how your apps should be built, such as by specifying build options, target architectures, and app entitlements.
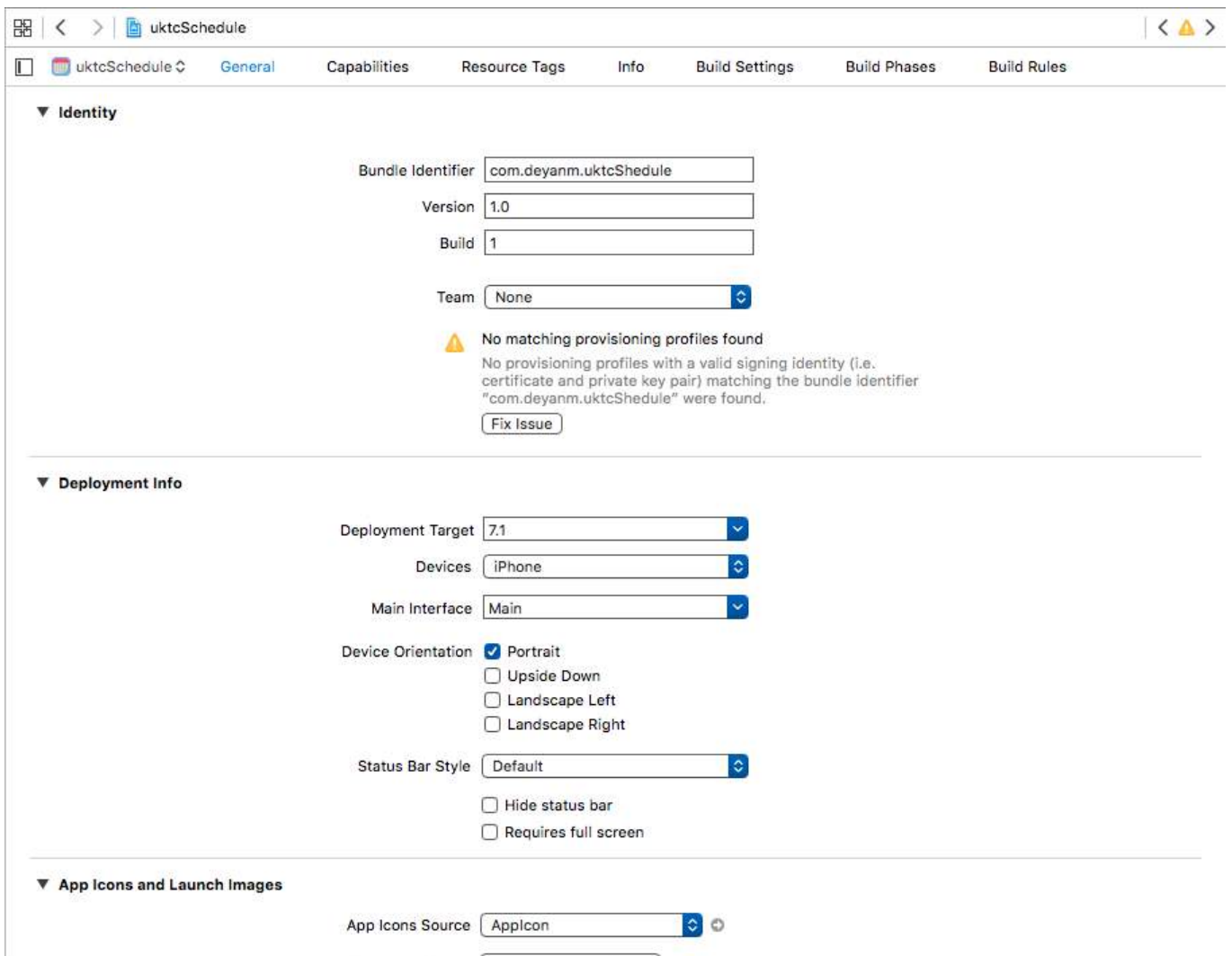
Configure the editor area for a given task with the editor configuration buttons on the right side of the toolbar:



- **Standard Editor.** Fills the editor area with the contents of the selected file.
- **Assistant Editor.** Presents a separate editor pane with content logically related to content in the standard editor pane. You can also change the content.
- **Version Editor.** Shows the differences between the selected file in one pane and another version of that same file in a second pane. This editor works only when your project is under source control.

**Resources and Elements in Utilities Area**

The utilities area on the far right of the workspace window gives you quick access to these resources: Inspectors, for viewing and modifying characteristics of the file open in an editor Libraries of ready-made resources for use in your project

**The top panel of the utilities area displays inspectors. The bottom pane gives you access to libraries.**

Simulated Metrics

| | |
|---|---|
| Size | Inferred |
| Orientation | Inferred |
| Status Bar | Inferred |
| Top Bar | Inferred |
| Bottom Bar | Inferred |

View Controller

Title

☑ Is Initial View Controller

Layout ☑ Adjust Scroll View Insets
☐ Hide Bottom Bar on Push
☑ Resize View From NIB
☐ Use Full Screen (Deprecated)
Extend Edges ☑ Under Top Bars
☑ Under Bottom Bars
☐ Under Opaque Bars

Transition Style  Cover Vertical
Presentation  Full Screen
☐ Defines Context
☐ Provides Context
Content Size ☐ Use Preferred Explicit Size

| 600 | 600 |
|---|---|
| Width | Height |

Key Commands

+ —



🟡 **View Controller** - A controller that manages a view.

⬜ **Storyboard Reference** - Provides a placeholder for a view controller in an external storyboard.

🟡 **Navigation Controller** - A controller that manages navigation

▢▢  ⊖ Filter

The first panel (highlighted in red) is the **Inspector bar**, use it to choose the inspector best suited to your current task. Two inspectors are always visible in the inspector bar (additional inspectors are available in some editors):

- **File inspector.** View and manage metadata for the selected file. Typically you will localize storyboards and other media files and change settings for user interface files.
- **Quick Help.** View details about a symbol, an interface element, or a build setting in the file. For example, Quick Help displays a concise description of a method, where and how the method is declared, its scope, the parameters it takes, and its platform and architecture availability.

Use the **Library bar** (the second highlighted in red) to access ready-to-use libraries of resources for your project:

- **File templates.** Templates for common types of files and code constructs.
- **Code snippets.** Short pieces of source code for use in your software, such as class declarations, control flows, block declarations, and templates for commonly used Apple technologies.
- **Objects.** Items for your app's user interface.
- **Media.** Files containing graphics, icons, sound files, and the like.

> To use a library, drag it directly to the appropriate area. For example, to use a code snippet, drag it from the library to the source editor; to create a source file from a file template, drag its template to the project navigator.

To restrict the items displayed in a selected library, type relevant text into the text field in the **Filter bar** (the bottom pane). For example, type "button" in the text field to show all the buttons in the Objects library.

**Manage Tasks with the Workspace Toolbar**

The toolbar at the top of the workspace window provides quick access to frequently used commands. The **Run button** builds and runs your products. The **Stop button** terminates your running code. The **Scheme menu** lets you configure the products you want to build and run. The **activity viewer** shows the progress of tasks currently executing by displaying status messages, build progress, and other information about your project.

The **editor configuration buttons** (the first group of three buttons) let you configure the editor area, and the **workspace configuration buttons** (the second group of three buttons) hide or show the optional navigator, debug, and utilities areas.



The **View menu** includes commands to hide or show the toolbar.
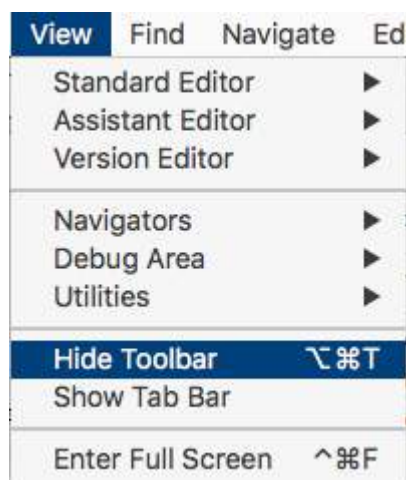


# Section 1.4: Create your first program in Swift 3

Here I am presenting how to create first basic program in Swift 3 language. First you need to have any basic programming language knowledge or not having then be ready to learn it from beginning.

Requirements for developments:

1. MAC OS - Version 10.11.6 or later for new Xcode 8.2
2. Xcode - Version 8.2 [Apple Document for Xcode introduction.](#)

Xcode 8.2 has new Swift 3 language features with new iOS 10 compatible APi's.

**Create your first program**

First go to Application and open your Xcode 8.2.
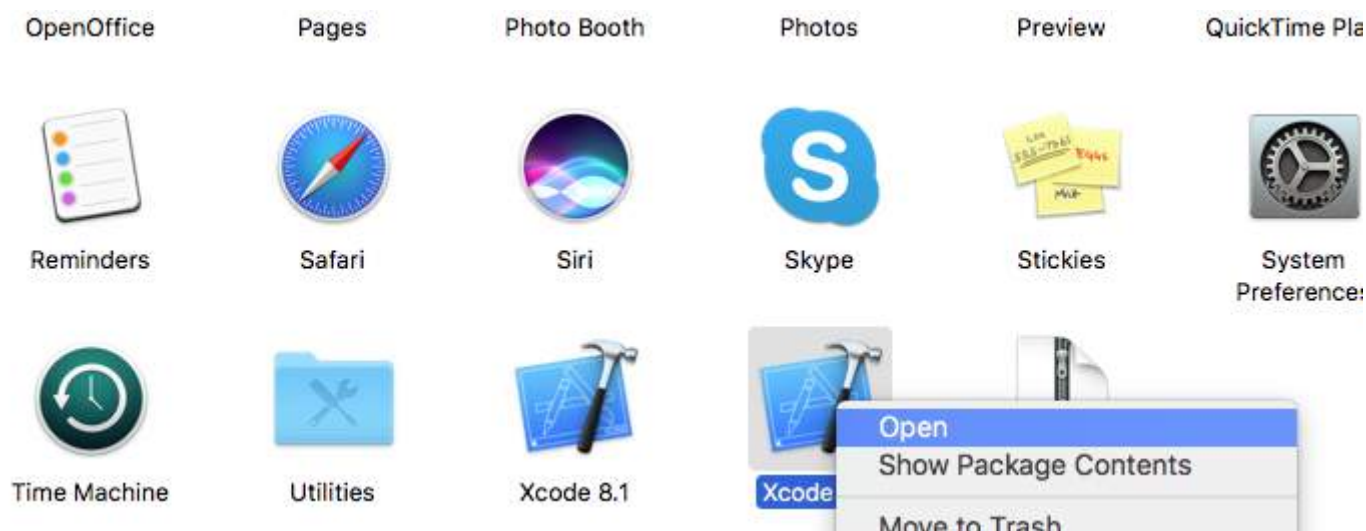


**After that you will see the screen**



**Then choose Create new Project and after that you will see next screen**
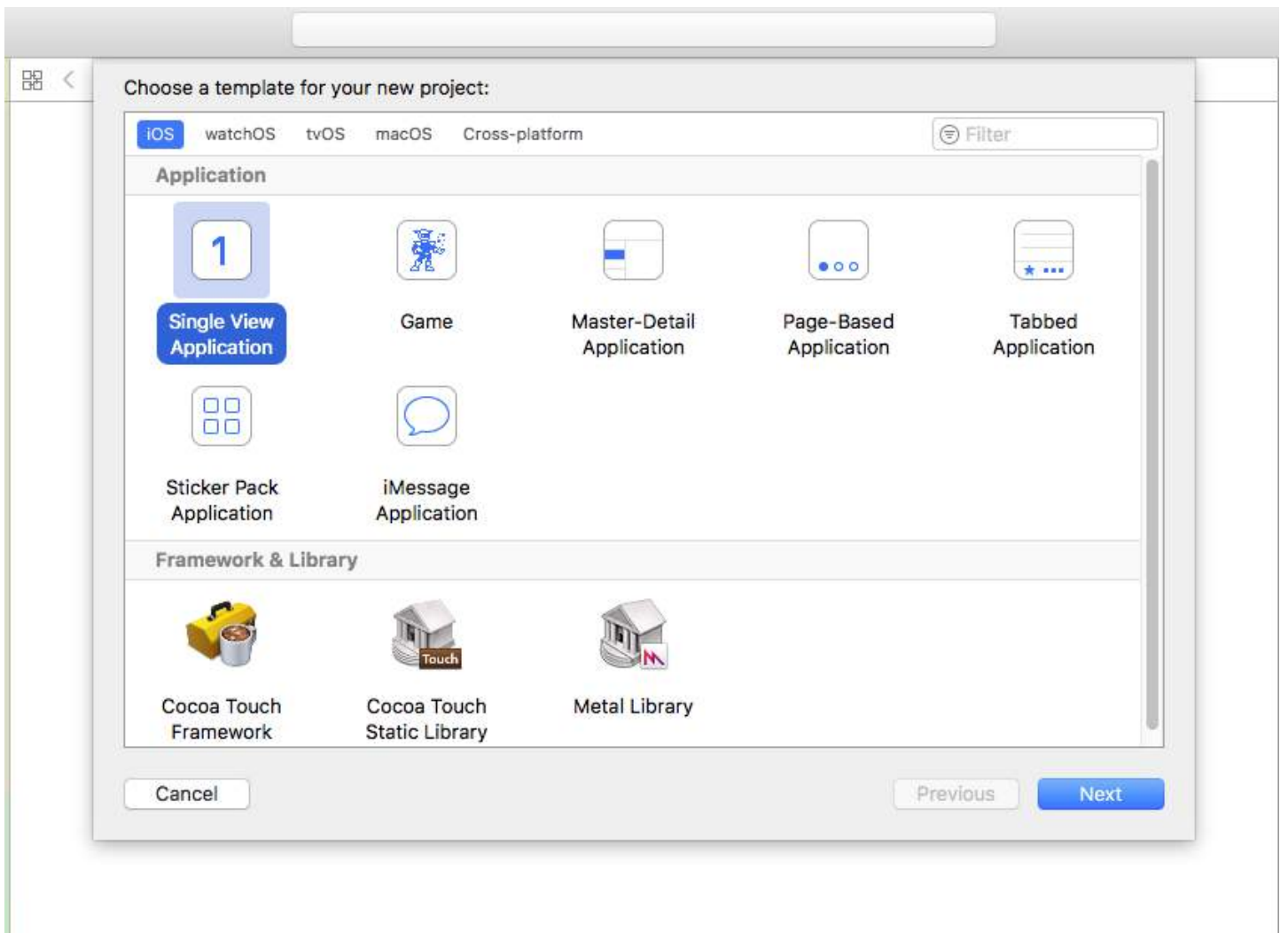
This is also very important part inside Xcode for selecting our project type. We need to choose our project according to types of OS. There are five types of options available on the top side:

1. [iOS](#)
2. [watchOS](#)
3. [macOS](#)
4. Cross-platform

Now we are choosing iOS platform for development and creating very basic project with the single view application option:
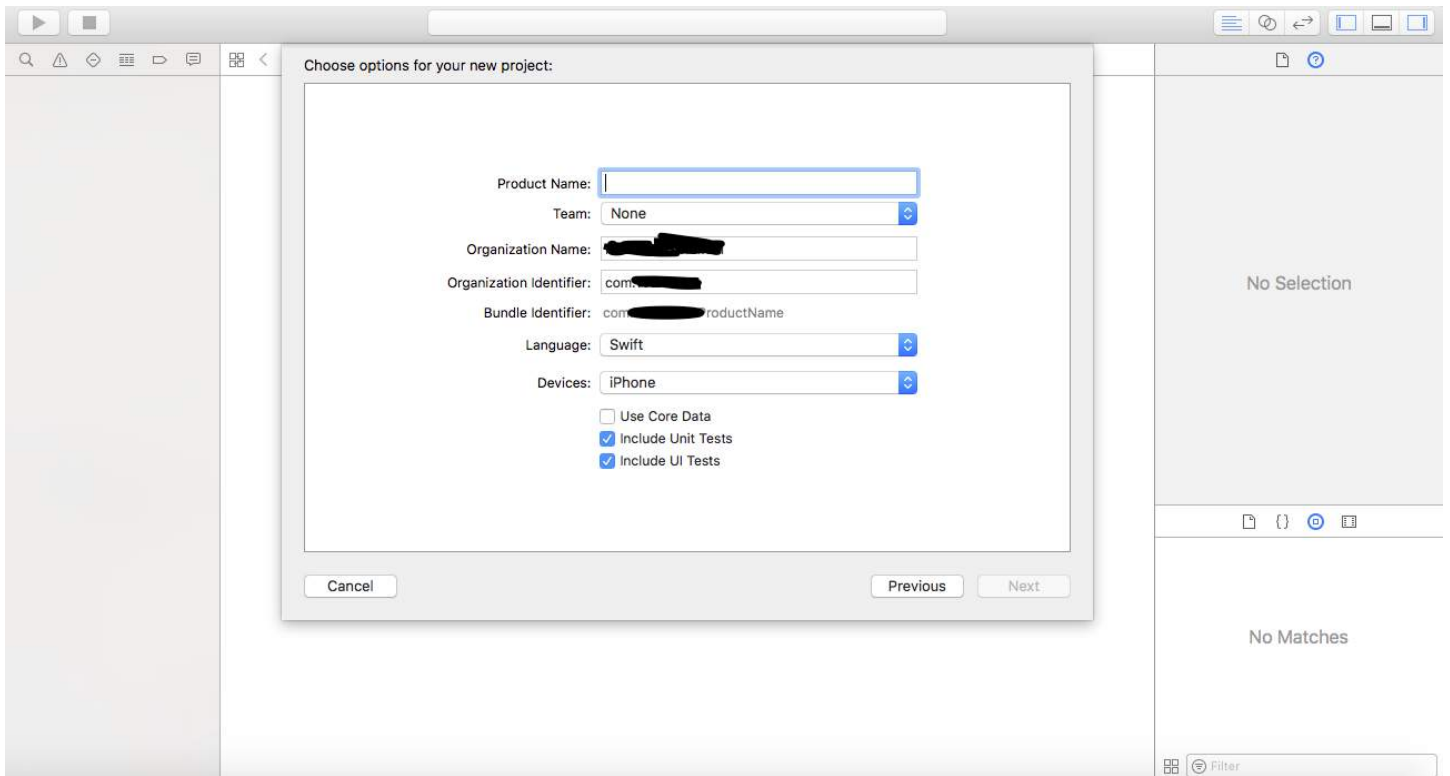
Then we need to give Product Name, this will represent your Bundle name and application name.

> Application name you can change later as per your requirements. Then we need to click "Create" and after that your screen will look like this one below:
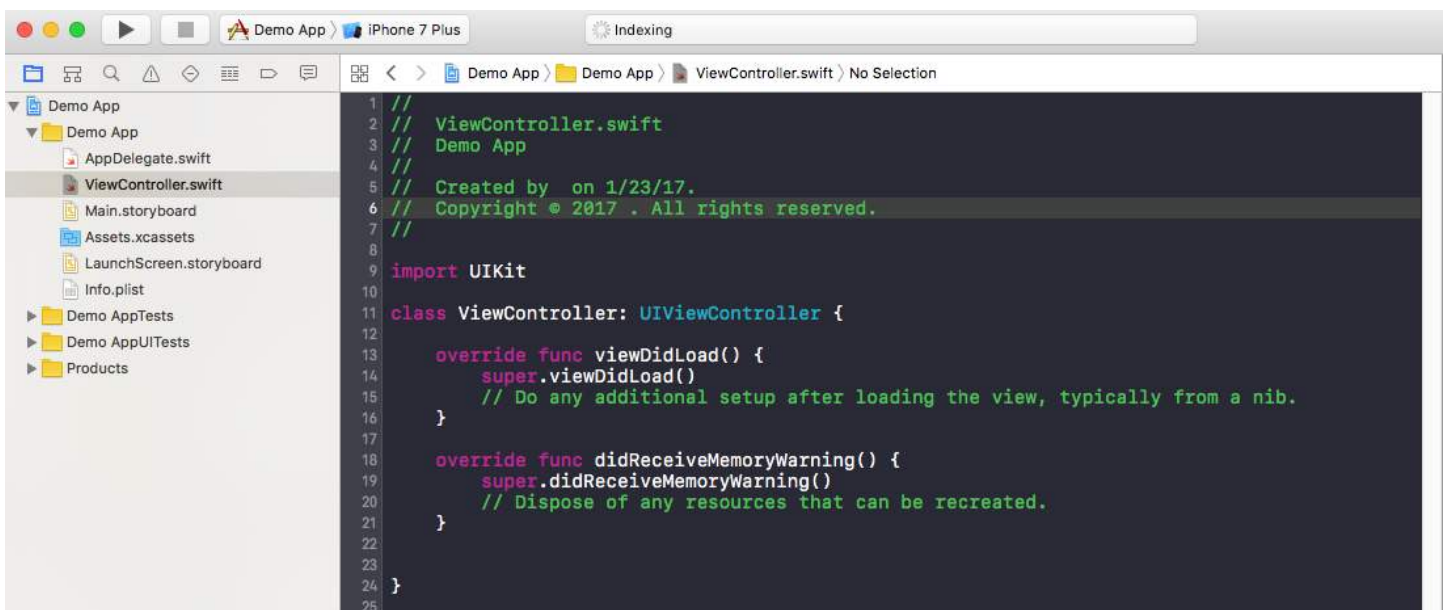


Inside this class you can see the file name is ViewController.swift and inside the class the name is also ViewController which is inheritance by the UIViewController super class and finally we're creating our first variable which name is **myString** of the type 'String'. Add the following under 'super.viewDidLoad()'

```
let myString = "Hello, World!"
```

We are going to print the content of this variable. First, select your simulator type at the top left hand side of the screen and then click on the "Run" button.

```
5  //  Created by   on 1/23/17.
6  //  Copyright © 2017 . All rights reserved.
7  //
8
9  import UIKit
10
11 class ViewController: UIViewController {
12
13     override func viewDidLoad() {
14         super.viewDidLoad()
15         let myString = "Hello, World!"
16
17         //Print here value of myString variable
18         print(myString)
19     }
20
21 }
22
23
```
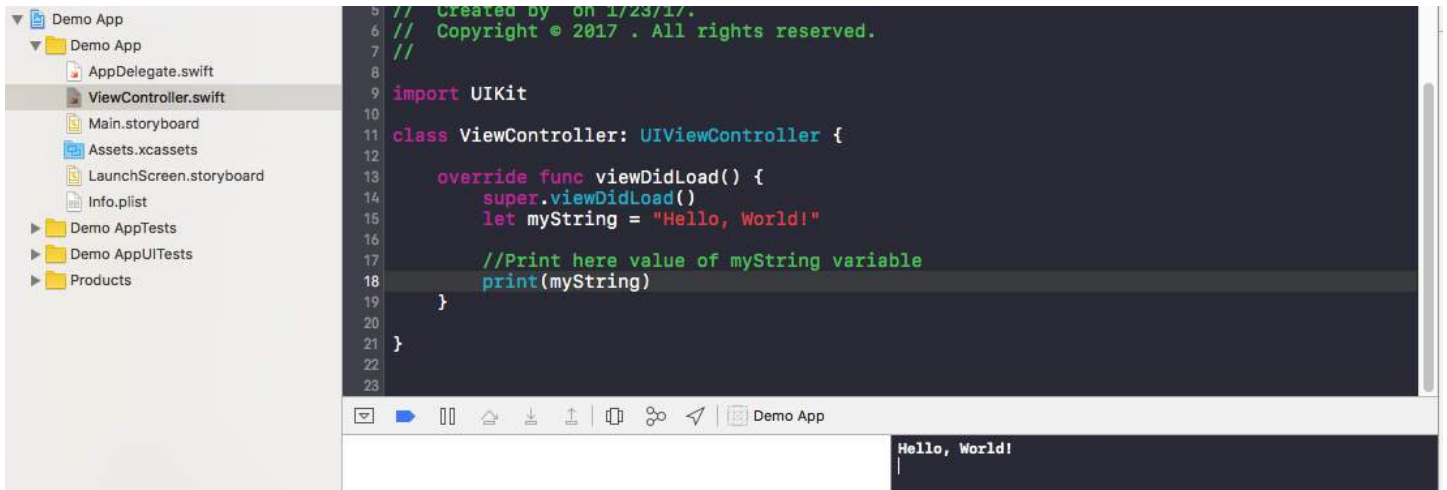
Hello, World!

After that your output will be shown on terminal which is on bottom right hand side. Congratulations, This is your first Hello World program inside Xcode.

# Chapter 2: UILabel

The UILabel class implements a read-only text view. You can use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface. The base UILabel class provides support for both simple and complex styling of the label text. You can also control over aspects of appearance, such as whether the label uses a shadow or draws with a highlight. If needed, you can customize the appearance of your text further by subclassing.

## Section 2.1: Create a UILabel

**With a Frame**

When you know the exact dimensions you want to set for your label, you can initialize a UILabel with a CGRect frame.

**Swift**

```
let frame = CGRect(x: 0, y: 0, width: 200, height: 21)
let label = UILabel(frame: frame)
view.addSubview(label)
```

**Objective-C**

```
CGRect frame = CGRectMake(0, 0, 200, 21);
UILabel *label = [[UILabel alloc] initWithFrame:frame];
[view addSubview:label];
```

**With Auto Layout**

You can add constraints on a UILabel when you want iOS to dynamically calculate its frame at runtime.

**Swift**

```
let label = UILabel()
label.backgroundColor = .red
label.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(label)

NSLayoutConstraint.activate([
    //stick the top of the label to the top of its superview:
    label.topAnchor.constraint(equalTo: view.topAnchor)

    //stick the left of the label to the left of its superview
    //if the alphabet is left-to-right, or to the right of its
    //superview if the alphabet is right-to-left:
    label.leadingAnchor.constraint(equalTo: view.leadingAnchor)

    //stick the label's bottom to the bottom of its superview:
    label.bottomAnchor.constraint(equalTo: view.bottomAnchor)

    //the label's width should be equal to 100 points:
    label.widthAnchor.constraint(equalToConstant: 100)
])
```

**Objective-C**

```
UILabel *label = [[UILabel alloc] init];
```

**With Objective-c + Visual Format Language (VFL)**

```
UILabel *label = [UILabel new];
label.translatesAutoresizingMaskIntoConstraints = NO;
[self.view addSubview label];
```

```
    // add horizontal constraints with 5 left and right padding from the leading and trailing

    [self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"V:|-5-[labelName]-5-
|"
                                                                      options:0
                                                                      metrics:nil

views:@{@"labelName":label}]];
    // vertical constraints that will use the height of the superView with no padding on top and
bottom
    [self.view addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"H:|[labelName]|"
                                                                      options:0
                                                                      metrics:nil
                                                                        views:@{@"labelName":label}]]
```

VFL documentation can be found [here](#)

After the label has been created, be sure to set the dimensions via Auto Layout. Xcode will display errors if it is done improperly.

**With Interface Builder**

You also use Interface Builder to add a `UILabel` to your `Storyboard` or `.xib` file by dragging a `Label` from the Object Library panel and dropping it into a view in the canvas:
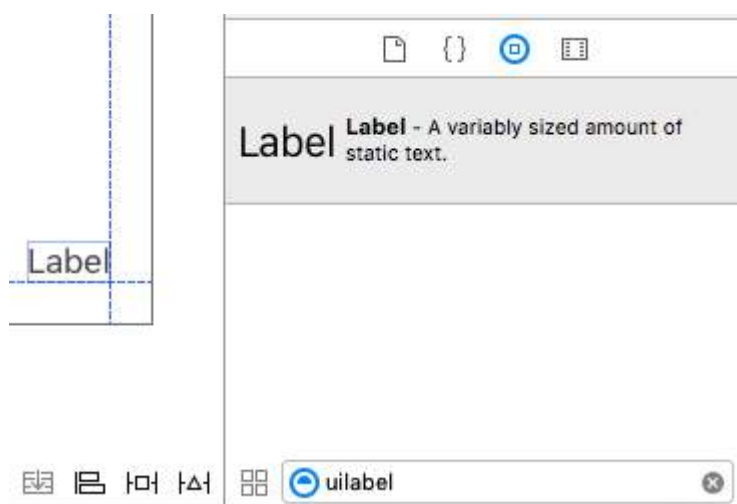


Instead of specifying a frame (position and size) for a `UILabel` programmatically, a `Storyboard` or a `.xib` lets you use Auto Layout to add constraints to the control.

In order to access this label created from `storyboard` or `xib` create an IBOutlet of this label.

**Linking Between Interface Builder and View Controller**

Once you have added a `UILabel` to your `Storyboard` or `.xib` the file you can link it to your code by pressing `Control` ? and then dragging the mouse between the `UILabel` to your `ViewController`, or you could drag to the code while right clicking for it to have the same effect.
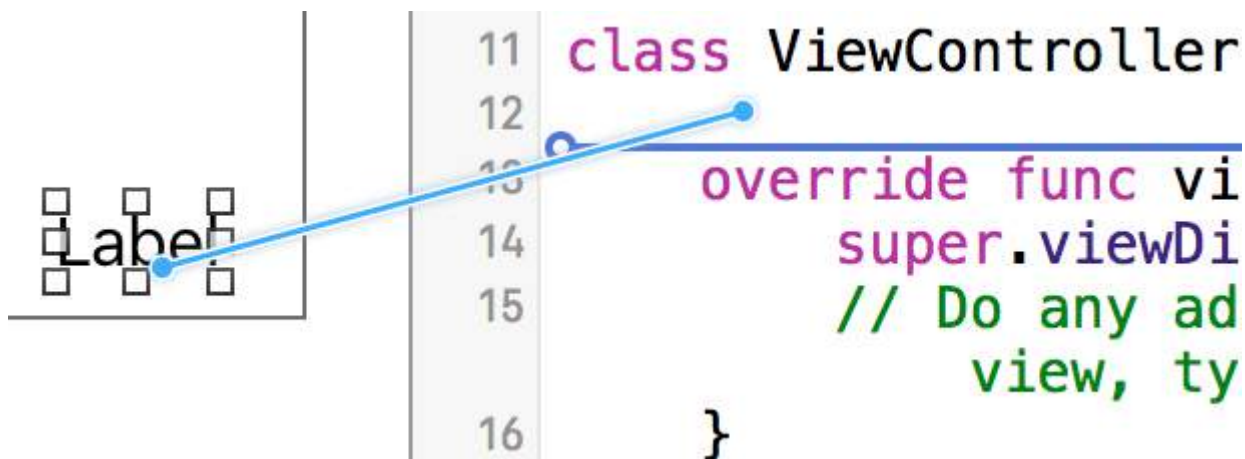
In the properties dialog, you can set the name of `UILabel`, and set it as `strong` or `weak`. For more information about `strong` and `weak`, see this,

The other way is to make the outlet programmatically as follows:

**Swift**

```swift
@IBOutlet weak var nameLabel : UILabel!
```

**Objective-C**

```objc
@property (nonatomic, weak) IBOutlet UILabel *nameLabel;
```

# Section 2.2: Number of Lines

When you make a label and set its text to be more than a single line that it can display, it will be truncated and you will see only one line of text ending with three dots (...). This is because a property called numberOfLines is set to 1, and therefore only one line will be displayed. It is a common mistake in handling `UILabel`s, and many people think of it as a bug, or they may use more than one label to show more than a line of text, but just by editing this property, we can tell a `UILabel` to accept up to the specified number of lines. For example, if this property is set to 5, the label can show 1, 2, 3, 4 or 5 lines of data.

**Setting the value programmatically**

To set this property, simply assign a new integer to it:

**Swift**

```swift
label.numberOfLines = 2
```

**Objective-C**

```objc
label.numberOfLines = 2;
```

> **Note**
>
> It is possible to set this property to 0. However, this doesn't mean that it won't accept any lines, instead it means that the label can have as many lines as needed (aka "Infinity"):

**Swift**

```swift
label.numberOfLines = 0
```

**Objective-C**

```objc
label.numberOfLines = 0;
```

> **Note**
>
> If the label has a height constraint, the constraint will be respected. In this case, `label.numberOfLines = 0` may not work as expected.

> **Note**
>
> For a more complex multi-line text, UITextView may be a better fit.*

**Setting the value in the Interface Builder**

Instead of setting `numberOfLines` programmatically, you can use a `Storyboard` or a `.xib` and set the `numberOfLines` property. That way, we achieve the same results as the above code.

Like as below:
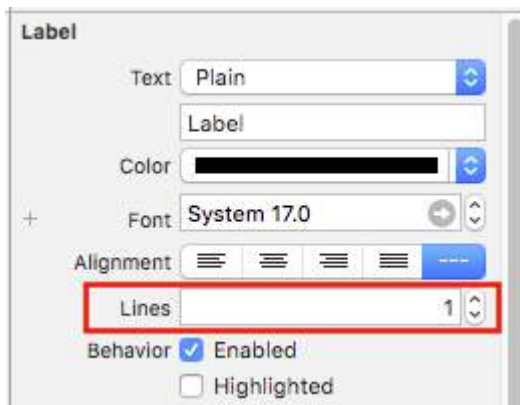


# Section 2.3: Set Font

**Swift**

```swift
let label = UILabel()
```

**Objective-C**

```objectivec
UILabel *label = [[UILabel alloc] init];
or
UILabel *label = [UILabel new]; // convenience method for calling alloc-init
```

**Change the default font's size**

**Swift**

```swift
label.font = UIFont.systemFontOfSize(17)
```

**Swift 3**

```swift
label.font = UIFont.systemFont(ofSize: 17)
```

**Objective-C**

```objectivec
label.font = [UIFont systemFontOfSize:17];
```

**Use a specific font weight**

Version ≥ iOS 8.2

**Swift**

```swift
label.font = UIFont.systemFontOfSize(17, weight: UIFontWeightBold)
```

**Swift3**

```swift
label.font = UIFont.systemFont(ofSize: 17, weight: UIFontWeightBold)
```

**Objective-C**

```
label.font = [UIFont systemFontOfSize:17 weight:UIFontWeightBold];
```
Version < iOS 8.2

**Swift**
```
label.font = UIFont.boldSystemFontOfSize(17)
```
**Swift3**
```
label.font = UIFont.boldSystemFont(ofSize: 17)
```
**Objective-C**
```
label.font = [UIFont boldSystemFontOfSize:17];
```
**Use a Dynamic Type text style.**

The font and point size will be based on the user's preferred reading size.

**Swift**
```
label.font = UIFont.preferredFontForTextStyle(UIFontTextStyleBody)
```
**Swift 3**
```
label.font = UIFont.preferredFont(forTextStyle: .body)
```
**Objective-C**
```
label.font = [UIFont preferredFontForTextStyle:UIFontTextStyleBody];
```
**Use a different font altogether**
**Swift**
```
label.font = UIFont(name: "Avenir", size: 15)
```
**Objective-C**
```
label.font = [UIFont fontWithName:@"Avenir" size:15];
```
**Override font size**

A way to set the font size without knowing the font family is to use the **font** property of the `UILabel`.

**Swift**
```
label.font = label.font.fontWithSize(15)
```
**Swift 3**
```
label.font = label.font.withSize(15)
```
**Objective-C**
```
label.font = [label.font fontWithSize:15];
```
**Use Custom Font Swift**

Refer to this link

# Section 2.4: Text Color

You can use the label's `textColor` property to apply a text color to the entire text of the label.

**Swift**
```
label.textColor = UIColor.redColor()
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

**Swift 3**
```
label.textColor = UIColor.red
```

```
label.textColor = UIColor(red: 64.0/255.0, green: 88.0/255.0, blue: 41.0/225.0, alpha: 1)
```

**Objective-C**

```
label.textColor = [UIColor redColor];
label.textColor = [UIColor colorWithRed:64.0f/255.0f green:88.0f/255.0f blue:41.0f/255.0f
alpha:1.0f];
```

**Applying text color to a portion of the text**

You can also vary the text color (or other attributes) of portions of the text by using NSAttributedString:

**Objective-C**

```
attributedString = [[NSMutableAttributedString alloc] initWithString:@"The grass is green; the sky
is blue."];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(13, 5)];
[attributedString addAttribute: NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(31, 4)];
label.attributedText = attributesString;
```

**Swift**

```
let attributedString = NSMutableAttributedString(string: "The grass is green; the sky is blue.")
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.green(), range:
NSRange(location: 13, length: 5))
attributedString.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue(), range:
NSRange(location: 31, length: 4))
label.attributedText = attributedString
```

# Section 2.5: Background Color

**Swift**

```
label.backgroundColor = UIColor.redColor()

label.backgroundColor = .redColor()
```

**Swift 3**

```
label.backgroundColor = UIColor.red
```

**Objective-C**

```
label.backgroundColor = [UIColor redColor];
```

# Section 2.6: Size to fit

Suppose you have a UILabel on your storyboard and you have created an IBOutlet for it in
ViewController.swift / ViewController.m and named it labelOne.

To make the changes easily visible, change the backgroundColor and textColor of labelOne in the viewDidLoad
method:

The function sizeToFit is used when you want to automatically resize a label based on the content stored within it.

---

**Swift**

```
labelOne.backgroundColor = UIColor.blueColor()
labelOne.textColor = UIColor.whiteColor()
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

**Swift 3**

```
labelOne.backgroundColor = UIColor.blue
labelOne.textColor = UIColor.white
labelOne.text = "Hello, World!"
labelOne.sizeToFit()
```

**Objective-C**

```
labelOne.backgroundColor = [UIColor blueColor];
labelOne.textColor = [UIColor whiteColor];
labelOne.text = @"Hello, World!";
[labelOne sizeToFit];
```

The output for the above code is:
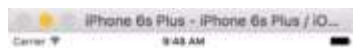


As you can see, there is no change as the text is perfectly fitting in labelOne. sizeToFit only changes the label's frame.

Let's change the text to a slightly longer one:

```
labelOne.text = "Hello, World! I'm glad to be alive!"
```

Now, labelOne looks like this:

Even calling `sizeToFit` doesn't change anything. This is because by default, the numberOfLines shown by the UILabel is set to 1. Let's change it to zero on the storyboard:



This time, when we run the app, labelOne appears correctly:

The `numberOfLines` property can also be changed in the `ViewController` file :

```
// Objective-C
labelOne.numberOfLines = 0;

// Swift
labelOne.numberOfLines = 0
```

# Section 2.7: Text alignment

**Swift**

```
label.textAlignment = NSTextAlignment.left
//or the shorter
label.textAlignment = .left
```

Any value in the NSTextAlignment enum is valid: `.left`, `.center`, `.right`, `.justified`, `.natural`

**Objective-C**

```
label.textAlignment = NSTextAlignmentLeft;
```

Any value in the NSTextAlignment enum is valid: NSTextAlignmentLeft, NSTextAlignmentCenter, NSTextAlignmentRight, NSTextAlignmentJustified, NSTextAlignmentNatural

Vertical alignment in `UILabel` is not supported out of the box: Vertically align text to top within a UILabel

# Section 2.8: Calculate Content Bounds (for i.e. dynamic cell heights)

A common use case for wanting to calculate the frame a label will take up is for sizing table view cells appropriately. The recommended way of doing this is using the `NSString` method `boundingRectWithSize:options:attributes:context:.`

options takes String drawing options:

- `NSStringDrawingUsesLineFragmentOrigin` should be used for labels with multiple lines

- NSStringDrawingTruncatesLastVisibleLine should be added using the | operator if there are a maximum number of lines

`attributes` is an `NSDictionary` of attributes that effect attributed strings (full list: [Apple Docs](#)) but the factors that effect height include:

- **NSFontAttributeName**: Very important, the size and font family is a critical part of the label's displayed size.

- **NSParagraphStyleAttributeName**: For customizing how the text is displayed. This includes line spacing, text alignment, truncation style, and a few other options. If you did not explicitly change any of these values you should not have to worry about this much, but may be important if you toggled some values on IB.

`context` should be `nil` since the primary `NSStringDrawingContext` use case is for allowing font to resize to fit a specified rect, which shouldn't be the case if we're calculating a dynamic height.

**Objective C**

```objc
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {

    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];

    NSString *labelContent = cell.theLabel.text;
    // you may choose to get the content directly from the data source if you have done minimal
customizations to the font or are comfortable with hardcoding a few values
//    NSString *labelContent = [self.dataSource objectAtIndexPath:indexPath];

    // value may be hardcoded if retrieved from data source
    NSFont *labelFont = [cell.theLabel font];

    // The NSParagraphStyle, even if you did not code any changes these values may have been
altered in IB
    NSMutableParagraphStyle *paragraphStyle = [NSMutableParagraphStyle new];
    paragraphStyle.lineBreakMode = NSLineBreakByWordWrapping;
    paragraphStyle.alignment = NSTextAlignmentCenter;

    NSDictionary *attributes = @{NSFontAttributeName: labelFont,
                                 NSParagraphStyleAttributeName: paragraphStyle};

    // The width is also important to the height
    CGFloat labelWidth = CGRectGetWidth(cell.theLabel.frame);
    // If you have been hardcoding up to this point you will be able to get this value by
subtracting the padding on left and right from tableView.bounds.size.width
//    CGFloat labelWidth = CGRectGetWidth(tableView.frame) - 20.0f - 20.0f;

    CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, CGFLOAT_MAX)
options:NSStringDrawingUsesLineFragmentOrigin attributes:attributes context:nil];

    return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
}
```

**Swfit 3**

```swift
override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    var cell = tableView.cellForRow(atIndexPath: indexPath)!
    var labelContent = cell.theLabel.text
    var labelFont = cell.theLabel.font
    var paragraphStyle = NSMutableParagraphStyle()

    paragraphStyle.lineBreakMode = .byWordWrapping
```

```
    paragraphStyle.alignment = .center

    var attributes = [NSFontAttributeName: labelFont, NSParagraphStyleAttributeName:
paragraphStyle]

    var labelWidth: CGFloat = cell.theLabel.frame.width

    var bodyBounds = labelContent.boundingRect(withSize: CGSize(width: width, height: CGFLOAT_MAX),
options: .usesLineFragmentOrigin, attributes: attributes, context: nil)

    return bodyBounds.height + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel
}
```

Conversely, if you do have a set maximum number of lines you will first need calculate the height of a single line to make sure we don't get a value taller than the allowed size:

```
    // We calculate the height of a line by omitting the NSStringDrawingUsesLineFragmentOrigin
option, which will assume an infinitely wide label
    CGRect singleLineRect = [labelContent boundingRectWithSize:CGSizeMake(CGFLOAT_MAX, CGFLOAT_MAX)
                                           options:NSStringDrawingTruncatesLastVisibleLine
                                           context:nil];
    CGFloat lineHeight = CGRectGetHeight(singleLineRect);
    CGFloat maxHeight = lineHeight * cell.theLabel.numberOfLines;

    // Now you can call the method appropriately
    CGRect bodyBounds = [labelContent boundingRectWithSize:CGSizeMake(width, maxHeight)
options:(NSStringDrawingUsesLineFragmentOrigin|NSStringDrawingTruncatesLastVisibleLine)
attributes:attributes context:nil];

    return CGRectGetHeight(bodyBounds) + heightOfObjectsOnTopOfLabel + heightOfObjectBelowLabel;
```

# Section 2.9: Label Attributed Text

**01. Underline Text :- Single/Double Line , Strike Through :- Single/Double Line**

**Step 1**

Select the Label and change the label type Plain to Attributed
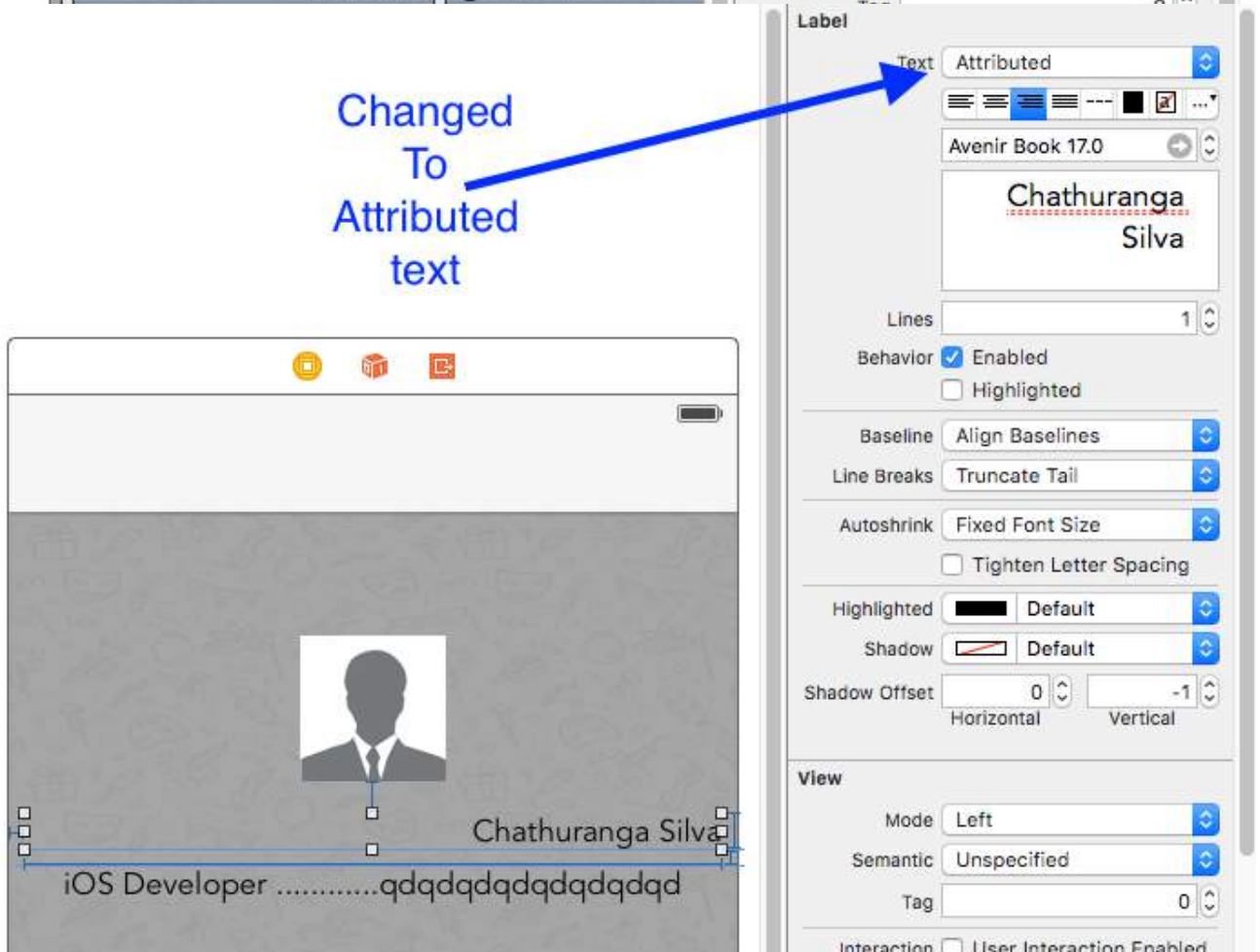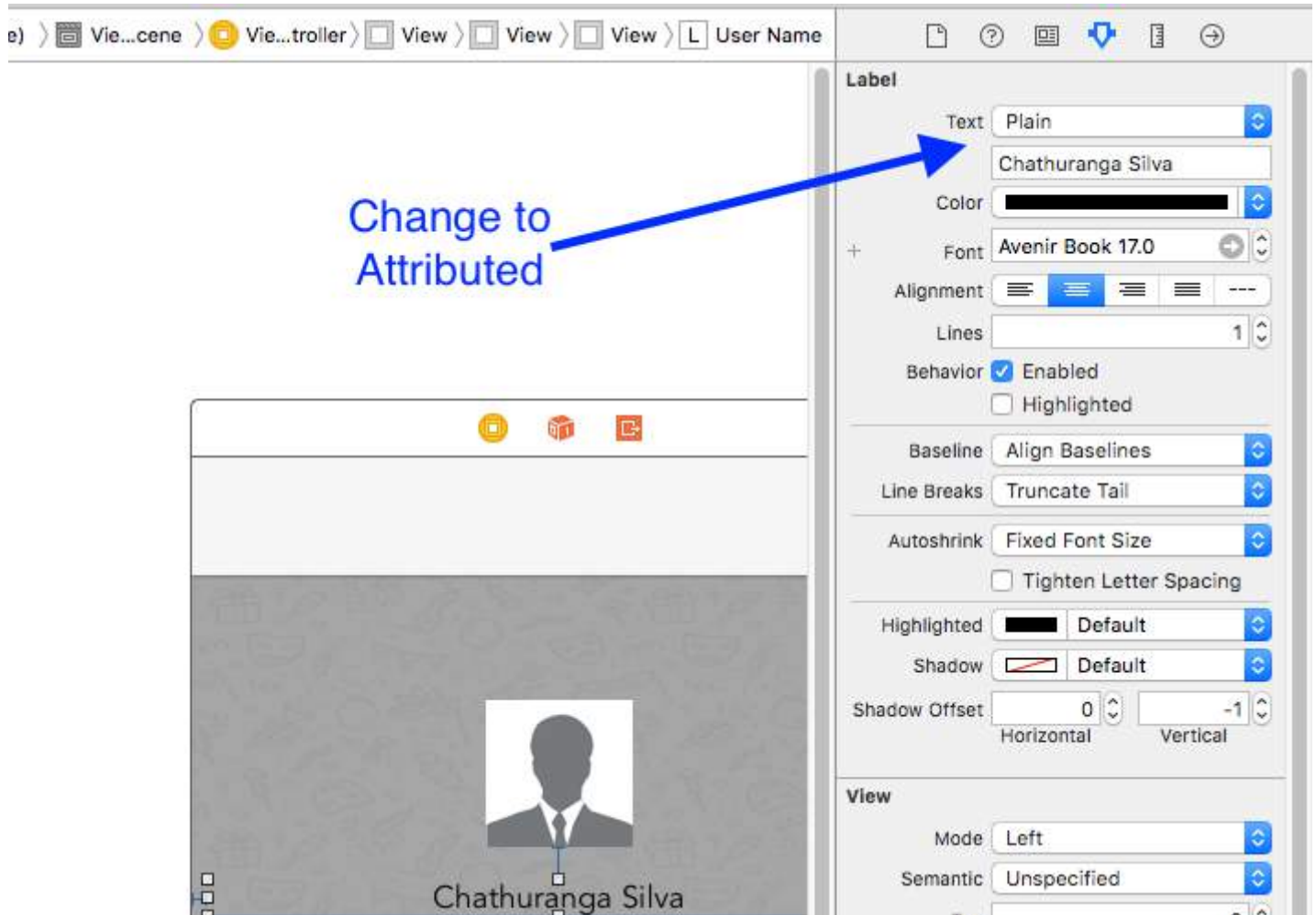
**Step 2**

Click the label text and Right click
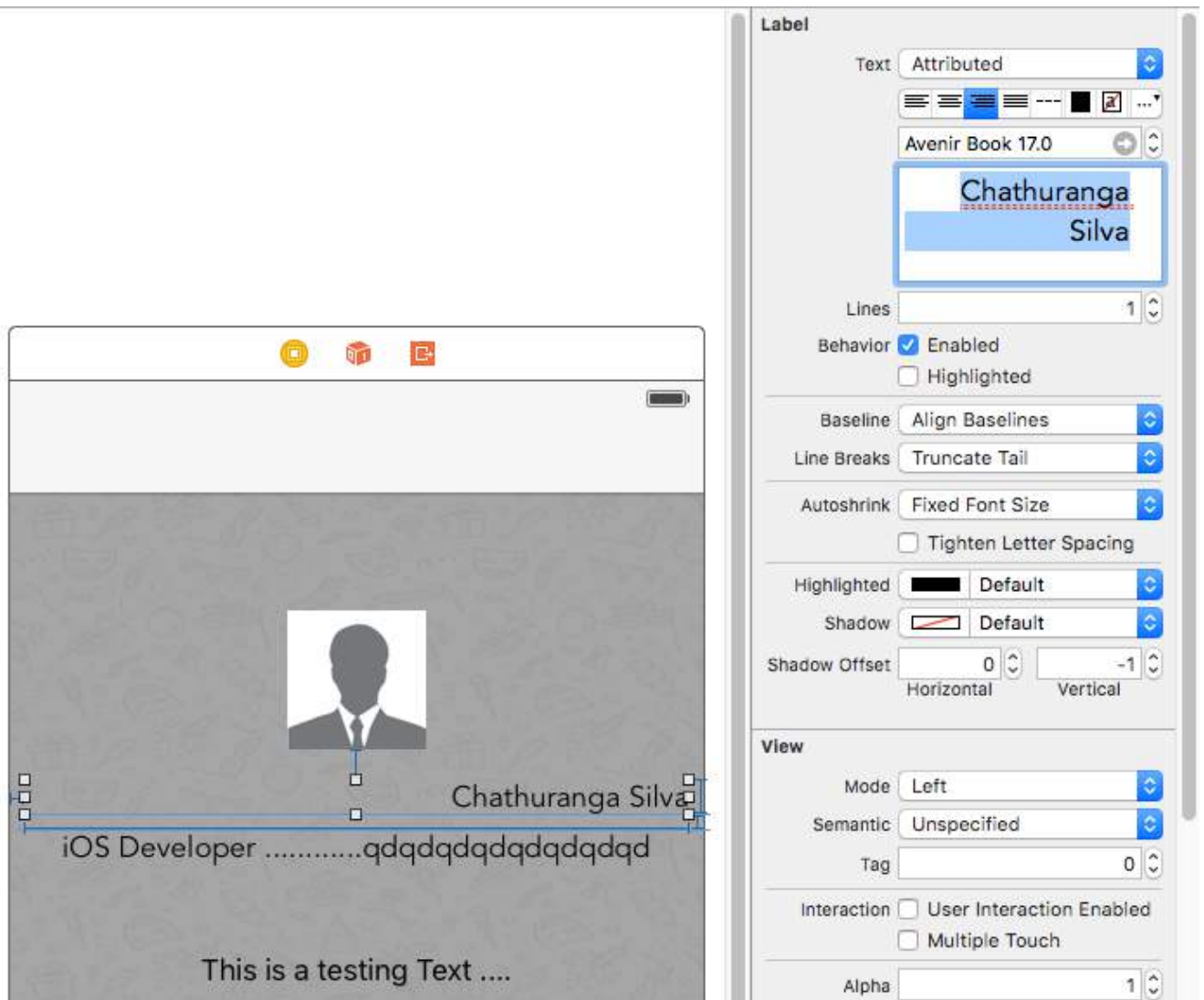


**Step 3**

Then click Font -> Show Fonts
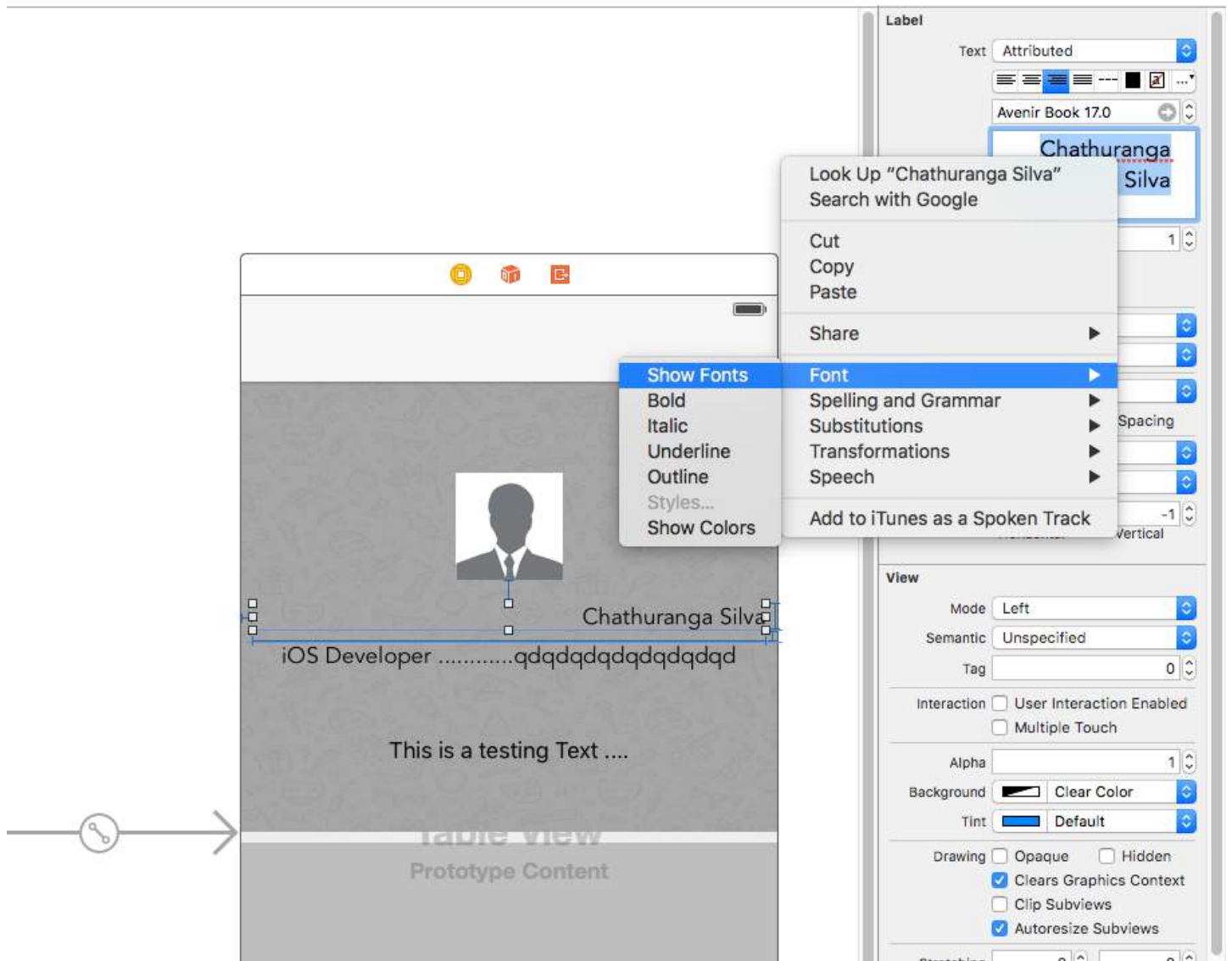
**Step 4**

Then font view will show up and click underline button to make text underline or click strikethrough button to make the text strikethrough.And select single line or double line.
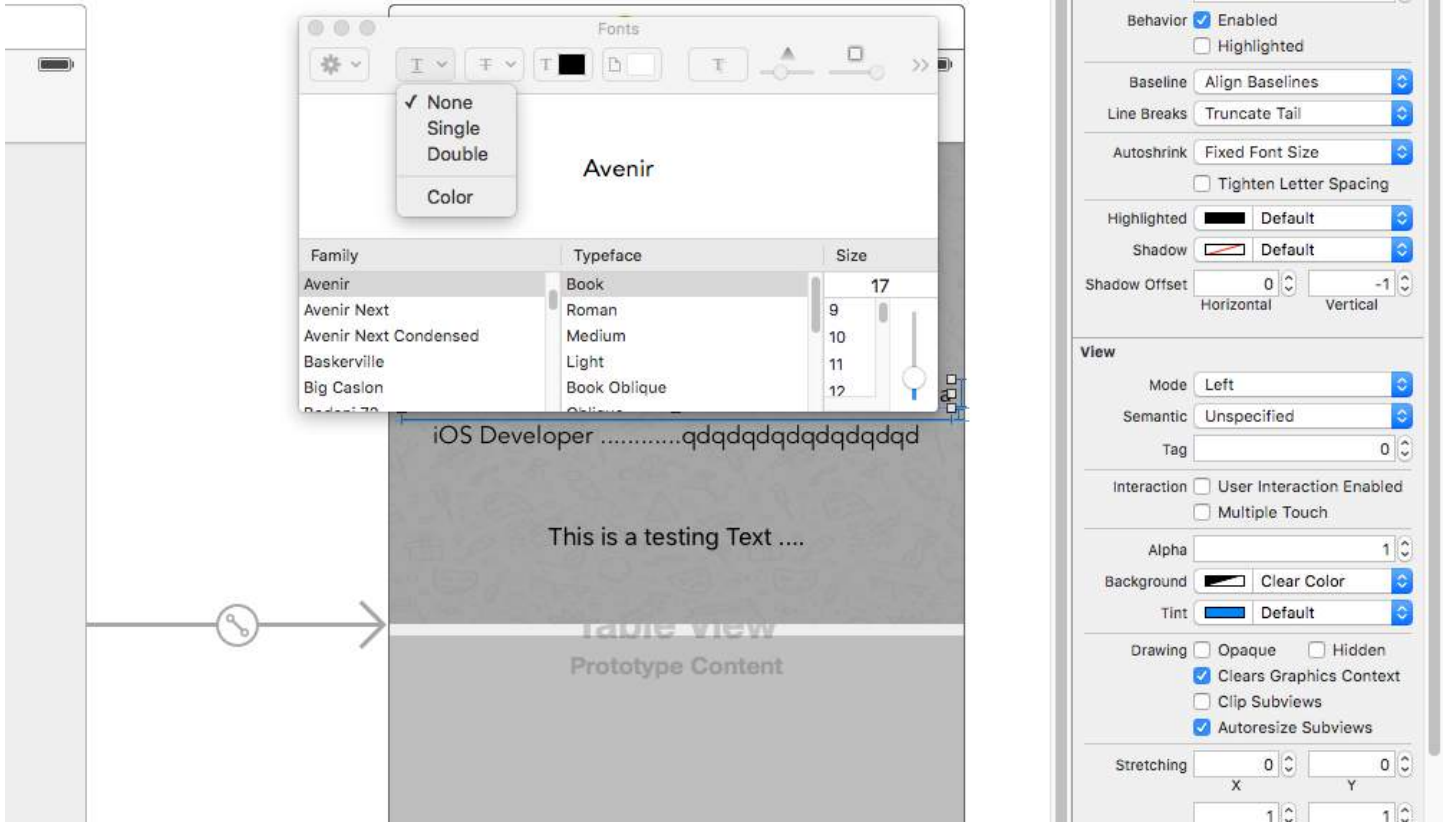
Finally click enter and label will be shown underline or strikethrough according to your selection.
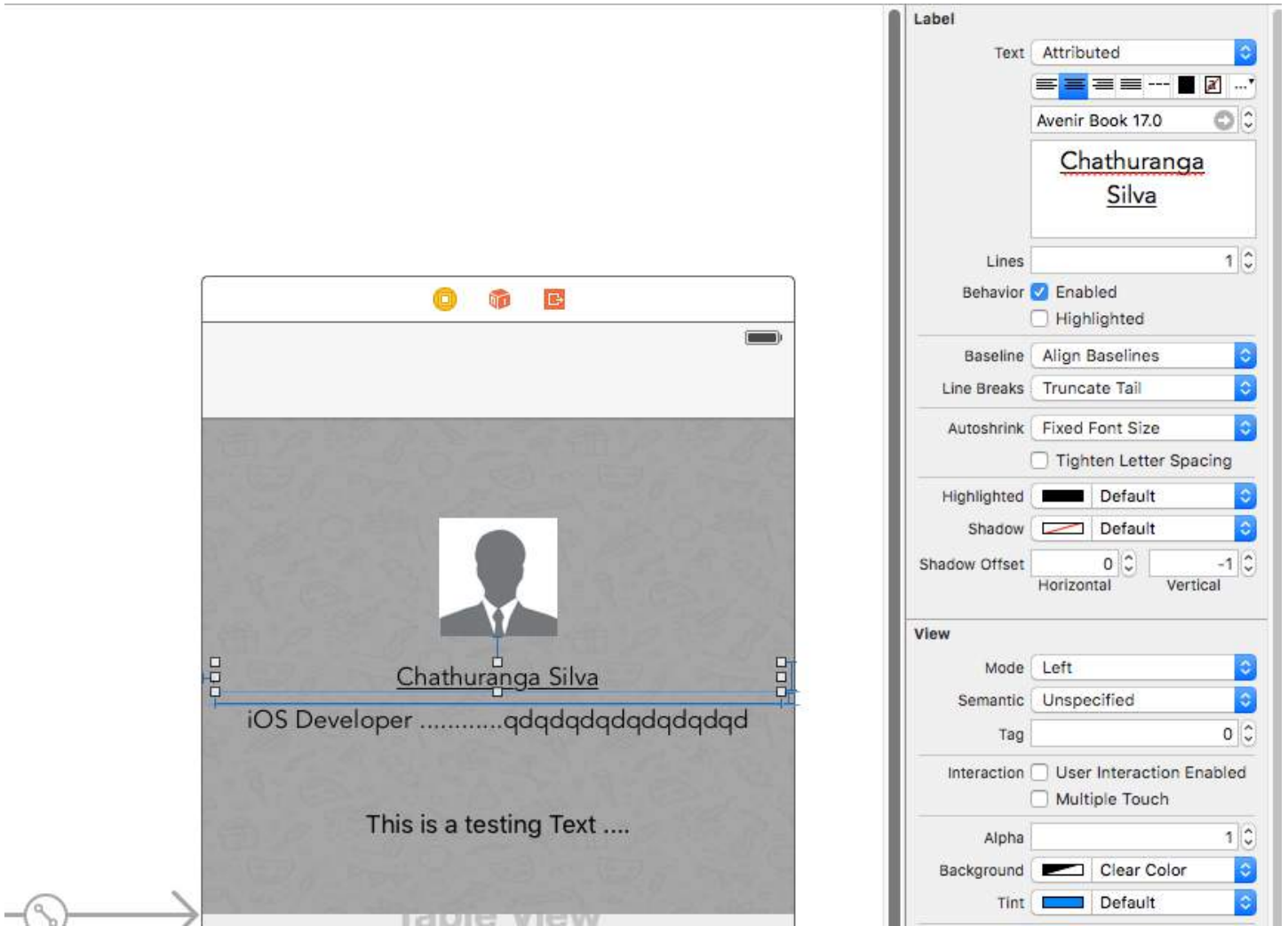
**02. Add text shaddow/background blur effects**

Get the Font view as the above described and click the effects button.
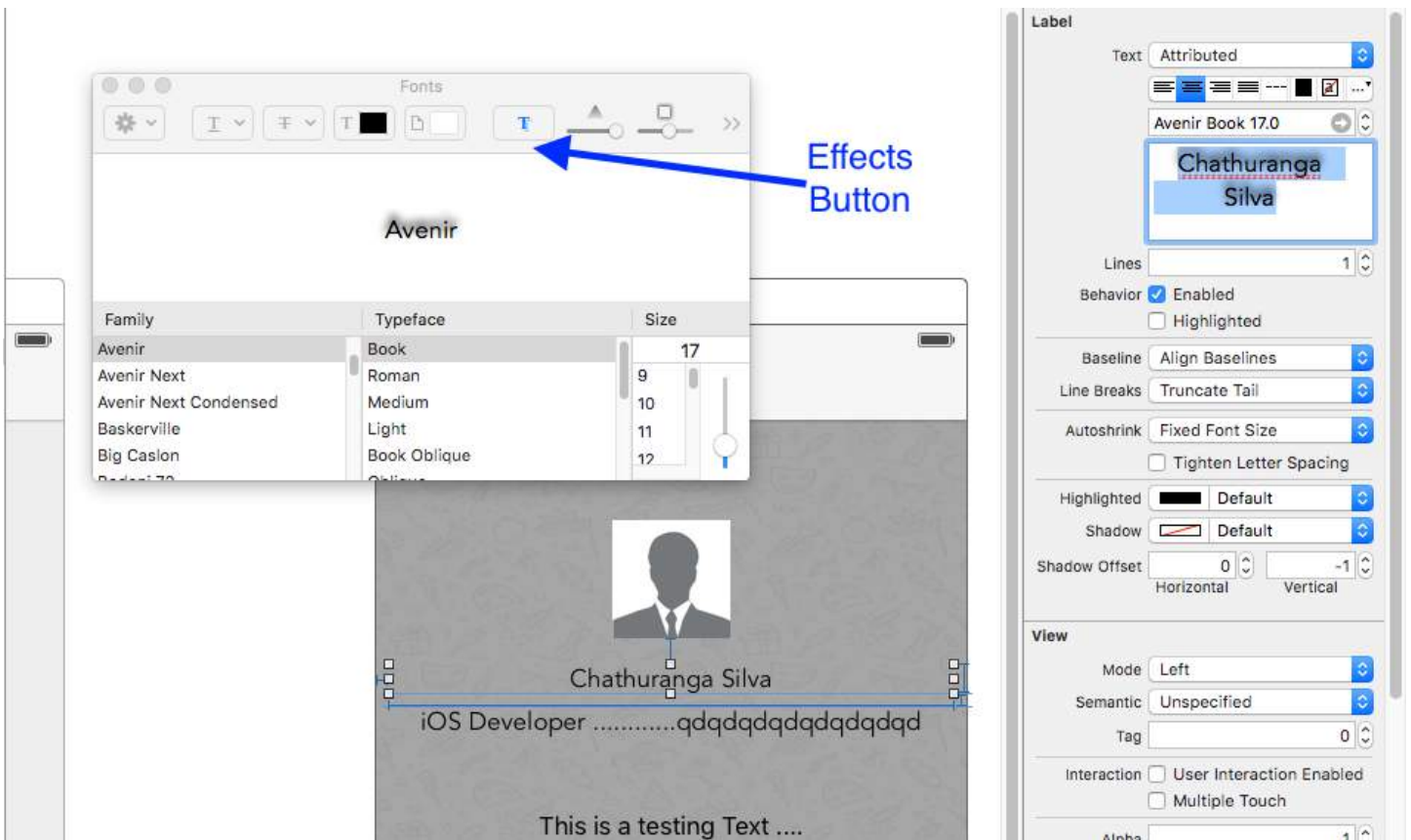
If you don't See the preview click the show image in settings



Finally change shaddow and offset according to your preferences.
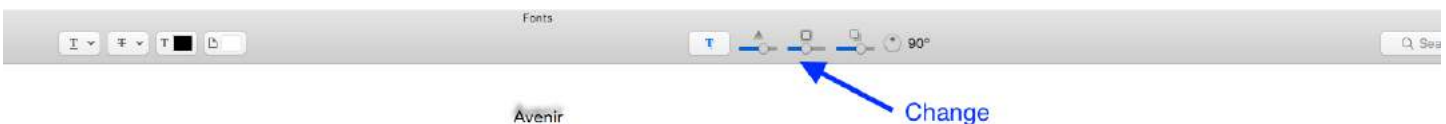


# Section 2.10: Clickable Label

> **NOTE:** In most cases, it is better to use a `UIButton` instead of making a `UILabel` you can tap on. Only use this example, if you are sure, that you don't want to use a `UIButton` for some reason.

1. Create label
2. Enable user interaction
3. Add `UITapGestureRecognizer`

The key to create a clickable `UILabel` is to enable user interaction.

**Swift**

```swift
let label = UILabel()
```

```
label.userInteractionEnabled = true

let gesture = UITapGestureRecognizer(target: self, action: #selector(labelClicked(_:)))
label.addGestureRecognizer(gesture)
```

**Objective-C**

```
UILabel *label = [[UILabel alloc] init];
[label setUserInteractionEnabled:YES];

UITapGestureRecognizer* gesture = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(labelClicked:)];
[label addGestureRecognizer:gesture];
```

**Setting "userInteractionEnabled" in storyboard's attributes inspector**

Instead of using code, you can select the UILabel inside the storyboard and check the option:



# Section 2.11: Variable height using constraints

You can make an UILabel with a dynamic height using auto layout.

You need to set the numberOfLines to zero (0), and add a minimal height by setting up a constraints with a relation of type .GreaterThanOrEqual on the .Height attribute

Version ≥ iOS 6
**Swift**

```
label.numberOfLines = 0

let heightConstraint = NSLayoutConstraint(
    item: label,
    attribute: .Height,
    relatedBy: .GreaterThanOrEqual,
    toItem: nil,
    attribute: .NotAnAttribute,
    multiplier: 0,
    constant: 20
)

label.addConstraint(heightConstraint)
```

Version ≥ iOS 9
**Swift**

```
label.numberOfLines = 0
label.translatesAutoresizingMaskIntoConstraints = false
label.heightAnchor.constraintGreaterThanOrEqualToConstant(20).active = true
```

# Section 2.12: LineBreakMode

**Using code**

```
UILabel.lineBreakMode: NSLineBreakMode
```

**Swift**

```swift
label.lineBreakMode = .ByTruncatingTail
```

- `.ByWordWrapping`
- `.ByCharWrapping`
- `.ByClipping`
- `.ByTruncatingHead`
- `.ByTruncatingTail`
- `.ByTruncatingMiddle`

**Swift 3**

```swift
label.lineBreakMode = .byTruncatingTail
```

- `.byWordWrapping`
- `.byCharWrapping`
- `.byClipping`
- `.byTruncatingHead`
- `.byTruncatingTail`
- `.byTruncatingMiddle`

**Objective-C**

```objectivec
[label setLineBreakMode:NSLineBreakByTruncatingTail];
```

- NSLineBreakByWordWrapping
- NSLineBreakByCharWrapping
- NSLineBreakByClipping
- NSLineBreakByTruncatingHead
- NSLineBreakByTruncatingTail
- NSLineBreakByTruncatingMiddle

**Using storyboard**

This can also be set in the attributes inspector of a UILabel:
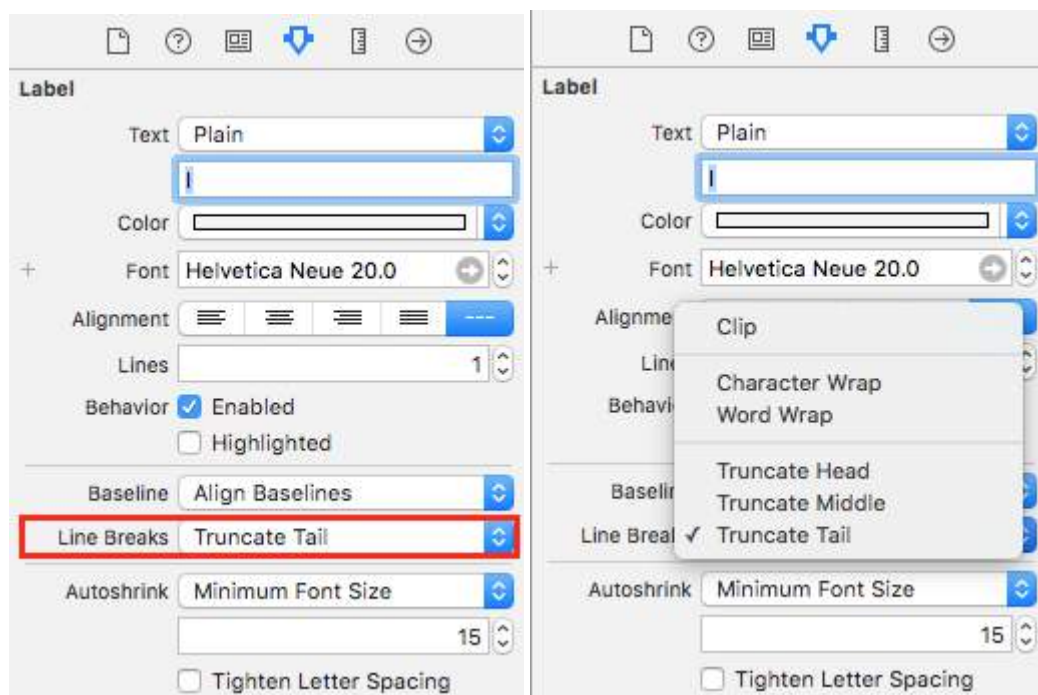


**Constants**

- Word Wrapping - wrapping occurs at word boundaries, unless the word itself doesn't fit on a single line
- Char Wrapping - wrapping occurs before the first character that doesn't fit
- Clipping - lines are simply not drawn past the edge of the text container
- Truncating Head - the line is displayed so that the end fits in the container and the missing text at the beginning of the line is indicated by an ellipsis glyph
- Truncating Tail - the line is displayed so that the beginning fits in the container and the missing text at the end of the line is indicated by an ellipsis glyph
- Truncating Middle - the line is displayed so that the beginning and end fit in the container and the missing text in the middle is indicated by an ellipsis glyph

# Section 2.13: Add shadows to text

***Swift***

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.layer.shadowRadius = 2
```

**Swift 3**

```
label1.layer.shadowOffset = CGSize(width: 3, height: 3)
label1.layer.shadowOpacity = 0.7
label1.layer.shadowRadius = 2
```

***Objective-C***

```
label1.layer.shadowOffset = CGSizeMake(3, 3);
label1.layer.shadowOpacity = 0.7;
label1.layer.shadowRadius = 2;
```

I Like My Cat

# Section 2.14: Changing Text in an Existing Label

Changing the text of an existing `UILabel` can be done by accessing and modifying the `text` property of the `UILabel`. This can be done directly using `String` literals or indirectly using variables.

**Setting the text with `String` literals**

**Swift**

```
label.text = "the new text"
```

**Objective-C**

```
// Dot Notation
label.text = @"the new text";

// Message Pattern
```

```
[label setText:@"the new text"];
```

**Setting the text with a variable**

**Swift**

```swift
let stringVar = "basic String var"
label.text = stringVar
```

**Objective-C**

```objc
NSString * stringVar = @"basic String var";

// Dot Notation
label.text = stringVar;

// Message Pattern
[label setText: stringVar];
```

# Section 2.15: Auto-size label to fit text

This example shows how a label's width can automatically resize when the text content changes.
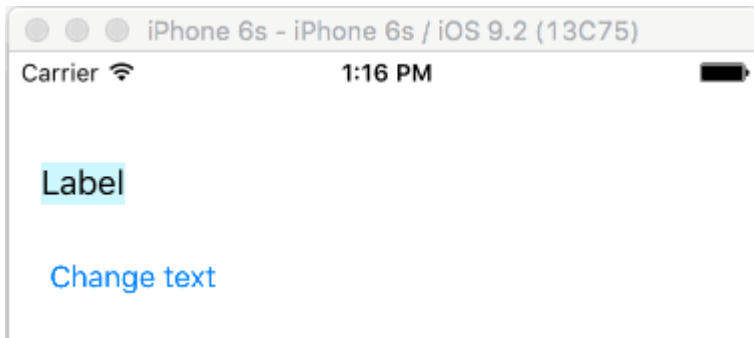


**Pin the left and top edges**

Just use auto layout to add constraints to pin the left and top sides of the label.
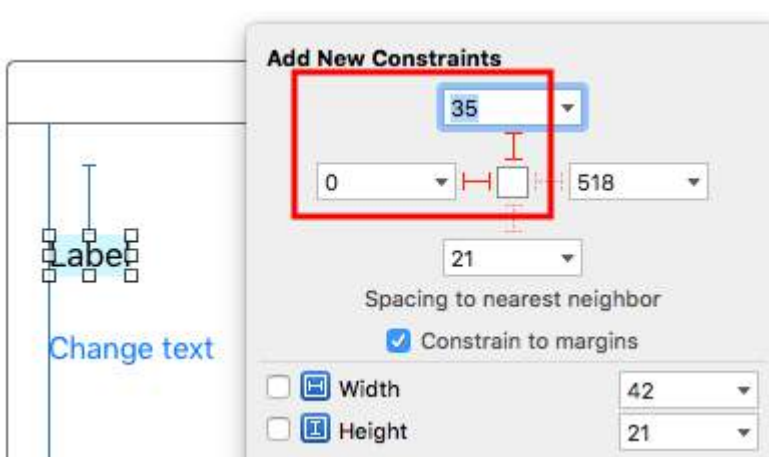


After that it will automatically resize.

**Notes**

- This example comes from this Stack Overflow answer.

- Don't add constraints for the width and height. Labels have an *intrinsic* size based on their text content.

- No need to set `sizeToFit` when using auto layout. The complete code for the example project is here:

```swift
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var myLabel: UILabel!

    @IBAction func changeTextButtonTapped(sender: UIButton) {
        myLabel.text = "my name is really long i want it to fit in this box"
    }
}
```

- This method can also be used to correctly space multiple labels horizontally as in [this example](#).



- If you want your label to line wrap then set the number of lines to 0 in IB and add
  `myLabel.preferredMaxLayoutWidth = 150 // or whatever` in code. (The button is also pinned to the bottom of the label so that it will move down when the label height increased.)
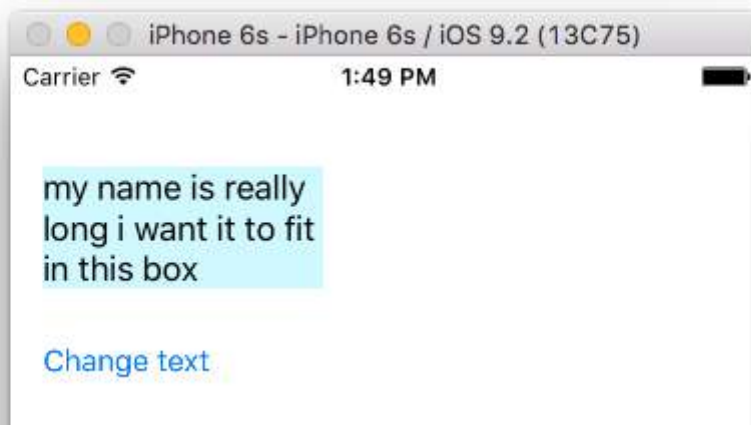


# Section 2.16: Get UILabel's size strictly based on its text and font

`NSString` provides method `boundingRectWithSize` which can be used to predict the resulting CGSize of a `UILabel` based on its text and font without the need of creating a `UILabel`

**Objective-C**

```objc
[[text boundingRectWithSize:maxSize options:(NSStringDrawingTruncatesLastVisibleLine |
NSStringDrawingUsesLineFragmentOrigin) attributes:@{NSFontAttributeName: fontName} context:nil]
size];
```

**Swift**

```
let nsText = text as NSString?
nsText?.boundingRectWithSize(maxSize, options: [.TruncatesLastVisibleLine,
.UsesLineFragmentOrigin], attributes: [NSFontAttributeName: fontName], context: nil).size
```

**Swift**

Create Label and label Height constraint outlet. Add below code where you will asign text to label.

```
@IBOutlet var lblDescriptionHeightConstration: NSLayoutConstraint!
@IBOutlet weak var lblDescription: UILabel!

let maxWidth = UIScreen.mainScreen().bounds.size.width - 40
let sizeOfLabel = self.lblDesc.sizeThatFits(CGSize(width: maxWidth, height: CGFloat.max))
self.lblDescriptionHeightConstration.constant = sizeOfLabel.height
```

Note: "40" is the space of left and right side of screen.

# Section 2.17: Highlighted and Highlighted Text Color

**Objective-C**

```
UILabel *label = [[UILabel alloc] init];
label.highlighted = YES;
label.highlightedTextColor = [UIColor redColor];
```

**Swift**

```
let label = UILabel()
label.highlighted = true
label.highlightedTextColor = UIColor.redColor()
```

**Swift 3**

```
let label = UILabel()
label.isHighlighted = true
label.highlightedTextColor = UIColor.red
```

# Section 2.18: Justify Text

**Swift**

```
let sampleText = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

// Create label
let label = UILabel(frame: CGRectMake(0, 0, view.frame.size.width, 400))
label.numberOfLines = 0
label.lineBreakMode = NSLineBreakMode.ByWordWrapping

// Justify text through paragraph style
let paragraphStyle = NSMutableParagraphStyle()
paragraphStyle.alignment = NSTextAlignment.Justified
let attributes = [NSParagraphStyleAttributeName: paragraphStyle, NSBaselineOffsetAttributeName:
NSNumber(float: 0)]
```

```
let attributedString = NSAttributedString(string: sampleText, attributes: attributes)
label.attributedText = attributedString
view.addSubview(label)
```

**Objective-C**

```
  NSString *sampleText = @"Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.";

    // Create label
    UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
400)];
    label.numberOfLines = 0;
    label.lineBreakMode = NSLineBreakByWordWrapping;


    // Justify text through paragraph style
    NSMutableParagraphStyle *paragraphStyle = [[NSMutableParagraphStyle alloc] init];
    paragraphStyle.alignment = NSTextAlignmentJustified;
    NSAttributedString *attributedString = [[NSAttributedString alloc] initWithString:sampleText
attributes:@{
                    NSParagraphStyleAttributeName : paragraphStyle,
                    NSBaselineOffsetAttributeName : [NSNumber numberWithFloat:0]
                    }];
    label.attributedText = attributedString;
    [self.view addSubview:label];
```

# Section 2.19: Dynamic label frame from unknown text length

Sometimes we have to resize a UILabel based on dynamic content where the text length is unknown. In this
example, width of the UILabel is fixed at 280 points and the height is infinite, lets say 9999. Estimating the frame
with respect to the text style and maximumLabelSize.

**Objective-C**

```
UILabel * label = [[UILabel alloc] init];

NSString *message = @"Some dynamic text for label";

//set the text and style if any.
label.text = message;

label.numberOfLines = 0;

CGSize maximumLabelSize = CGSizeMake(280, 9999); //280:max width of label and 9999-max height of
label.

// use font information from the UILabel to calculate the size
CGSize expectedLabelSize = [label sizeThatFits:maximumLabelSize];

//Deprecated in iOS 7.0
//CGSize expectedLabelSize = [message sizeWithFont:label.font constrainedToSize:maximumLabelSize
lineBreakMode:NSLineBreakByWordWrapping];

// create a frame that is filled with the UILabel frame data
CGRect newFrame = label.frame;
```

```
// resizing the frame to calculated size
newFrame.size.height = expectedLabelSize.height;

// put calculated frame into UILabel frame
label.frame = newFrame;
```

**Swift**

```
var message: String = "Some dynamic text for label"
//set the text and style if any.
label.text = message
label.numberOfLines = 0
var maximumLabelSize: CGSize = CGSize(width: 280, height: 9999)
var expectedLabelSize: CGSize = label.sizeThatFits(maximumLabelSize)
// create a frame that is filled with the UILabel frame data
var newFrame: CGRect = label.frame
// resizing the frame to calculated size
newFrame.size.height = expectedLabelSize.height
// put calculated frame into UILabel frame
label.frame = newFrame
```

# Chapter 3: UILabel text underlining

## Section 3.1: Underlining a text in a UILabel using Objective C

```
UILabel *label=[[UILabel alloc]initWithFrame:CGRectMake(0, 0, 320, 480)];
label.backgroundColor=[UIColor lightGrayColor];
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply Underlining"];
[attributedString addAttribute:NSUnderlineStyleAttributeName value:@1 range:NSMakeRange(0,
[attributedString length])];
[label setAttributedText:attributedString];
```

## Section 3.2: Underlining a text in UILabel using Swift

```
let label = UILabel.init(frame: CGRect(x: 0, y:0, width: 100, height: 40))
label.backgroundColor = .lightGray
let attributedString = NSMutableAttributedString.init(string: "Apply UnderLining")
attributedString.addAttribute(NSUnderlineStyleAttributeName, value: 1, range:
NSRange.init(location: 0, length: attributedString.length))
label.attributedText = attributedString
```

# Chapter 4: attributedText in UILabel

The current styled text that is displayed by the label.

You can add HTML text in `UILabel` using attributedText property or customized single `UILabel` text with different property

## Section 4.1: HTML text in UILabel

```
NSString * htmlString = @"<html><body> <b> Example bold text in HTML </b> </body></html>";
NSAttributedString * attrStr = [[NSAttributedString alloc] initWithData:[htmlString
dataUsingEncoding:NSUnicodeStringEncoding] options:@{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType } documentAttributes:nil error:nil];

UILabel * yourLabel = [[UILabel alloc] init];
yourLabel.attributedText = attrStr;
```

## Section 4.2: Set different property to text in single UILabel

The first step you need to preform is to create a `NSMutableAttributedString` object. The reason we create a `NSMutableAttributedString` instead of `NSAttributedString` is because it enables us to append string to it.

```
NSString *fullStr = @"Hello World!";
NSMutableAttributedString *attString =[[NSMutableAttributedString alloc]initWithString:fullStr];

// Finding the range of text.
NSRange rangeHello = [fullStr rangeOfString:@"Hello"];
NSRange rangeWorld = [fullStr rangeOfString:@"World!"];

// Add font style for Hello
[attString addAttribute: NSFontAttributeName
                value: [UIFont fontWithName:@"Copperplate" size:14]
                range: rangeHello];
// Add text color for Hello
[attString addAttribute: NSForegroundColorAttributeName
                value: [UIColor blueColor]
                range: rangeHello];

// Add font style for World!
[attString addAttribute: NSFontAttributeName
                value: [UIFont fontWithName:@"Chalkduster" size:20]
                range: rangeWorld];
// Add text color for World!
[attString addAttribute: NSForegroundColorAttributeName
                value: [UIColor colorWithRed:(66.0/255.0) green:(244.0/255.0) blue:(197.0/255.0)
alpha:1]
                range: rangeWorld];

// Set it to UILabel as attributedText
UILabel * yourLabel = [[UILabel alloc] initWithFrame:CGRectMake(10, 150, 200, 100)];
yourLabel.attributedText = attString;
[self.view addSubview:yourLabel];
```

Output :

HELLO World!

# Chapter 5: UIButton

UIButton : UIControl intercepts touch events and sends an action message to a target object when it's tapped. You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.

## Section 5.1: Creating a UIButton

UIButtons can be initialized in a frame:

**Swift**

```
let button = UIButton(frame: CGRect(x: x, y: y, width: width, height: height)
```

**Objective C**

```
UIButton *button = [[UIButton alloc] initWithFrame:CGRectMake(x, y, width, height)];
```

A specific type of UIButton can be created like this:

**Swift**

```
let button = UIButton(type: .Custom)
```

**Objective C**

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
```

where `type` is a `UIButtonType`:

```
enum UIButtonType : Int {
    case Custom
    case System
    case DetailDisclosure
    case InfoLight
    case InfoDark
    case ContactAdd
    static var RoundedRect: UIButtonType { get }
}
```

## Section 5.2: Attaching a Method to a Button

To add a method to a button, first create an action method:

**Objective-C**

```
-(void) someButtonAction{
    NSLog(@"Button is tapped");

}
```

**Swift**

```
func someButtonAction() {
```

```
        print("Button is tapped")
    }
```

Now to add this action method to your button, you have to write following line of code:

**Objective C**

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)
forControlEvents:UIControlEventTouchUpInside];
```

**Swift**

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:
.touchUpInside)
```

For ControlEvents, all members of ENUM UIControlEvents are valid.

# Section 5.3: Setting Font

**Swift**

```
myButton.titleLabel?.font =  UIFont(name: "YourFontName", size: 20)
```

**Objective C**

```
myButton.titleLabel.font = [UIFont fontWithName:@"YourFontName" size:20];
```

# Section 5.4: Set Image

**Swift**

```
button.setImage(UIImage(named:"test-image"), forState: .normal)
```

**Objective C**

```
[self.button setImage:[UIImage imageNamed:@"test-image"] forState:UIControlStateNormal];
```

**Multiple Control States**

You can also set an image for multiple UIControlStates, for example to set the same image for the Selected and Highlighted state:

**Swift**

```
button.setImage(UIImage(named:"test-image"), forState:[.selected, .highlighted])
```

**Objective C**

```
[self.button setImage:[UIImage imageNamed:@"test-image"]
forState:UIControlStateSelected|UIControlStateHighlighted];
```

# Section 5.5: Get UIButton's size strictly based on its text and font

To get the the exact size of a UIButton's text based on its font, use the function intrinsicContentSize.

**Swift**

```
button.intrinsicContentSize.width
```

**Objective-C**

```
button.intrinsicContentSize.width;
```

# Section 5.6: Disabling a UIButton

A button can be disabled by

**Swift**

```
myButton.isEnabled = false
```

**Objective-C:**

```
myButton.enabled = NO;
```

The button will become gray:



If you don't want the button appearance to change when disabled set `adjustsImageWhenDisabled` to `false` / `NO`

# Section 5.7: Set title

**Swift**

```
button.setTitle(titleString, forState: controlState)
```

**Objective C**

```
[button setTitle:(NSString *) forState:(UIControlState)];
```

To set the default title to "Hello, World!"

**Swift**

```
button.setTitle("Hello, World!", forState: .normal)
```

**Objective C**

```
[button setTitle:@"Hello, World!" forControlState:UIControlStateNormal];
```

# Section 5.8: Set title color

```
//Swift
button.setTitleColor(color, forControlState: controlState)

//Objective-C
[button setTitleColor:(nullable UIColor *) forState:(UIControlState)];
```

To set the title color to blue

```
//Swift
button.setTitleColor(.blue, for: .normal)

//Objective-C
[button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal]
```

# Section 5.9: Horizontally aligning contents

**Swift**

```
//Align contents to the left of the frame
button.contentHorizontalAlignment = .left

//Align contents to the right of the frame
button.contentHorizontalAlignment = .right

//Align contents to the center of the frame
button.contentHorizontalAlignment = .center

//Make contents fill the frame
button.contentHorizontalAlignment = .fill
```

**Objective C**

```
//Align contents to the left
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentLeft;

//Align contents to the right
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentRight;

//Align contents to the center
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentCenter;

//Align contents to fill the frame
button.contentHorizontalAlignment = UIControlContentHorizontalAlignmentFill;
```

# Section 5.10: Getting the title label

The underlying title label, if one exists, can be fetched using

**Swift**

```
var label: UILabel? = button.titleLabel
```

**Objective C**

```
UILabel *label = button.titleLabel;
```

This can be used to set the font of the title label, for example

**Swift**

```
button.titleLabel?.font = UIFont.boldSystemFontOfSize(12)
```

**Objective C**

```
button.titleLabel.font = [UIFont boldSystemFontOfSize:12];
```

# Section 5.11: Adding an action to an UIButton via Code (programmatically)

To add a method to a button, first create an action method:

**Objective-C**

```
-(void)someButtonAction:(id)sender {
  // sender is the object that was tapped, in this case its the button.
    NSLog(@"Button is tapped");
}
```

**Swift**

```
func someButtonAction() {
    print("Button is tapped")
}
```

Now to add this action method to your button, you have to write following line of code:

**Objective C**

```
[yourButtonInstance addTarget:self action:@selector(someButtonAction)
forControlEvents:UIControlEventTouchUpInside];
```

**Swift**

```
yourButtonInstance.addTarget(self, action: #selector(someButtonAction), forControlEvents:
.TouchUpInside)
```

For ControlEvents parameter, all members of ENUM [UIControlEvents](UIControlEvents) are valid.

# Chapter 6: UIDatePicker

## Section 6.1: Create a Date Picker

**Swift**

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200)
```

**Objective-C**

```
UIDatePicker *datePicker = [[UIDatePicker alloc] initWithFrame:CGRectMake(x: 0, y: 0, width: 320,
height: 200)];
```

## Section 6.2: Setting Minimum-Maximum Date

You can set the minimum and the maximum date that UIDatePicker can show.

**Minimum date**

```
[datePicker setMinimumDate:[NSDate date]];
```

**Maximum date**

```
[datePicker setMaximumDate:[NSDate date]];
```

## Section 6.3: Modes

UIDatePicker has various picker modes.

```
enum UIDatePickerMode : Int {
    case Time
    case Date
    case DateAndTime
    case CountDownTimer
}
```

- `Time` - The date picker displays hours, minutes, and (optionally) an AM/PM designation.
- `Date` - The date picker displays months, days of the month, and years.
- `DateAndTime` - The date picker displays dates (as unified day of the week, month, and day of the month values) plus hours, minutes, and (optionally) an AM/PM designation.
- `CountDownTimer` - The date picker displays hour and minute values, for example [ 1 | 53 ]. The application must set a timer to fire at the proper interval and set the date picker as the seconds tick down.

Setting property datePickerMode

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200)
datePicker.datePickerMode = .Date
```

## Section 6.4: Setting minute interval

You can change property `minuteInterval` to set the interval displayed by the minutes wheel. The default value is 1, the maximum value is 30.

```
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200)
datePicker.minuteInterval = 15
```

# Section 6.5: Count Down Duration

The `NSTimeInterval` value of this property indicates the seconds from which the date picker in countdown-timer mode counts down. If the mode of the date picker is not `CountDownTimer`, this value is ignored. Maximum value is 86,399 seconds (23:59)

```swift
let datePicker = UIDatePicker(frame: CGRect(x: 0, y: 0, width: 320, height: 200)
datePicker.countDownDuration = 60 * 60
```

# Chapter 7: UILocalNotification

Local notifications allow your app to notify the user about content which does not require the use of a server.

Unlike remote notifications which are triggered from a server, local notifications are scheduled and triggered within an app. Notifications in general are targeted to increase user interaction with the app, inviting or tempting the user to open and interact with it.

UILocalNotification was deprecated in iOS 10. Use the UserNotifications framework instead.

## Section 7.1: Scheduling a local notification

Make sure you see Registering for local notifications in order for this to work:

**Swift**

```swift
let notification = UILocalNotification()
notification.alertBody = "Hello, local notifications!"
notification.fireDate = NSDate().dateByAddingTimeInterval(10) // 10 seconds after now
UIApplication.sharedApplication().scheduleLocalNotification(notification)
```

**Objective-C**

```objc
UILocalNotification *notification = [[UILocalNotification alloc] init];
notification.alertBody = @"Hello, local notifications!";
notification.fireDate = [NSDate dateWithTimeIntervalSinceNow:10]; // 10 seconds after now
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

To see the notification in the iOS Simulator, type ^⌘H (control-command-H) to go home and then type ⌘L (command-L) to lock the device. Wait a few seconds, and the notification should appear (this appearance will vary depending on notification type discussed in "Registering for local notifications"):
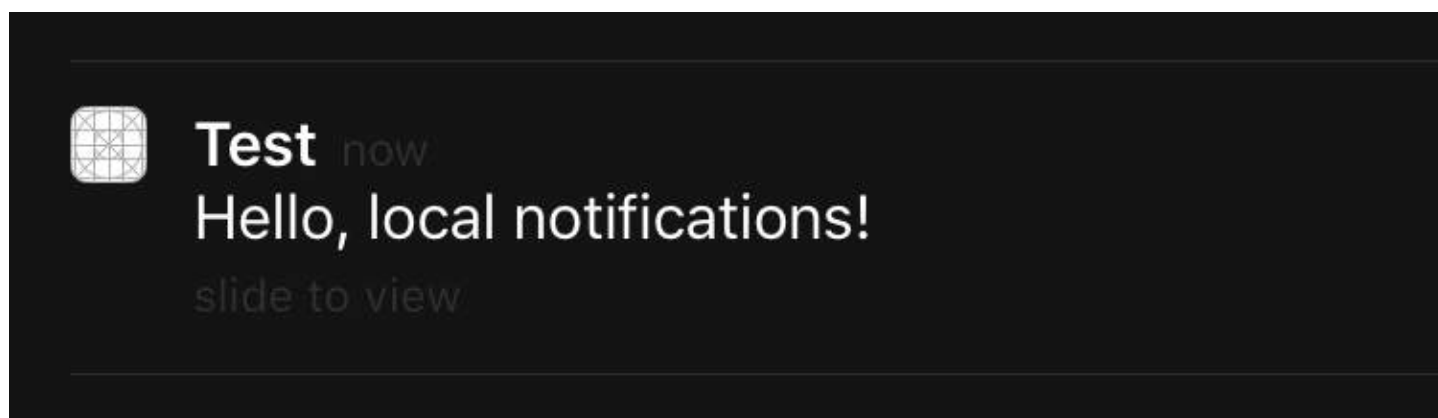


Swipe on the notification to get back to the app (note that if you called this in the first view controller's `viewDidLoad`, `viewWillAppear`, `viewDidAppear`, etc., the notification will be scheduled again).

## Section 7.2: Presenting a local notification immediately

If you want to show local notification immediately, you should call:

**Swift 3**

```swift
UIApplication.shared.presentLocalNotificationNow(notification)
```

**Swift 2**

```swift
UIApplication.sharedApplication().presentLocalNotificationNow(notification)
```

**Objective-C**

```objc
[[UIApplication sharedApplication] presentLocalNotificationNow:notification];
```

An advantage of using this is so you won't have to set the `fireDate` and `timeZone` properties of your `UILocalNotification` object.

# Section 7.3: Managing local notifications using UUID

Often times you will need to be able to manage your notifications, by being able to keep track of them and cancel them.

**Track a notification**

You can assign a UUID (universally unique identifier) to a notification, so you can track it:

**Swift**

```swift
let notification = UILocalNotification()
let uuid = NSUUID().uuidString
notification.userInfo = ["UUID": uuid]
UIApplication.shared.scheduleLocalNotification(notification)
```

**Objective-C**

```objc
UILocalNotification *notification = [[UILocalNotification alloc] init];
NSString *uuid = [[NSUUID UUID] UUIDString];
notification.userInfo = @{ @"UUID": uuid };
[[UIApplication sharedApplication] scheduleLocalNotification:notification];
```

**Cancel a notification**

To cancel a notification, we first get a list of all the notifications and then find the one with a matching UUID. Finally, we cancel it.

**Swift**

```swift
let scheduledNotifications = UIApplication.shared.scheduledLocalNotifications

guard let scheduledNotifications = scheduledNotifications else {
    return
}

for notification in scheduledNotifications where "\(notification.userInfo!["UUID"]!)" ==
UUID_TO_CANCEL {
    UIApplication.sharedApplication().cancelLocalNotification(notification)
}
```

**Objective-C**

```objc
NSArray *scheduledNotifications = [[UIApplication sharedApplication] scheduledLocalNotifications];

for (UILocalNotification *notification in scheduledNotifications) {
```

```
    if ([[notification.userInfo objectForKey:"UUID"] compare: UUID_TO_CANCEL]) {
        [[UIApplication sharedApplication] cancelLocalNotification:notification];
        break;
    }
}
```

You would probably want to store all of these UUID's in Core Data or Realm.

# Section 7.4: Registering for local notifications

Version ≥ iOS 8

In order to present local notifications to the user, you have to register your app with the device:

**Swift**

```
let settings = UIUserNotificationSettings(forTypes: [.Badge, .Sound, .Alert], categories: nil)
UIApplication.sharedApplication().registerUserNotificationSettings(settings)
```

**Objective-C**

```
UIUserNotificationSettings *settings = [UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeBadge | UIUserNotificationTypeSound |
UIUserNotificationTypeAlert) categories:nil];
[[UIApplication sharedApplication] registerUserNotificationSettings:settings];
```

This will present an alert the first time it is called:



> "Test" Would Like to Send
> You Notifications
>
> Notifications may include alerts,
> sounds, and icon badges. These can
> be configured in Settings.
>
> Don't Allow          OK
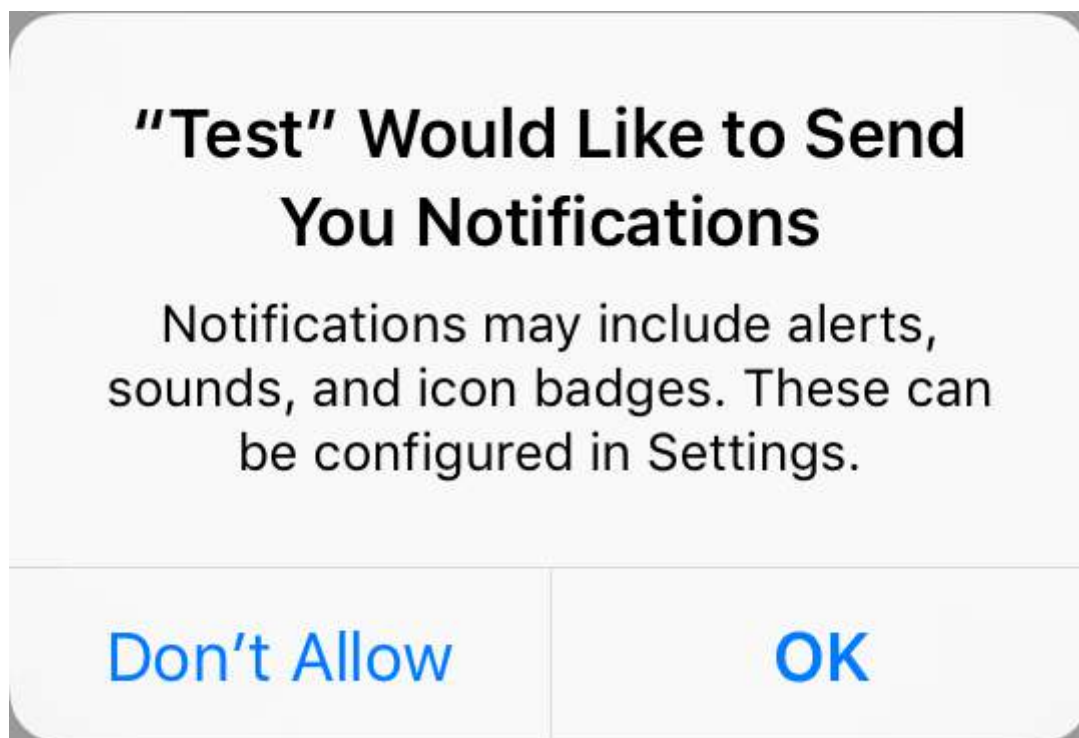
Regardless of what the user chooses, the alert will not show up again and changes will have to be initiated by the user in Settings.

# Section 7.5: what's new in UILocalNotification with iOS10

You can use UILocalNotification, old APIs also works fine with iOS10, but we had better use the APIs in the User

---

Notifications framework instead. There are also some new features, you can only use with iOS10 User Notifications framework.

This also happens to Remote Notification, for more information: [Here](#).

New Features:

1. Now you can either present alert, sound or increase badge while the app is in foreground too with iOS 10
2. Now you can handle all event in one place when user tapped (or slided) the action button, even while the app has already been killed.
3. Support 3D touch instead of sliding gesture.
4. Now you can remove specifical local notification just by one row code.
5. Support Rich Notification with custom UI.

It is really easy for us to convert `UILocalNotification` APIs to iOS10 User Notifications framework APIs, they are really similar.

I write a Demo here to show how to use new and old APIs at the same time: **iOS10AdaptationTips** .

For example,

With Swift implementation:

1. import UserNotifications

```
///   Notification become independent from UIKit
import UserNotifications
```

2. request authorization for localNotification

```
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
    // Enable or disable features based on authorization.
}
```

3. schedule localNotification

4. update application icon badge number

```
@IBAction  func triggerNotification(){
    let content = UNMutableNotificationContent()
    content.title = NSString.localizedUserNotificationString(forKey: "Elon said:", arguments: nil)
    content.body = NSString.localizedUserNotificationString(forKey: "Hello Tom?Get up, let's play with Jerry!", arguments: nil)
    content.sound = UNNotificationSound.default()
    content.badge = UIApplication.shared().applicationIconBadgeNumber + 1;
    content.categoryIdentifier = "com.elonchan.localNotification"
    // Deliver the notification in five seconds.
    let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60.0, repeats: true)
    let request = UNNotificationRequest.init(identifier: "FiveSecond", content: content, trigger: trigger)

    // Schedule the notification.
    let center = UNUserNotificationCenter.current()
    center.add(request)
}
```

```
@IBAction func stopNotification(_ sender: AnyObject) {
    let center = UNUserNotificationCenter.current()
    center.removeAllPendingNotificationRequests()
    // or you can remove specific notification:
    // center.removePendingNotificationRequests(withIdentifiers: ["FiveSecond"])
}
```

Objective-C implementation:

1. import UserNotifications

```
// Notifications are independent from UIKit
#import <UserNotifications/UserNotifications.h>
```

2. request authorization for localNotification

```
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center requestAuthorizationWithOptions:(UNAuthorizationOptionBadge |
UNAuthorizationOptionSound | UNAuthorizationOptionAlert)
                      completionHandler:^(BOOL granted, NSError * _Nullable error) {
                          if (!error) {
                              NSLog(@"request authorization succeeded!");
                              [self showAlert];
                          }
                      }];
```

3. schedule localNotification

4. update application icon badge number

```
UNMutableNotificationContent *content = [[UNMutableNotificationContent alloc] init];
content.title = [NSString localizedUserNotificationStringForKey:@"Elon said:"
                                                      arguments:nil];
content.body = [NSString localizedUserNotificationStringForKey:@"Hello Tom?Get up, let's play
with Jerry!"
                                                      arguments:nil];
content.sound = [UNNotificationSound defaultSound];

// 4. update application icon badge number
content.badge = [NSNumber numberWithInteger:([UIApplication
sharedApplication].applicationIconBadgeNumber + 1)];
// Deliver the notification in five seconds.
UNTimeIntervalNotificationTrigger *trigger = [UNTimeIntervalNotificationTrigger
                                              triggerWithTimeInterval:5.f
                                              repeats:NO];
UNNotificationRequest *request = [UNNotificationRequest requestWithIdentifier:@"FiveSecond"
                                                                      content:content
                                                                      trigger:trigger];

/// 3. schedule localNotification
UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
[center addNotificationRequest:request withCompletionHandler:^(NSError * _Nullable error) {
    if (!error) {
        NSLog(@"add NotificationRequest succeeded!");
    }
}];
```

Go to here for more information: **iOS10AdaptationTips**.

#updated

> Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'time interval must be at least 60 if repeating'

```
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 60, repeats: true)
```

# Section 7.6: Responding to received local notification

**IMPORTANT: This delegate method is only called in the foreground.**

**Swift**

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {

}
```

**Objective-C**

```
- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification
*)notification {

}
```

This method is generally overridden in the AppDelegate, which conforms to the UIApplicationDelegate protocol.

# Section 7.7: Register and Schedule Local Notification in Swift 3.0 (iOS 10)

**Registration**

in **AppDelegate**

```
import UserNotifications
```

in **didFinishLaunchingWithOptions** method,

```
UNUserNotificationCenter.current().requestAuthorization(options: [.alert,.sound,.badge]) {
(granted, error) in

// Here you can check Request is Granted or not.

}
```

Create and Schedule notification.

```
    let content = UNMutableNotificationContent()
    content.title = "10 Second Notification Demo"
    content.subtitle = "From Wolverine"
    content.body = "Notification after 10 seconds - Your pizza is Ready!!"
    content.categoryIdentifier = "myNotificationCategory"

    let trigger = UNTimeIntervalNotificationTrigger(
```

```
        timeInterval: 10.0,
        repeats: false)

    let request = UNNotificationRequest(
        identifier: "10.second.message",
        content: content,
        trigger: trigger
    )
    UNUserNotificationCenter.current().add(request, withCompletionHandler: nil)
```

Where ever this part of code is triggered, If you have allowed Notification Permission, you will receive an notification.

To test it properly, Make sure your application in Background mode.

# Chapter 8: UIImage

## Section 8.1: Creating UIImage

**With local image**

**Swift**

```swift
let image = UIImage(named: "imageFromBundleOrAsset")
```

**Objective-C**

```objc
UIImage *image = [UIImage imageNamed:@"imageFromBundleOrAsset"];
```

> **Note**
>
> The method <u>imageNamed</u> caches the image's contents to memory. Loading many large images that way can cause low memory warnings which can lead the app to be terminated. This can be fixed by utilising the method `imageWithContentsOfFile` of `UIImage`, which doesn't use caching.

**With NSData**

**Swift**

```swift
let imageData = Data(base64Encoded: imageString, options:
Data.Base64DecodingOptions.ignoreUnknownCharacters)

let image = UIImage(data: imageData!)
```

**With UIColor**

**Swift**

```swift
let color = UIColor.red
let size = CGSize(width: 200, height: 200)

UIGraphicsBeginImageContextWithOptions(size, false, 0.0)
UIGraphicsGetCurrentContext()!.setFillColor(color.cgColor)
UIGraphicsGetCurrentContext()!.fill(CGRect(origin: .zero, size: size))
let colorImage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

**Objective-C**

```objc
UIColor *color=[UIColor redColor];
CGRect frame = CGRectMake(0, 0, 80, 100);
UIGraphicsBeginImageContext(frame.size);
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetFillColorWithColor(context, [color CGColor]);
CGContextFillRect(context, frame);
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
```

**With file content**

**Objective-C**

Example:

```objc
UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

Using Array:

Example:

```
NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // check if a file exists
    if ([UIImage imageNamed:fileName]) {
        // if it exists, add it to the array
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle
mainBundle]pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}
```

//Using image array for animations here:

```
self.myImageView.animationImages = imageArray;
```

# Section 8.2: Comparing Images

> The `isEqual:` method is the only reliable way to determine whether two images contain the same image data. The image objects you create may be different from each other, even when you initialize them with the same cached image data. The only way to determine their equality is to use the `isEqual:` method, which compares the actual image data. Listing 1 illustrates the correct and incorrect ways to compare images.
>
> Source: Apple Documentation

**Swift**

```swift
// Load the same image twice.
let image1 = UIImage(named: "MyImage")
let image2 = UIImage(named: "MyImage")

// The image objects may be different, but the contents are still equal
if let image1 = image1, image1.isEqual(image2) {
    // Correct. This technique compares the image data correctly.
}

if image1 == image2 {
    // Incorrect! Direct object comparisons may not work.
}
```

**Objective-C**

```objc
// Load the same image twice.
UIImage* image1 = [UIImage imageNamed:@"MyImage"];
UIImage* image2 = [UIImage imageNamed:@"MyImage"];

// The image objects may be different, but the contents are still equal
if ([image1 isEqual:image2]) {
    // Correct. This technique compares the image data correctly.
}

if (image1 == image2) {
    // Incorrect! Direct object comparisons may not work.
}
```

# Section 8.3: Gradient Image with Colors

Creating Gradient `UIImage` with colors in `CGRect`

Swift:

```swift
extension UIImage {
    static func gradientImageWithBounds(bounds: CGRect, colors: [CGColor]) -> UIImage {
        let gradientLayer = CAGradientLayer()
        gradientLayer.frame = bounds
        gradientLayer.colors = colors

        UIGraphicsBeginImageContext(gradientLayer.bounds.size)
        gradientLayer.render(in: UIGraphicsGetCurrentContext()!)
        let image = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return image!
    }
}
```

Usage:

```swift
let image = UIImage.gradientImageWithBounds(CGRect(x: 0, y: 0, width: 200, height: 200), colors:
[UIColor.yellowColor().CGColor, UIColor.blueColor().CGColor])
```

Objective-C:

```objc
+ (UIImage *)gradientImageWithBounds:(CGRect)bounds colors:(NSArray *)colors {
    CAGradientLayer *gradientLayer = [CAGradientLayer layer];
    gradientLayer.frame = bounds;
    gradientLayer.colors = colors;

    UIGraphicsBeginImageContext(gradientLayer.bounds.size);
    [gradientLayer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return image;
}
```

# Section 8.4: Convert UIImage to/from base64 encoding

**Encoding**

```swift
//convert the image to NSData first
let imageData:NSData = UIImagePNGRepresentation(image)!
// convert the NSData to base64 encoding
let strBase64:String = imageData.base64EncodedStringWithOptions(.Encoding64CharacterLineLength)
```

**Decoding**

```swift
let dataDecoded:NSData = NSData(base64EncodedString: strBase64, options:
NSDataBase64DecodingOptions(rawValue: 0))!
let decodedimage:UIImage = UIImage(data: dataDecoded)!
```

# Section 8.5: Take a Snapshot of a UIView

```swift
//Here self.webView is the view whose screenshot I need to take
```

```
//The screenshot is saved in jpg format in the application directory to avoid any loss of quality
in retina display devices i.e. all current devices running iOS 10
UIGraphicsBeginImageContextWithOptions(self.webView.bounds.size, NO, [UIScreen mainScreen].scale);
[self.webView.layer renderInContext:UIGraphicsGetCurrentContext()];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
NSString  *jpgPath = [NSHomeDirectory() stringByAppendingPathComponent:@"Documents/Test.jpg"];
[UIImageJPEGRepresentation(image, 1.0) writeToFile:jpgPath atomically:YES];
UIImage *pop=[[UIImage alloc]initWithContentsOfFile:jpgPath];
//pop is the final image in jpg format and high quality with the exact resolution of the view you
selected in pixels and not just points
```

## Section 8.6: Change UIImage Color

**Swift** Add this extension to UIImage :

```
extension UIImage {
    func maskWithColor(color: UIColor) -> UIImage? {

        let maskImage = self.CGImage
        let width = self.size.width
        let height = self.size.height
        let bounds = CGRectMake(0, 0, width, height)

        let colorSpace = CGColorSpaceCreateDeviceRGB()
        let bitmapInfo = CGBitmapInfo(rawValue: CGImageAlphaInfo.PremultipliedLast.rawValue)
        let bitmapContext = CGBitmapContextCreate(nil, Int(width), Int(height), 8, 0, colorSpace,
bitmapInfo.rawValue) //needs rawValue of bitmapInfo

        CGContextClipToMask(bitmapContext, bounds, maskImage)
        CGContextSetFillColorWithColor(bitmapContext, color.CGColor)
        CGContextFillRect(bitmapContext, bounds)

        //is it nil?
        if let cImage = CGBitmapContextCreateImage(bitmapContext) {
            let coloredImage = UIImage(CGImage: cImage)

            return coloredImage

        } else {
            return nil
        }
    }
}
```

Then, to change the color of your UIImage

```
my_image.maskWithColor(UIColor.blueColor())
```

Found at this link

## Section 8.7: Apply UIColor to UIImage

Use same UIImage with multiple theme base app by just applying UIColor to UIImage instance as following.

```
// *** Create an UIImage instance with RenderingMode AlwaysTemplate ***
UIImage *imgMenu = [[UIImage imageNamed:@"iconMenu"]
imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate];
```

```
// *** Now Apply `tintColor` to `UIImageView` of UIImageView or UIButton and convert image in given
color ***
[btn setImage:imgMenu forState:UIControlStateNormal]; // Set UIImage in UIButton.

[button.imageView setTintColor:[UIColor blueColor]]; // It changes image color of UIButton to blue
color
```

Now lets say you want to do same with UIImageView then use following code

```
[imageView setImage:imgMenu]; // Assign UIImage to UIImageView
[imageView setTintColor:[UIColor greenColor]]; // Change imageview image color to green.
[imageView setTintColor:[UIColor redColor]]; // Change imageview image color to red.
```

# Section 8.8: Creating and Initializing Image Objects with file contents

Creating and returning an image object by loading the image data from the file at the specified path.

Example:

```
 UIImage *image = [UIImage imageWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:[cellCountry objectForKey:@"Country_Flag"] ofType:nil]];
```

Using Array:

Example

```
 NSMutableArray *imageArray = [[NSMutableArray alloc] init];

for (int imageNumber = 1; self.myPhoto != nil; imageNumber++) {
    NSString *fileName = [NSString stringWithFormat:@"%@.jpg", self.myPhoto];

    // check if a file exists
    if ([UIImage imageNamed:fileName]) {
        // if it exists, add it to the array
        [imageArray addObject:[UIImage imageWithContentsOfFile:[[NSBundle
mainBundle]pathForResource:[NSString stringWithFormat:@"%@", fileName] ofType:@""]]];
    } else {
        break;
    }
}

//Using image array for animations here
self.myImageView.animationImages = imageArray;
```

# Section 8.9: Resizable image with caps

In the example of a message bubble illustrated below: the corners of the image should remain unchanged which is specified by `UIEdgeInsets`, but the borders and center of the image should expand to cover the new size.

```
let insets = UIEdgeInsetsMake(12.0, 20.0, 22.0, 12.0)
let image = UIImage(named: "test")
image?.resizableImageWithCapInsets(insets, resizingMode: .Stretch)
```

## Section 8.10: Gradient Background Layer for Bounds

```
+ (CALayer *)gradientBGLayerForBounds:(CGRect)bounds colors:(NSArray *)colors
{
    CAGradientLayer * gradientBG = [CAGradientLayer layer];
    gradientBG.frame = bounds;
    gradientBG.colors = colors;
    return gradientBG;
}
```

# Chapter 9: Convert NSAttributedString to UIImage

## Section 9.1: NSAttributedString to UIImage Conversion

**Objective-C**

```objc
NSMutableAttributedString *str = [[NSMutableAttributedString alloc] initWithString:@"Hello. That
is a test attributed string."];
[str addAttribute:NSBackgroundColorAttributeName value:[UIColor yellowColor]
range:NSMakeRange(3,5)];
[str addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(10,7)];
[str addAttribute:NSFontAttributeName value:[UIFont fontWithName:@"HelveticaNeue-Bold" size:20.0]
range:NSMakeRange(20, 10)];
UIImage *customImage = [self imageFromAttributedString:str];
```

The function `imageFromAttributedString` is as defined below:

```objc
- (UIImage *)imageFromAttributedString:(NSAttributedString *)text
{
    UIGraphicsBeginImageContextWithOptions(text.size, NO, 0.0);

    // draw in context
    [text drawAtPoint:CGPointMake(0.0, 0.0)];

    // transfer image
    UIImage *image = [UIGraphicsGetImageFromCurrentImageContext()
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];
    UIGraphicsEndImageContext();

    return image;
}
```

# Chapter 10: UIImagePickerController

UIImagePickerController provides an almost out of the box solution to allow the user to select an image from their device or take a picture with the camera and then present that image. By conforming to the UIImagePickerControllerDelegate, you can create logic that specifies in your app how to present the image and what to do with it (using didFinishPickingMediaWithInfo) and also what to do if the user declines to select an image or take a picture (using imagePickerControllerDidCancel).

## Section 10.1: Generic usage of UIImagePickerController

Step 1: Create the controller, set the delegate, and conform to the protocol

```swift
//Swift
class ImageUploadViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {

    let imagePickerController = UIImagePickerController()

    override func viewDidLoad() {
        super.viewDidLoad()
        imagePickerController.delegate = self
    }
}
```

```objectivec
//Objective-C
@interface ImageUploadViewController : UIViewController
<UIImagePickerControllerDelegate,UINavigationControllerDelegate> {

    UIImagePickerController *imagePickerController;

}

@end

@implementation ImageUploadViewController

- (void)viewDidLoad {

    [super viewDidLoad];

    imagePickerController.delegate = self;

}

@end
```

note: We actually will not implement anything defined in `UINavigationControllerDelegate`, but `UIImagePickerController` inherits from `UINavigationController` and changes the behavior of `UINavigationController`. Therefore, we still need to say our view controller conforms to `UINavigationControllerDelegate`.

Step 2: Whenever you need to show `UIImagePickerController`:

```swift
//Swift
self.imagePickerController.sourceType = .Camera // options: .Camera , .PhotoLibrary ,
.SavedPhotosAlbum
self.presentViewController(self.imagePickerController, animated: true, completion: nil)
```

```
//Objective-C
imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera; // options:
UIImagePickerControllerSourceTypeCamera, UIImagePickerControllerSourceTypePhotoLibrary,
UIImagePickerControllerSourceTypeSavedPhotosAlbum
[self presentViewController:imagePickerController animated:YES completion:nil];
```

Step 3: Implement the delegate methods:

```
//Swift
func imagePickerController(picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[String : AnyObject]) {
    if let pickedImage = info[UIImagePickerControllerOriginalImage] as? UIImage {
        // Your have pickedImage now, do your logic here
    }
    self.dismissViewControllerAnimated(true, completion: nil)
}

func imagePickerControllerDidCancel(picker: UIImagePickerController) {
    self.dismissViewControllerAnimated(true, completion: nil)
}

//Objective-C
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {

    UIImage *pickedImage = info[UIImagePickerControllerOriginalImage];

    if (pickedImage) {

        //You have pickedImage now, do your logic here

    }

    [self dismissViewControllerAnimated:YES completion:nil];

}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {

    [self dismissViewControllerAnimated:YES completion:nil];

}
```

# Chapter 11: UIImageView

## Section 11.1: UIImage masked with Label

This makes image masked to the shape of the letters of the label:

**Objective-C**
```objc
self.maskImage.layer.mask = self.maskLabel.layer;
self.maskImage.layer.masksToBounds = YES;
```

**Swift 3**
```swift
maskImageView.mask = maskLabel
maskImageView.masksToBounds = true
```

Here is the result:



## Section 11.2: Making an image into a circle or rounded

This example shows, how to make a `UIView` or `UIImageView`, rounded with some radius like this:
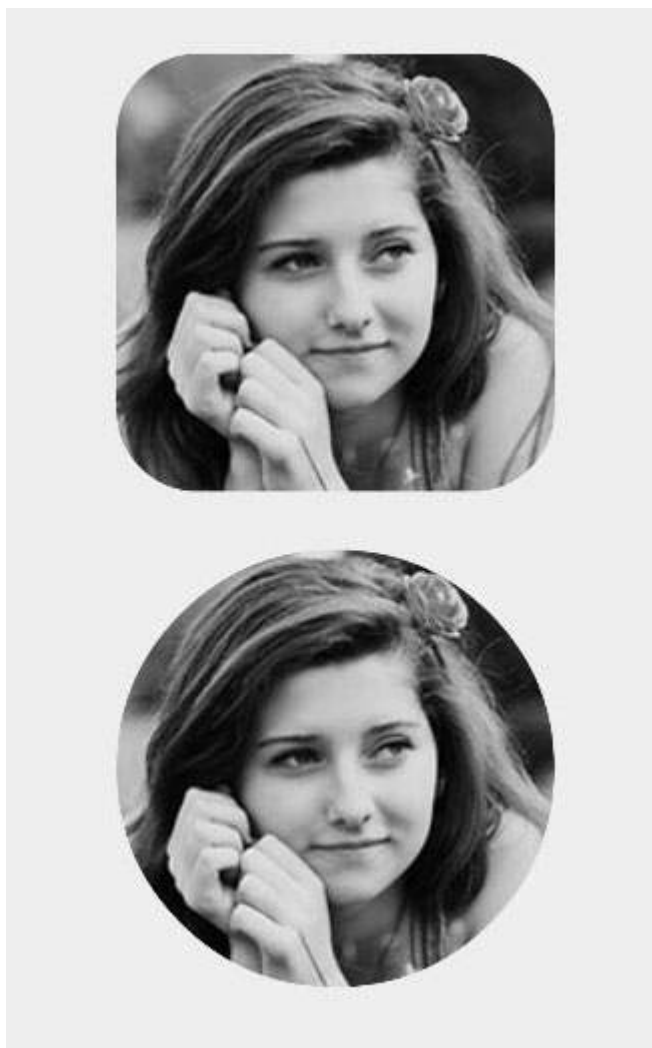
**Objective-C**

```
someImageView.layer.cornerRadius = CGRectGetHeight(someImageView.frame) / 2;
someImageView.clipsToBounds = YES;
```

**Swift**

```
someImageView.layer.cornerRadius = someImageView.frame.height/2
// this should alleviate the performance hit that adding transparency may cause - see
http://stackoverflow.com/a/6254531/189804
// Be sure to check scrolling performance with Instruments if you take this approach.
someImageView.layer.shouldRasterize = true
someImageView.clipsToBounds = true // All parts of the image that are outside its bounds (the
frame) are cut out (makes the rounded corners visible)
```

It is suggested that if you use autolayout that you put the `someImageView.layer.cornerRadius` code in `viewDidLayoutSubviews`. This will allow the image's `cornerRadius` to update if the image changes size.

```
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    someImageView.layer.cornerRadius = someImageView.frame.size.width/2
    someImageView.layer.masksToBounds = true
}
```

# Section 11.3: How the Mode property affects an image

The [content mode property](#) of a view tells how its content should be laid out. In the Interface Builder, the various modes can be selected in the Attributes Inspector.
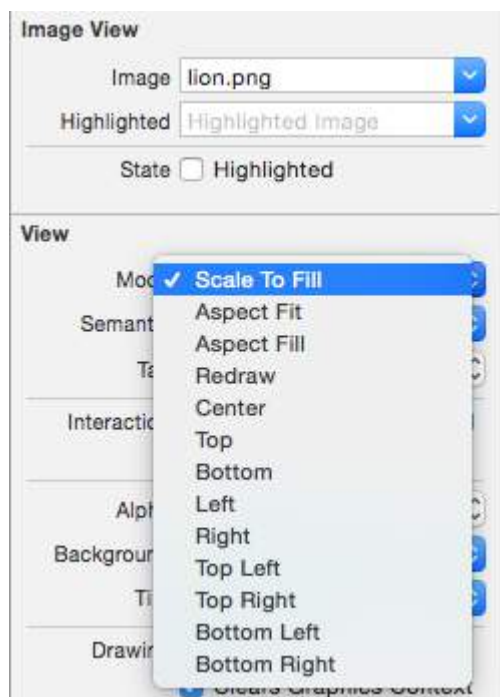


Let's use two image views to see how the various modes work.

**Scale to Fill**



The image heights and widths are stretched to match the size of the `UIImageView`.

**Aspect Fit**



The longest side (either height or width) of the image is stretched to match the view. This makes the image as big as possible while still showing the entire image and not distorting the height or width. (I set the `UIImageView` background to blue so that its size is clear.)

**Aspect Fill**

The shortest side (either height or width) of the image is stretched to match the view. Like "Aspect Fit", the proportions of the image are not distorted from their original aspect ratio.

**Redraw**



Redraw is only for custom views that need to do their own scaling and resizing. We aren't using a custom view, so we shouldn't use Redraw. Notice that here `UIImageView` just gives us the same result as Scale to Fill, but it is doing more work behind the scenes.

About Redraw, the [Apple documentation](#) says:

> Content modes are good for recycling the contents of your view, but you can also set the content mode to the `UIViewContentModeRedraw` value when you specifically want your custom views to redraw themselves during scaling and resizing operations. Setting your view's content mode to this value forces the system to call your view's `drawRect:` method in response to geometry changes. In general, you should avoid using this value whenever possible, and you should certainly not use it with the standard system views.

**Center**

The image is centered in the view, but the length and width of the image are not stretched.

**Top**
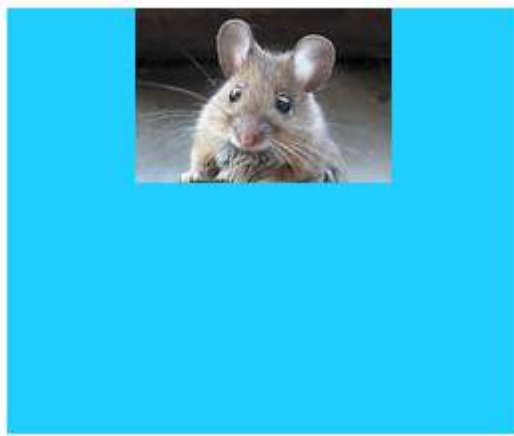


The top edge of the image is centered horizontally at the top of the view, and the length and width of the image are not stretched.
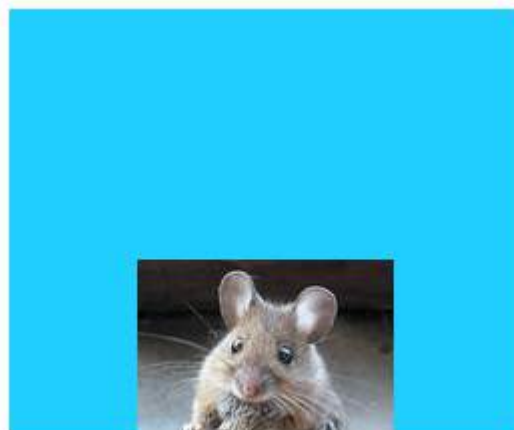
**Bottom**



The bottom edge of the image is centered horizontally at the bottom of the view, and the length and width of the image are not stretched.

**Left**

The left edge of the image is centered vertically at the left of the view, and the length and width of the image are not stretched.

**Right**
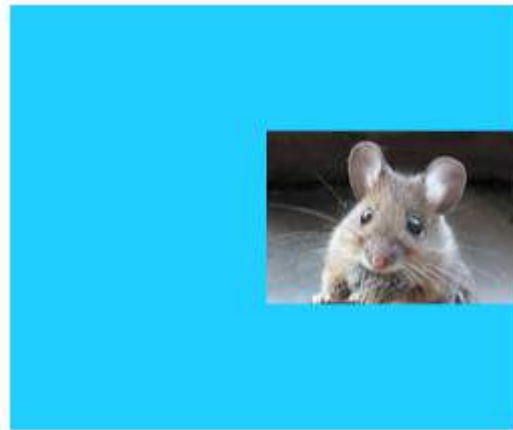


The right edge of the image is centered vertically at the right of the view, and the length and width of the image are not stretched.

**Top Left**
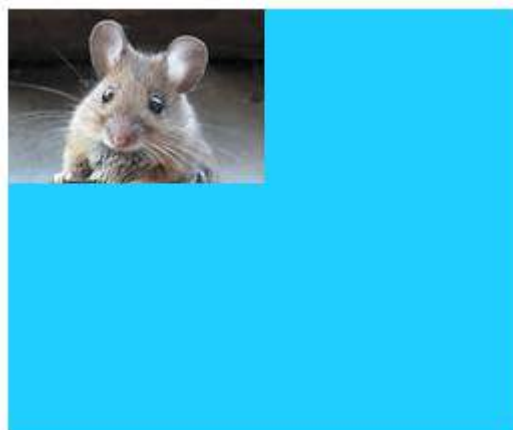


The top left corner of the image is placed at the top left corner of the view. The length and width of the image are not stretched.

**Top Right**
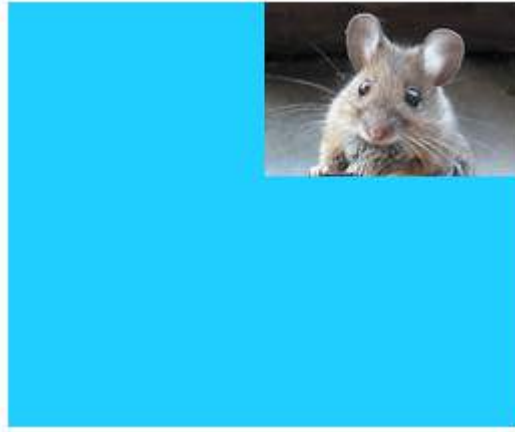
The top right corner of the image is placed at the top right corner of the view. The length and width of the image are not stretched.

**Bottom Left**
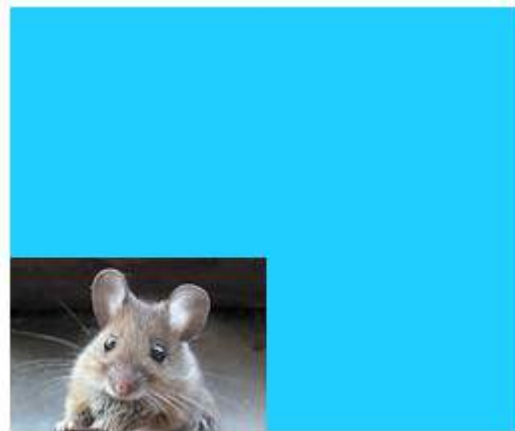


The bottom left corner of the image is placed at the bottom left corner of the view. The length and width of the image are not stretched.

**Bottom Right**
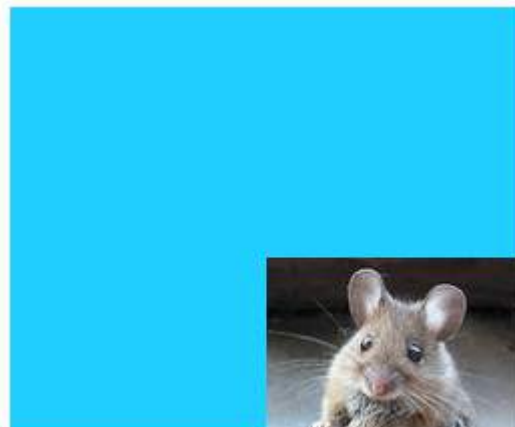


The bottom right corner of the image is placed at the bottom right corner of the view. The length and width of the image are not stretched.

**Notes**

- This example comes originally from [here](#).

- If the content (in our case the image) is the same size as the view (in our case the `UIImageView`), then changing the content mode will make no noticeable difference.

- See this and this question for a discussion about content modes for views other than `UIImageView`.

- In Swift, to set to set the content mode programmatically you do the following:

```swift
imageView.contentMode = UIViewContentMode.scaleToFill
imageView.contentMode = UIViewContentMode.scaleAspectFit
imageView.contentMode = UIViewContentMode.scaleAspectFill
imageView.contentMode = UIViewContentMode.redraw
imageView.contentMode = UIViewContentMode.center
imageView.contentMode = UIViewContentMode.top
imageView.contentMode = UIViewContentMode.bottom
imageView.contentMode = UIViewContentMode.left
imageView.contentMode = UIViewContentMode.right
imageView.contentMode = UIViewContentMode.topLeft
imageView.contentMode = UIViewContentMode.topRight
imageView.contentMode = UIViewContentMode.bottomLeft
imageView.contentMode = UIViewContentMode.bottomRight
```

# Section 11.4: Animating a UIImageView

You can animate a `UIImageView` by quickly displaying images on it in a sequence using the `UIImageView`'s animation properties:

```swift
imageView.animationImages = [UIImage(named: "image1")!,
                             UIImage(named: "image2")!,
                             UIImage(named: "image3")!,
                             UIImage(named: "image4")!,
                             UIImage(named: "image5")!,
                             UIImage(named: "image6")!,
                             UIImage(named: "image7")!,
                             UIImage(named: "image8")!]
imageView.animationDuration = 0.3
imageView.animationRepeatCount = 1
```

The `animationImages` property is an `Array` of UIImages that is run through from top to bottom when the animation is triggered.

The `animationDuration` property is a `Double` saying how many seconds the animation will run for.

The `animationRepeatCount` property is an `Int` that says how many times the animation will run.

To start and stop the animation, you can call the appropriate methods to do so:

```swift
imageView.startAnimating()
imageView.stopAnimating()
```

There is method `isAnimating()` which returns a `Boolean` value indicating whether the animation is running at a moment or not.

Please note that this's not a very efficient way to create animations: it's quite slow and resource-consuming. Consider using Layers or Sprites for better results

# Section 11.5: Create a UIImageView

To create a `UIImageView` programmatically, all you need to do is create an instance of `UIImageView`:

```
//Swift
let imageView = UIImageView()

//Objective-C
UIImageView *imageView = [[UIImageView alloc] init];
```

You can set the size and position of the `UIImageView` with a `CGRect`:

```
//Swift
imageView.frame = CGRect(x: 0, y: 0, width: 200, height: 200)

//Objective-C
imageView.frame = CGRectMake(0,0,200,200);
```

Or you can set the size during initialization:

```
//Swift
UIImageView(frame: CGRect(x: 0, y: 0, width: 200, height: 200))

//Objective-C
UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0,0,200,200);

//Alternative way of defining frame for UIImageView
UIImageView *imageView = [[UIImageView alloc] init];
CGRect imageViewFrame = imageView.frame;
imageViewFrame.size.width = 200;
imageViewFrame.size.height = 200;
imageViewFrame.origin.x = 0;
imageViewFrame.origin.y = 0;
imageView.frame = imageViewFrame;
```

> Note: You must import UIKit to use a `UIImageView`.

# Section 11.6: Change color of an image

```
//Swift
imageView.tintColor = UIColor.redColor()
imageView.image = imageView.image?.imageWithRenderingMode(.AlwaysTemplate)

//Swift 3
imageView.tintColor = UIColor.red
imageView.image = imageView.image?.withRenderingMode(.alwaysTemplate)

//Objective-C
imageView.tintColor = [UIColor redColor];
imageView.image = [imageView.image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate]
```

# Section 11.7: Assigning an image to a UIImageView

You can assign an image to a `UIImageView` during initialization, or later using the `image` property:

```
//Swift
```

```
UIImageView(image: UIImage(named: "image1"))

UIImageView(image: UIImage(named: "image1"), highlightedImage: UIImage(named: "image2"))

imageView.image = UIImage(named: "image1")

//Objective-C
[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"];

[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"image1"] highlightedImage:[UIImage
imageNamed:@"image2"]];

imageView.image = [UIImage imageNamed:@"image1"];
```

# Chapter 12: Resizing UIImage

**CGInterpolationQuality**  **Levels of interpolation quality for rendering an image.**
Interpolation quality is a graphics state parameter typedef enum CGInterpolationQuality CGInterpolationQuality;

## Section 12.1: Resize any image by size & quality

```objc
- (UIImage *)drawImageBySize:(CGSize)size quality:(CGInterpolationQuality)quality
{
    UIGraphicsBeginImageContextWithOptions(size, NO, 0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetInterpolationQuality(context, quality);
    [self drawInRect: CGRectMake (0, 0, size.width, size.height)];
    UIImage *resizedImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return resizedImage;
}
```

# Chapter 13: Cut a UIImage into a circle

## Section 13.1: Cut a image into a circle - Objective C

import #include <math.h>

The code in the viewDidLoad or loadView should look something look something like this

```
- (void)loadView
{
[super loadView];
UIImageView *imageView=[[UIImageView alloc]initWithFrame:CGRectMake(0, 50, 320, 320)];
[self.view addSubview:imageView];
UIImage *image=[UIImage imageNamed:@"Dubai-Photos-Images-Travel-Tourist-Images-
Pictures-800x600.jpg"];
imageView.image=[self circularScaleAndCropImage:[UIImage imageNamed:@"Dubai-Photos-Images-Travel-
Tourist-Images-Pictures-800x600.jpg"] frame:CGRectMake(0, 0, 320, 320)];
}
```

Finally the function which does the heavy lifting `circularScaleAndCropImage` is as defined below

```
- (UIImage*)circularScaleAndCropImage:(UIImage*)image frame:(CGRect)frame {
    // This function returns a newImage, based on image, that has been:
    // - scaled to fit in (CGRect) rect
    // - and cropped within a circle of radius: rectWidth/2

    //Create the bitmap graphics context
    UIGraphicsBeginImageContextWithOptions(CGSizeMake(frame.size.width, frame.size.height), NO,
0.0);
    CGContextRef context = UIGraphicsGetCurrentContext();

    //Get the width and heights
    CGFloat imageWidth = image.size.width;
    CGFloat imageHeight = image.size.height;
    CGFloat rectWidth = frame.size.width;
    CGFloat rectHeight = frame.size.height;

    //Calculate the scale factor
    CGFloat scaleFactorX = rectWidth/imageWidth;
    CGFloat scaleFactorY = rectHeight/imageHeight;

    //Calculate the centre of the circle
    CGFloat imageCentreX = rectWidth/2;
    CGFloat imageCentreY = rectHeight/2;

    // Create and CLIP to a CIRCULAR Path
    // (This could be replaced with any closed path if you want a different shaped clip)
    CGFloat radius = rectWidth/2;
    CGContextBeginPath (context);
    CGContextAddArc (context, imageCentreX, imageCentreY, radius, 0, 2*M_PI, 0);
    CGContextClosePath (context);
    CGContextClip (context);

    //Set the SCALE factor for the graphics context
    //All future draw calls will be scaled by this factor
    CGContextScaleCTM (context, scaleFactorX, scaleFactorY);

    // Draw the IMAGE
    CGRect myRect = CGRectMake(0, 0, imageWidth, imageHeight);
```

```
    [image drawInRect:myRect];

    UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return newImage;
}
```

# Section 13.2: SWIFT 3 Example

```swift
override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        let imageView = UIImageView(frame: CGRect(x: CGFloat(0), y: CGFloat(50), width:
CGFloat(320), height: CGFloat(320)))
        view.addSubview(imageView)
        let image = UIImage(named: "Dubai-Photos-Images-Travel-Tourist-Images-
Pictures-800x600.jpg")
        imageView.image = circularScaleAndCropImage(UIImage(named: "Dubai-Photos-Images-Travel-
Tourist-Images-Pictures-800x600.jpg")!, frame: CGRect(x: CGFloat(0), y: CGFloat(0), width:
CGFloat(100), height: CGFloat(100)))
    }
```

Finally the function which does the heavy lifting circularScaleAndCropImage is as defined below

```swift
func circularScaleAndCropImage(_ image: UIImage, frame: CGRect) -> UIImage{
        // This function returns a newImage, based on image, that has been:
        // - scaled to fit in (CGRect) rect
        // - and cropped within a circle of radius: rectWidth/2
        //Create the bitmap graphics context
        UIGraphicsBeginImageContextWithOptions(CGSize(width: CGFloat(frame.size.width), height:
CGFloat(frame.size.height)), false, 0.0)
        let context: CGContext? = UIGraphicsGetCurrentContext()
        //Get the width and heights
        let imageWidth: CGFloat = image.size.width
        let imageHeight: CGFloat = image.size.height
        let rectWidth: CGFloat = frame.size.width
        let rectHeight: CGFloat = frame.size.height
        //Calculate the scale factor
        let scaleFactorX: CGFloat = rectWidth / imageWidth
        let scaleFactorY: CGFloat = rectHeight / imageHeight
        //Calculate the centre of the circle
        let imageCentreX: CGFloat = rectWidth / 2
        let imageCentreY: CGFloat = rectHeight / 2
        // Create and CLIP to a CIRCULAR Path
        // (This could be replaced with any closed path if you want a different shaped clip)
        let radius: CGFloat = rectWidth / 2
        context?.beginPath()
        context?.addArc(center: CGPoint(x: imageCentreX, y: imageCentreY), radius: radius,
startAngle: CGFloat(0), endAngle: CGFloat(2 * Float.pi), clockwise: false)
        context?.closePath()
        context?.clip()
        //Set the SCALE factor for the graphics context
        //All future draw calls will be scaled by this factor
        context?.scaleBy(x: scaleFactorX, y: scaleFactorY)
        // Draw the IMAGE
        let myRect = CGRect(x: CGFloat(0), y: CGFloat(0), width: imageWidth, height: imageHeight)
        image.draw(in: myRect)
        let newImage: UIImage? = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
```

```
        return newImage!
    }
```

# Chapter 14: UITableView

A simple, widely-used, yet very powerful view that can present data in a list form using rows and a single column. Users may scroll vertically through the items in a table view, and optionally manipulate and select content.

## Section 14.1: Self-Sizing Cells

In iOS 8 Apple introduced the self sizing cell. Layout your UITableViewCells with Autolayout explicitly and UITableView takes care of the rest for you. Row height is calculated automatically, by default `rowHeight` value is UITableViewAutomaticDimension.

UITableView property `estimatedRowHeight` is used when self-sizing cell is calculating.

> When you create a self-sizing table view cell, you need to set this property and use constraints to define the cell's size.
>
> -- *Apple, UITableView Documentation*

```
self.tableView.estimatedRowHeight = 44.0
```

Note that the tableView's delegate's `heightForRowAtIndexPath` is *unnecessary* if you want to have a dynamic height for all cells. Simply set the above property when necessary and before reloading or loading the table view. However, you can set specific cells' height while having others dynamic via the following function:

**Swift**

```swift
override func tableView(tableView: UITableView, heightForRowAtIndexPath indexPath: NSIndexPath) ->
CGFloat {
    switch indexPath.section {
    case 1:
        return 60
    default:
        return UITableViewAutomaticDimension
    }
}
```

**Objective-C**

```objc
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    switch (indexPath.section) {
        case 1:
            return 60;
        default:
            return UITableViewAutomaticDimension;
    }
}
```

## Section 14.2: Custom Cells

Customizing a `UITableViewCell` can allow for very powerful, dynamic, and responsive interfaces. With extensive customization and in combination with other techniques you can do things like: update specific properties or interface elements as they change, animate or draw things in the cell, efficiently load video as the user scrolls, or even display pictures as they download from a network. The possibilities here are nearly endless. Below is a simple

example of what a custom cell may look like.
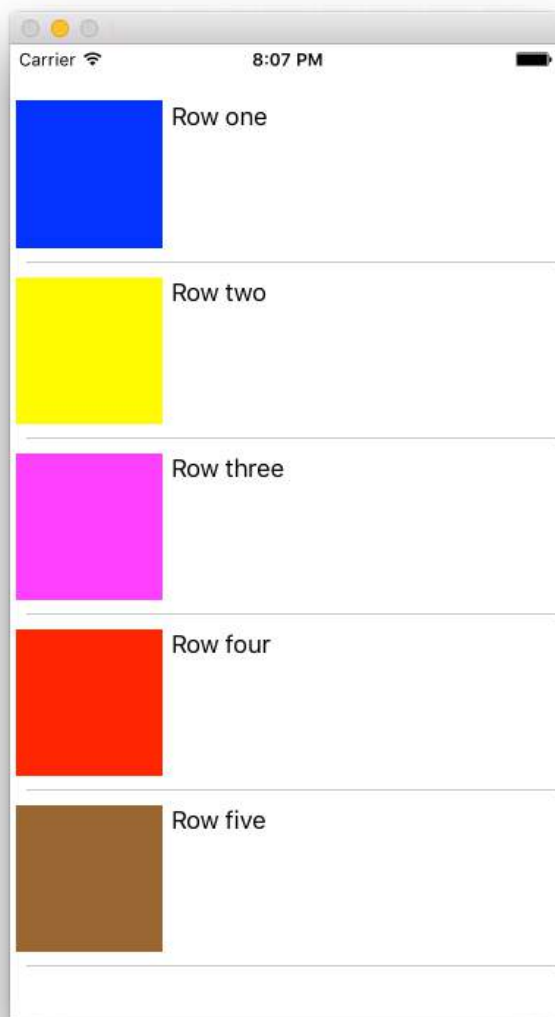


This section covers the basics, and hopefully will be expanded to detail more complex processes like those described above.

**Creating Your Custom Cell**

First, create a new subclass of `UITableViewCell` (create a new Cocoa Touch Class in Xcode and set `UITableViewCell` as the superclass). Below is what your code may look like after subclassing.

**Swift**

```swift
class CustomTableViewCell: UITableViewCell {
    static var identifier: String {
        return NSStringFromClass(self)
    }

    var customLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
        customLabel = UILabel(frame: CGRect(x: 0, y: 0, width: contentView.frame.width, height:
contentView.frame.height))
        customLabel.textAlignment = .center
```

```
            contentView.addSubview(customLabel)
        }
}
```

Optionally, check 'Also create a XIB file' when creating your new file to customize using Interface Builder. In the case that you do, connect `customLabel` as an `@IBOutlet`
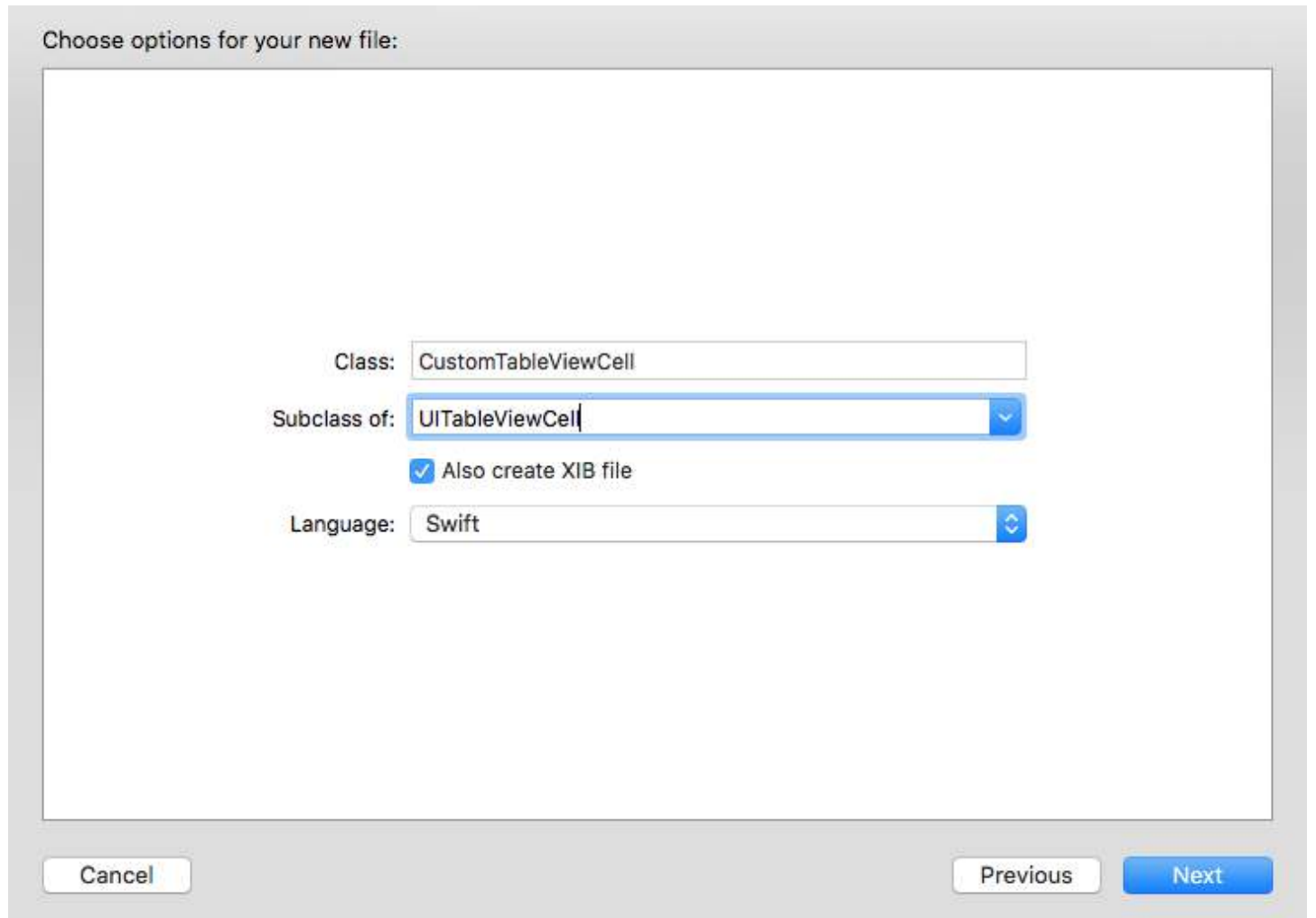


In a `UIViewController` containing the `tableView`, register the new custom cell's class (see below). *Note, this is only necessary if you do not design the cell with a Storyboard in your table view's interface.*

**Swift**

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Register Cell Class
    tableView.register(CustomTableViewCell.self, forCellReuseIdentifier:
CustomTableViewCell.identifier)
}
```

If you chose to use a XIB file, `registerNib` instead:

**Swift**

```
// Register Nib
tableView.register(UINib(nibName: CustomTableViewCell.identifier, bundle: nil),
forCellReuseIdentifier: CustomTableViewCell.identifier)
```

Now that your `tableView` knows about your custom cell, you can dequeue it in `cellForRowAtIndexPath`:

**Swift**

```swift
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
    // Load the CustomTableViewCell. Make sure the identifier supplied here matches the one from
your cell
    let cell: CustomTableViewCell =
tableView.dequeueReusableCellWithIdentifier(CustomTableViewCell.identifier) as! CustomTableViewCell

    // This is where the magic happens - setting a custom property on your very own cell
    cell.customLabel.text = "My Custom Cell"

    return cell
}
```

# Section 14.3: Separator Lines

**Editing the width of Separator Lines**

You can set make your table view's separator lines extend the to various widths across the table by changing the `layoutMargins:` property on your cell(s). This can be achieved in a number of ways.

**Changing the Separator Lines for specific cells**

In either your table view data source's `cellForRowAtIndexPath:` method *or* the `willDisplayCell:` method, set the cell's `layoutMargins:` property to `UIEdgeInsetsZero` (extends to full width of the table), or to whatever you may desire here.

**Objective-C**

```objc
[cell setLayoutMargins:UIEdgeInsetsZero];

// May also use separatorInset
[cell setSeparatorInset:UIEdgeInsetsZero];
```

**Swift**

```swift
func tableView(tableView: UITableView, willDisplayCell cell: UITableViewCell, forRowAtIndexPath
indexPath: NSIndexPath) {
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}

func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell
{
    cell.separatorInset = UIEdgeInsetsZero
    cell.layoutMargins = UIEdgeInsetsZero
}
```

**Remove all Separator Lines**

The thin gray lines between each cell may not be exactly the look you're going for. It's fairly straightforward to hide them from view.

In your encompassing `UIViewController`'s `viewDidLoad:` method add the following code. You may also set this

property at any time before loading or reloading the table view (does not necessarily need to be in the
`viewDidLoad:` method).

**Swift:**

```
tableView.separatorStyle = .None
```

**Objective-C:**

```
tableView.separatorStyle = UITableViewCellSeparatorStyleNone;
```

Alternatively, the property can be changed in your Storyboard or XIB by selecting your tableView and setting
`separator` (under the attributes inspector) to `None`.

**Hide excess Separator Lines**

You can hide the `UITableViewCell` separator lines for empty cells by setting an empty footer view at the bottom of
a UITableView:

**Swift**

```
tableView.tableFooterView = UIView()
```

**Objective-C**

```
tableView.tableFooterView = [[UIView alloc] initWithFrame:CGRectZero];
```

| Angry Birds | | Angry Birds | |
| Chess | | Chess | |
| Russian Roulette | | Russian Roulette | |
| Spin the Bottle | | Spin the Bottle | |
| Texas Hold'em Poker | ✓ | Texas Hold'em Poker | ✓ |
| Tic-Tac-Toe | | Tic-Tac-Toe | |
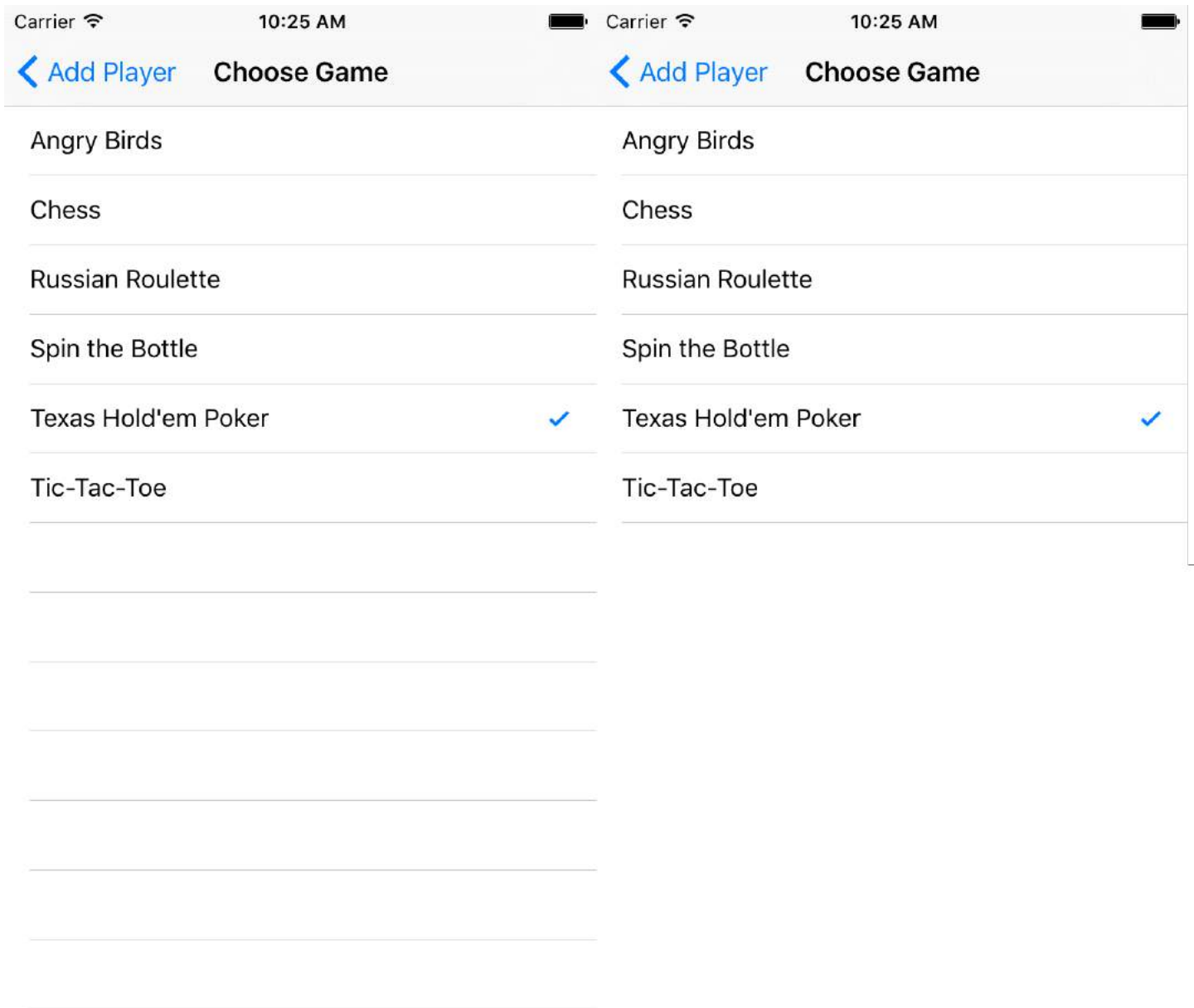
Image is from [Ray Wenderlich](#).

# Section 14.4: Delegate and Datasource

The `UITableViewDelegate` is used to control how the table is displayed, and `UITableViewDataSource` is used to define the `UITableView`'s data. There are two required methods and many optional ones which can be used to customize size, sections, headings, and cells in the `UITableView`.

**UITableViewDataSource**
**Required Methods**

> `numberOfRowsInSection`: This method defines how many cells will be displayed in each section of the tableview.

**Objective-C**

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    // Return the number of rows for the table view. Usually populated from an array,
```

```
    // or can be statically defined.
    return self.myArray.count;
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // Return the number of rows for the table view. Usually populated from an array,
    // or can be statically defined.
    return self.myArray.count
}
```

> cellForRowAtIndexPath: This method is where the UITableView's cells are created and configured.
> Should return either a UITableViewCell or a custom subclass.

**Note:** Using dequeueReusableCellWithIdentifier:forIndexPath: requires that the class or nib has been registered for that identifier using the UITableView's registerClass:forCellReuseIdentifier: or registerNib:forCellReuseIdentifier: methods. Usually, this will be done in the UIViewController's viewDidLoad method.

**Objective-C**

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    MyCustomCell *cell = [tableView dequeueReusableCellWithIdentifier:@"MyCustomCell"
                                                        forIndexPath:indexPath];

    // All additional customization goes here
    cell.titleLabel.text = [NSString stringWithFormat:@"Title Row %lu", indexPath.row];

    return cell;
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("MyCustomCell", forIndexPath:indexPath)

    // All additional customization goes here
    cell.titleLabel.text = String(format:"Title Row %lu", indexPath.row)

    return cell
}
```

**Optional Methods**

> titleForHeaderInSection: Defines a string as the title for each section header in the table view. This
> method only allows for changing the title, further customization can be done by defining the view for the
> header.

**Objective-C**

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    switch(section) {
```

```
        case 0:
            return @"Title 1";
            break;

        case 1:
            return @"Title 2";
            break;

        default:
            return nil;
            break;
    }
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    switch section {
        case 0:
            return "Title 1"
        case 1:
            return "Title 2"
        default:
            return nil
    }
}
```

> `titleForFooterInSection`: Defines a string as the title for each section header in the table view.

**Objective-C**

```
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section {
    return @"Footer text";
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, titleForFooterInSection section: Int) -> String? {
    return "Footer text"
}
```

> `canEditRowAtIndexPath`: Used to determine if the editing UI should be displayed for the specified row.
> Should return `YES` if the specified row can be deleted or added.

**Objective-C**

```
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return YES;
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool {
    return true
```

```
}
```

> commitEditingStyle:forRowAtIndexPath Should perform the work required to handle the addition or
> removal of the specified row. For example, remove the cell from the UITableView with animation, and
> remove the associated object from the table's data model.

**Objective-C**

```objc
- (void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath
*)indexPath {
    switch (editingStyle) {
        case UITableViewCellEditingStyleInsert:
            // Insert new data into the backing data model here
            [self insertNewDataIntoDataModel];
            [tableView insertRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        case UITableViewCellEditingStyleDelete:
            [self removeDataFromDataModelAtIndex:indexPath.row];
            [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationAutomatic];
            break;
        default:
            // Nothing to perform if the editingStyle was neither Insert or Delete
            break;

    }
}
```

**Swift 3**

```swift
func tableView(_ tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {
    switch editingStyle {
        case .Insert:
            self.insertNewDataIntoDataModel()
            tableView.insertRowsAtIndexPaths([indexPath], withRowAnimation:.Automatic)
        case .Delete:
            self.removeDataFromDataModelAtIndex(indexPath.row)
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation:.Automatic)
        default:
            // Nothing to perform if the editingStyle was neither Insert or Delete
    }
}
```

> editActions:forRowAt Allows ability to add aditional actions or buttons to the edit mode of a row inside
> a UITableview. For example if you wanted two buttons, an edit and delete button when user swipes to
> edit the row, then you would use this method.

**Swift 3**

```swift
override func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) ->
[UITableViewRowAction]? {
    // In the handler you will get passed the action as well as the indexPath for
```

```
    // the row that is being edited
    let editAction = UITableViewRowAction(style: .normal, title: "Edit", handler: { [unowned self]
action, indexPath in
        // Do something when edit is tapped
    })

    // Change the color of the edit action
    editAction.backgroundColor = UIColor.blue

    let deleteAction = UITableViewRowAction(style: .destructive, title: "Delete", handler: {
[unowned self] action, indexPath in
        // Handel the delete event
    })


    return [deleteAction, editAction]
}
```

**UITableViewDelegate**

All methods in UITableViewDelegate are optional, but a delegate that implements them will enable extra features for the UITableView.

> numberOfSectionsInTableView: By default this returns 1, but multiple section support is enabled by returning a different number of sections.

**Objective-C**

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return self.numSections;
}
```

**Swift 3**

```
func numberOfSectionsInTableView(_ tableView: UITableView) -> Int {
    return self.numSections
}
```

> viewForHeaderInSection Allows the configuration of a custom view as the header for the section.

**Objective-C**

```
- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section {

    UIView *view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, CGRectGetWidth(tableView.frame),
22)];
    view.backgroundColor = [UIColor groupTableViewBackgroundColor];

    UILabel *label = [[UILabel alloc] init];
    label.font = [UIFont systemFontOfSize:12];
    label.textColor = [UIColor darkGrayColor];

    switch (section) {
        case 1: {
            label.text = @"Title";
            label.frame = labelFrame;
```

```
            UIButton *more = [[UIButton alloc] initWithFrame:btnFrame];
            [more setTitle:@"See more" forState:UIControlStateNormal];
            [more.titleLabel setFont:[UIFont systemFontOfSize:12]];
            [view addSubview:more];
        }   break;

        default:
            label.frame = CGRectMake(0, 0, 0, 0);
            break;
    }

    [view addSubview:label];
    return view;
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView,  viewForHeaderInSection section: Int) -> UIView? {
    let view = UIView(frame: CGRect(x: 0, y: 0, width: tableView.frame.size.width, height: 22))
    view.backgroundColor = UIColor.groupTableViewBackgroundColor()

    let label = UILabel()
    label.font = UIFont.systemFontOfSize(12)
    label.textColor = UIColor.darkGrayColor()

    switch section {
        case 1:
            label.text = "Title"
            label.frame = labelFrame

            let more = UIButton(frame: btnFrame)
            more.setTitle("See more", forState:.Normal)
            view.addSubview(more)

        default:
            label.frame = CGRect.zero
    }

    view.addSubview(label)
    return view;
}
```

> heightForRowAtIndexPath: Define the height of each cell in the table view.

**Objective-C**

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    return 44;
}
```

**Swift 3**

```
func tableView(_ tableView: UITableView, heightForRowAtIndexPath indexPath: NSIndexPath) -> CGFloat
{
    return 44
}
```

> heightForHeaderInSection: and heightForFooterInSection Define the height for the header and footer of each section in the table view

**Objective-C**

```objc
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {
    return 33;
}
```

**Swift 3**

```swift
func tableView(_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {
    return 33
}
```

# Section 14.5: Creating a UITableView

A Table View is a list of rows that can be selected. Each row is populated from a data source. This example creates a simple table view in which each row is a single line of text.
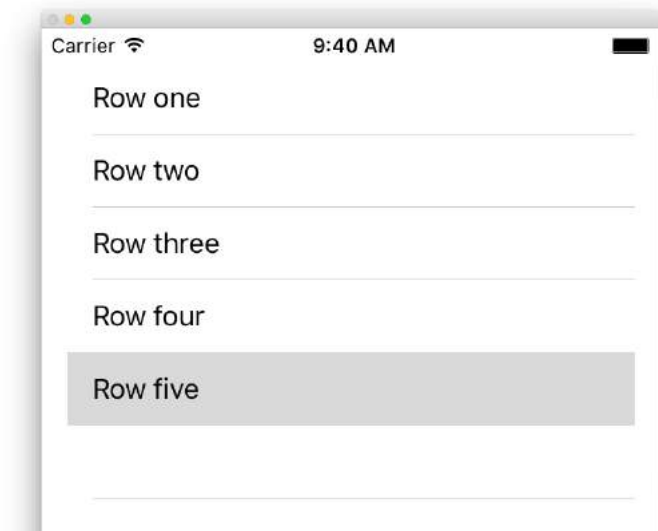


**Add a UITableView to your Storyboard**

Although there are a number of ways to create a `UITableView`, one of the easiest is to add one to a Storyboard. Open your Storyboard and drag a `UITableView` onto your `UIViewController`. Make sure to use Auto Layout to correctly align the table (pin all four sides).

**Populating Your Table with Data**

In order to display content dynamically (i.e. load it from a data source like an array, a Core Data model, a networked server, etc.) in your table view you need to setup the data source.

**Creating a simple data source**

A data source could, as stated above, be anything with data. Its entirely up to you how to format it and whats in it. The only requirement is that you must be able to read it later so that you can populate each row of your table with data when needed.

In this example, we'll just set an array with some strings (text) as our data source:

**Swift**

```swift
let myDataArray: [String] = ["Row one", "Row two", "Row three", "Row four", "Row five"]
```

**Objective-C**

```objc
// You'll need to define this variable as a global variable (like an @property) so that you can
access it later when needed.
NSArray *myDataArray = @[@"Row one", @"Row two", @"Row three", @"Row four", @"Row five"];
```

**Setting up your data source in your View Controller**

Make sure your view controller conforms to the UITableViewDataSource protocol.

**Swift**

```swift
class ViewController: UIViewController, UITableViewDataSource {
```

**Objective-C**

```objc
@interface ViewController : UIViewController <UITableViewDataSource>
```

As soon as your view controller has declared it will **conform** to the UITableViewDataSource (that's what we've just done above), you are *required* to implement at least the following methods in your view controller class:

- tableView:numberOfRowsInSection, this asks you how many rows your table view should have.

  ```swift
  // Swift

  func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
      return self.myDataArray.count
  }
  ```

- tableView:cellForRowAtIndexPath, requests that you create and return a cell for each row you specified in tableView:numberOfRowsInSection. So, if you said you needed 10 rows, this method will be called ten times for each row, and you need to create a cell for each of those rows.

  ```swift
  // Swift

  func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
  UITableViewCell {
      // Create a new cell here. The cellReuseIdentifier needs to match the reuse identifier
  from the cell in your Storyboard
      let cell: UITableViewCell =
  tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!

      // Set the label on your cell to the text from your data array
      cell.textLabel?.text = self.myDataArray[indexPath.row]

      return cell
  }
  ```

> **WARNING**: You may **NOT** return nil for any cells in cellForRowAtIndexPath:. This will cause your app to crash, and you will see the following error in the console:

```
Uncaught exception 'NSInternalInconsistencyException', reason: 'UITableView dataSource
must return a cell from tableView:cellForRowAtIndexPath:'
```

**Connecting the table view's data source to your view controller**

You can either do this via code by setting your table's `dataSource` property to `self` on your view controller. Or you may select your table view in your storyboard, open the Attributes Inspector, select the "Outlets" panel, and drag from `dataSource` to your view controller (**NOTE**: make sure you connect to the UIViewCONTROLLER, **not** a UIView or another object *in* your UIViewController).

**Handling row selections**

When a user taps on a row in your table view, generally, you'll want to do something - to respond. In many apps, when you tap on a row, more information about that item you tapped upon is displayed. Think of the Messages app: when you tap on the row showing one of your contacts, the conversation with that person is then displayed on screen.

In orer to do that, you must conform to the `UITableViewDelegate` protocol. Doing so is similar to conforming to the data source protocol. This time however, you'll just add it next to `UITableViewDataSource` and separate it with a comma. So it should look like this:

**Swift**

```
class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {
```

**Objective-C**

```
@interface ViewController : UIViewController <UITableViewDataSource, UITableViewDelegate>
```

There are no required methods to implement for the table view's delegate. However, to handle row selections you'll need to use the following method:

- `tableView:didSelectRowAtIndexPath`, this is called whenever a row is tapped, which allows you to do something in response. For our example, we'll just print a confirmation statement to the Xcode log.

```
// Swift

func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    print("You tapped cell number \(indexPath.row).")
 }

// Objective-C

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NSLog(@"You tapped cell number %ld.", (long)indexPath.row);
}
```

**The Final Solution**

See below for the full setup with just code, no explanation.

**Swift**

```
import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
```

```swift
    // Data model: These strings will be the data for the table view cells
    let myDataArray: [String] = ["Row one", "Row two", "Row three", "Row four", "Row five"]

    // cell reuse id (cells that scroll out of view can be reused)
    let cellReuseIdentifier = "cell"

    // don't forget to hook this up from the storyboard
    @IBOutlet var myTableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Register the table view cell class and its reuse id
        myTableView.registerClass(UITableViewCell.self, forCellReuseIdentifier:
cellReuseIdentifier)

        // This view controller itself will provide the delegate methods and row data for the table
view.
        myTableView.delegate = self
        myTableView.dataSource = self
    }

    // number of rows in table view
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.myDataArray.count
    }

    // create a cell for each table view row
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {

        // create a new cell if needed or reuse an old one
        let cell:UITableViewCell = tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier)
as UITableViewCell!

        // set the text from the data model
        cell.textLabel?.text = self.myDataArray[indexPath.row]

        return cell
    }

    // method to run when table view cell is tapped
    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        print("You tapped cell number \(indexPath.row).")
    }
}
```

**Objective-C**

**ViewController.h**

```objc
#import <UIKit/UIKit.h>

@interface ViewController: UIViewController <UITableViewDelegate, UITableViewDataSource> {
    IBOutlet UITableView *myTableView;
    NSArray *myDataArray;
}

@end
```

**ViewController.m**

```
#import "ViewController.h"

// cell reuse id (cells that scroll out of view can be reused)
NSString * _Nonnull cellReuseIdentifier = @"cell";

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Data model: These strings will be the data for the table view cells
    myDataArray = @[@"Row one", @"Row two", @"Row three", @"Row four", @"Row five"];

    // Register the table view cell class and its reuse id
    [myTableView registerClass:[UITableViewCell class] forCellReuseIdentifier:cellReuseIdentifier];

    // This view controller itself will provide the delegate methods and row data for the table
view.
    myTableView.delegate = self;
    myTableView.dataSource = self;
}

// number of rows in table view
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return myDataArray.count;
}

// create a cell for each table view row
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath {
    // create a new cell if needed or reuse an old one
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellReuseIdentifier];

    // set the text from the data model
    cell.textLabel.text = myDataArray[indexPath.row];

    return cell;
}

// method to run when table view cell is tapped
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    NSLog(@"You tapped cell number %ld.", (long)indexPath.row);
}

@end
```

## Section 14.6: Swipe to Delete Rows

I always think it is nice to have a very simple, self-contained example so that nothing is assumed when I am learning a new task. This answer is that for deleting UITableView rows. The project performs like this:
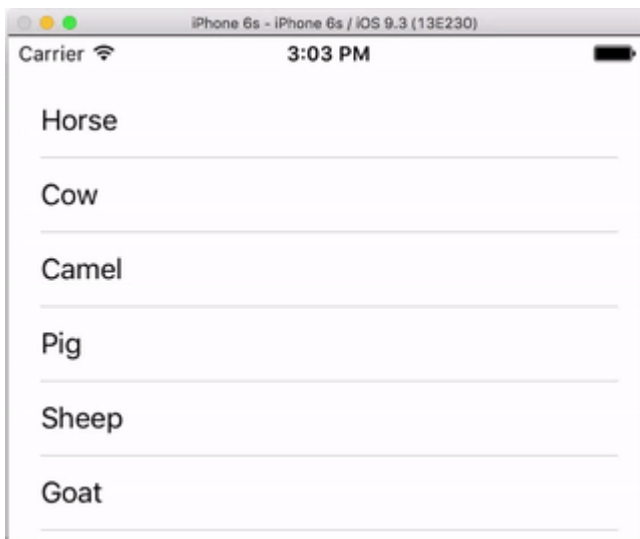
This project is based on the UITableView example for Swift.

**Add the Code**

Create a new project and replace the ViewController.swift code with the following.

```swift
import UIKit
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    // These strings will be the data for the table view cells
    var animals: [String] = ["Horse", "Cow", "Camel", "Pig", "Sheep", "Goat"]

    let cellReuseIdentifier = "cell"

    @IBOutlet var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // It is possible to do the following three things in the Interface Builder
        // rather than in code if you prefer.
        self.tableView.registerClass(UITableViewCell.self, forCellReuseIdentifier:
cellReuseIdentifier)
        tableView.delegate = self
        tableView.dataSource = self
    }

    // number of rows in table view
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.animals.count
    }

    // create a cell for each table view row
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {

        let cell:UITableViewCell =
self.tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier) as UITableViewCell!

        cell.textLabel?.text = self.animals[indexPath.row]

        return cell
    }
```

```swift
    // method to run when table view cell is tapped
    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
        print("You tapped cell number \(indexPath.row).")
    }

    // this method handles row deletion
    func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

        if editingStyle == .Delete {

            // remove the item from the data model
            animals.removeAtIndex(indexPath.row)

            // delete the table view row
            tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)

        } else if editingStyle == .Insert {
            // Not used in our example, but if you were adding a new row, this is where you would
do it.
        }
    }

}
```

The single key method in the code above that enables row deletion is the last one. Here it is again for emphasis:

```swift
func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // remove the item from the data model
        animals.removeAtIndex(indexPath.row)

        // delete the table view row
        tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Fade)
    }
}
```

**Storyboard**

Add a `UITableView` to the View Controller in the storyboard. Use auto layout to pin the four sides of the table view to the edges of the View Controller. Control drag from the table view in the storyboard to the `@IBOutlet var tableView: UITableView!` line in the code.

**Finished**

That's all. You should be able to run your app now and delete rows by swiping left and tapping "Delete".

**Notes**

- This is only available from iOS 8. See this answer for more details.
- If you need to change the number of buttons displayed or the button text then see this answer for more details.

**Further reading**

- How To Make A Swipeable Table View Cell With Actions – Without Going Nuts With Scroll Views
- Apple Documentation

# Section 14.7: Expanding & Collapsing UITableViewCells

In your Storyboard, add a UITableView object on your UIViewController and let it cover the entire view. Setup the UITableviewDataSource and UITableviewDelegate connections.

**Objective-C**

In your .h file

```
NSMutableArray *arrayForBool;
NSMutableArray *sectionTitleArray;
```

In your .m file

```objc
- (void)viewDidLoad   {
    [super viewDidLoad];

    arrayForBool = [[NSMutableArray alloc] init];
    sectionTitleArray = @[@"Sam",@"Sanju",@"John",@"Staffy"];

    for (int i=0; i<[sectionTitleArray count]; i++) {
        [arrayForBool addObject:[NSNumber numberWithBool:NO]];
    }

    _tableView.dataSource = self;
    _tableView.delegate = self;
}

 // Declare number of rows in section
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
     if ([[arrayForBool objectAtIndex:section] boolValue]) {
         return section+2;
     } else {
         return 0;
     }
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath {

static NSString *cellid=@"hello";
UITableViewCell *cell=[tableView dequeueReusableCellWithIdentifier:cellid];
if (cell==nil) {
    cell=[[UITableViewCell alloc]initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:cellid];
}
   BOOL manyCells  = [[arrayForBool objectAtIndex:indexPath.section] boolValue];

        /** If the section supposed to be closed******/
    if(!manyCells){
        cell.backgroundColor=[UIColor clearColor];
        cell.textLabel.text=@"";
    }
     /** If the section supposed to be Opened******/
    else{
        cell.textLabel.text=[NSString stringWithFormat:@"%@ %d",[sectionTitleArray
objectAtIndex:indexPath.section],indexPath.row+1];
        cell.backgroundColor=[UIColor whiteColor];
        cell.selectionStyle=UITableViewCellSelectionStyleNone ;
    }
```

```objc
cell.textLabel.textColor=[UIColor blackColor];

 /** Add a custom Separator with cell*/
  UIView* separatorLineView = [[UIView alloc]initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[cell.contentView addSubview:separatorLineView];
return cell;
}


- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
return [sectionTitleArray count];
}



- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath  *)indexPath
{

/*************** Close the section, once the data is selected **********************************/
[arrayForBool replaceObjectAtIndex:indexPath.section withObject:[NSNumber numberWithBool:NO]];

 [_expandableTableView reloadSections:[NSIndexSet indexSetWithIndex:indexPath.section]
withRowAnimation:UITableViewRowAnimationAutomatic];

 }

 - (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
if ([[arrayForBool objectAtIndex:indexPath.section] boolValue]) {
    return 40;
}
return 0;

   }

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section
{

UIView *sectionView=[[UIView alloc]initWithFrame:CGRectMake(0, 0, 280,40)];
sectionView.tag=section;
UILabel *viewLabel=[[UILabel alloc]initWithFrame:CGRectMake(10, 0,
_expandableTableView.frame.size.width-10, 40)];
viewLabel.backgroundColor=[UIColor clearColor];
viewLabel.textColor=[UIColor blackColor];
viewLabel.font=[UIFont systemFontOfSize:15];
viewLabel.text=[NSString stringWithFormat:@"List of %@",[sectionTitleArray objectAtIndex:section]];
[sectionView addSubview:viewLabel];
        /********** Add a custom Separator with Section view *******************/
UIView* separatorLineView = [[UIView alloc] initWithFrame:CGRectMake(15, 40,
_expandableTableView.frame.size.width-15, 1)];
separatorLineView.backgroundColor = [UIColor blackColor];
[sectionView addSubview:separatorLineView];

/********** Add UITapGestureRecognizer to SectionView   **************/

UITapGestureRecognizer  *headerTapped  = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(sectionHeaderTapped:)];
[sectionView addGestureRecognizer:headerTapped];

return  sectionView;
```

```
}

- (void)sectionHeaderTapped:(UITapGestureRecognizer *)gestureRecognizer{

NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:gestureRecognizer.view.tag];
if (indexPath.row == 0) {
    BOOL collapsed  = [[arrayForBool objectAtIndex:indexPath.section] boolValue];
    for (int i=0; i<[sectionTitleArray count]; i++) {
        if (indexPath.section==i) {
            [arrayForBool replaceObjectAtIndex:i withObject:[NSNumber numberWithBool:!collapsed]];
        }
    }
    [_expandableTableView reloadSections:[NSIndexSet indexSetWithIndex:gestureRecognizer.view.tag]
withRowAnimation:UITableViewRowAnimationAutomatic];

    }
 }
```

# Chapter 15: UITableViewController

UITableViewController controller object that manages a table view. For some certain scenario it will be recommended to use UITableViewController, for example if you have lot of cells and some have UITextfield.

## Section 15.1: TableView with dynamic properties with tableviewCellStyle basic

```swift
override func numberOfSections(in tableView: UITableView) -> Int {
    // You need to return minimum one to show the cell inside the tableview
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // return the number of rows inside the tableview.
    return 3
}


override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
// identifier string should be same as what you have entered in the cell Attribute inspector ->
identifier (see the image).

    // Configure the cell...
    cell.textLabel?.text = "Cell \(indexPath.row) :" + "Hello"
//cell have different style Custom, basic, right detail, left detail, subtitle.
//For custom you can use your own objects and constrains, for other styles all
//is ready just select according to your design. (see the image for changing the style)

    return cell
}

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        // this delegate method will trigger when you click a cell
}
```
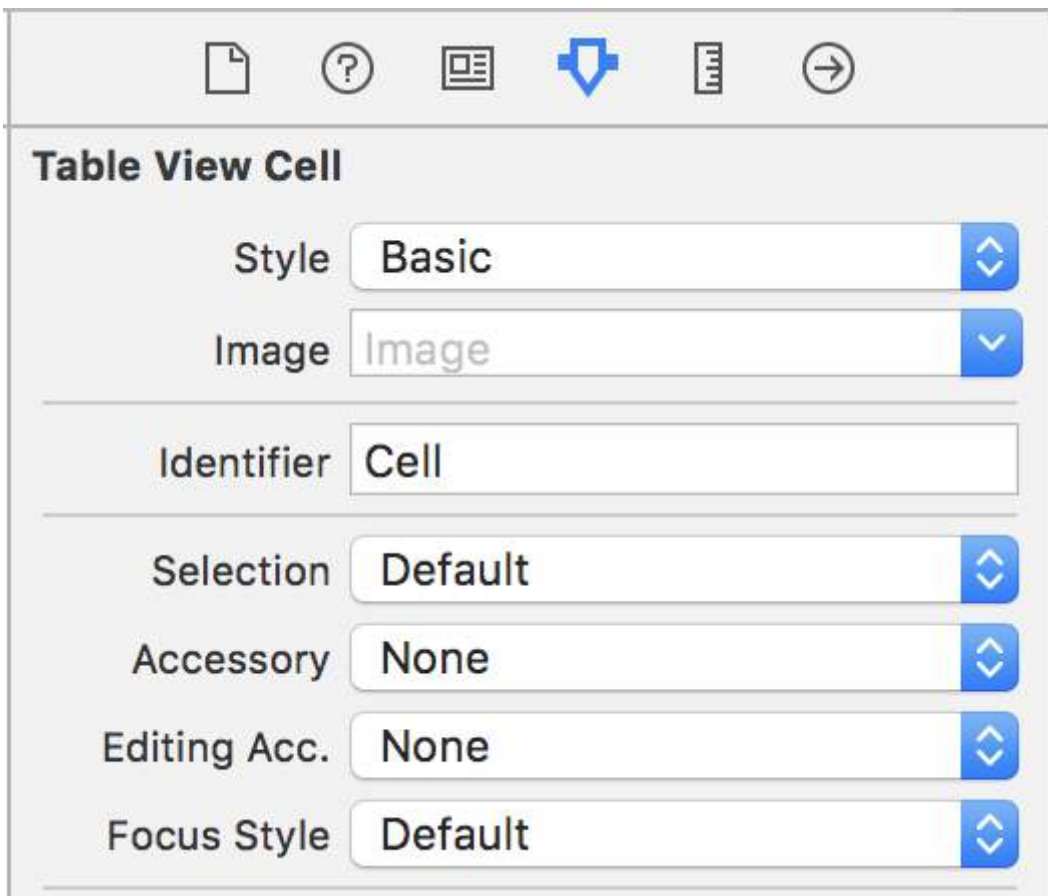
## Section 15.2: TableView with Custom Cell

For custom tableview cell you need a class that is subclass from `UITableViewCell`, an example class you can see below.

```
class TableViewCell: UITableViewCell {

@IBOutlet weak var lblTitle: UILabel!

override func awakeFromNib() {
    super.awakeFromNib()
    // Initialization code
}

override func setSelected(_ selected: Bool, animated: Bool) {
    super.setSelected(selected, animated: animated)

    // Configure the view for the selected state
}

}
```

Your tableview delegates

```
override func numberOfSections(in tableView: UITableView) -> Int {
        // You need to return minimum one to show the cell inside the tableview
        return 1
    }

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // return the number of rows inside the tableview.
    return 3
```

```
}

override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {

    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as!
TableViewCell
    // identifier string should be same as what you have entered in the cell Attribute inspector ->
identifier.

    // Configure the cell...
    cell.lblTitle.text = "Cell \(indexPath.row) :" + "Hello"


    return cell
}

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    // this delegate method will trigger when you click a cell
}
```

# Chapter 16: UIRefreshControl TableView

A UIRefreshControl object provides a standard control that can be used to initiate the refreshing of a table view's contents. You link a refresh control to a table through an associated table view controller object. The table view controller handles the work of adding the control to the table's visual appearance and managing the display of that control in response to appropriate user gestures.

## Section 16.1: Set up refreshControl on tableView:

```
UIRefreshControl *refreshControl = [[UIRefreshControl alloc] init];
[refreshControl addTarget:self action:@selector(pullToRefresh:)
forControlEvents:UIControlEventValueChanged];
self.scrollView.alwaysBounceVertical = YES;
[self.scrollView addSubview:refreshControl];

- (void)pullToRefresh:(UIRefreshControl*) sender{
//Do work off the main thread
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    // Simulate network traffic (sleep for 2 seconds)
    [NSThread sleepForTimeInterval:2];
    //Update data
    //Call complete on the main thread
    dispatch_sync(dispatch_get_main_queue(), ^{
        //Update network activity UI
        NSLog(@"COMPLETE");
        [sender endRefreshing];
    });
});

}
```

## Section 16.2: Objective-C Example

First declare a property like this in the ViewController

```
@property (nonatomic) UIRefreshControl *refreshControl;
```

Later in the `viewDidLoad()` set up the refreshControl as given below:

```
self.refreshControl = [[UIRefreshControl alloc]init];
[self.tableView addSubview:self.refreshControl];
[self.refreshControl addTarget:self action:@selector(refreshTable)
forControlEvents:UIControlEventValueChanged];
//Setting the tint Color of the Activity Animation
self.refreshControl.tintColor = [UIColor redColor];
//Setting the attributed String to the text
NSMutableAttributedString * string = [[NSMutableAttributedString alloc]
initWithString:@"firstsecondthird"];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor redColor]
range:NSMakeRange(0,5)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor greenColor]
range:NSMakeRange(5,6)];
[string addAttribute:NSForegroundColorAttributeName value:[UIColor blueColor]
range:NSMakeRange(11,5)];
self.refreshControl.attributedTitle = string;
```

Now The function `refreshTable` is defined as:

```
- (void)refreshTable {
    //TODO: refresh your data
    [self.refreshControl endRefreshing];
    [self.refreshControl beginRefreshing];
    [self.tableView reloadData];
    [self.refreshControl endRefreshing];
}
```

firstsecondthird

# Chapter 17: UITableViewCell

Load custom cell xib file uses the cell category class, no need to register the nib file

## Section 17.1: Xib file of UITableViewCell

Create a `UITableView` cell category class.

**UITableViewCell+RRCell.h file**

```objc
#import <UIKit/UIKit.h>

@interface UITableViewCell (RRCell)

-(id)initWithOwner:(id)owner;

@end
```

**UITableViewCell+RRCell.m file**

```objc
#import "UITableViewCell+RRCell.h"

@implementation UITableViewCell (RRCell)

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wobjc-designated-initializers"

-(id)initWithOwner:(id)owner {

    if (self = [super init]) {

        NSArray *nib = [[NSBundle mainBundle]loadNibNamed:NSStringFromClass([self class])
owner:self options:nil];
        self = [nib objectAtIndex:0];
    }
    return self;
}

#pragma clang diagnostic pop


@end
```

Import cell category class to use this method into the `cellForRowAtIndexPath` method

```objc
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
    //Creted custom cell xib file to load by cell category class
    CustomCell *cell = [[CustomCell alloc]initWithOwner:self];

    return cell;
}
```

# Chapter 18: Custom methods of selection of UITableViewCells

## Section 18.1: Distinction between single and double selection on row

An example of implementation UITableView which allows to detect if cell has been tapped single or double time.

```swift
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

# Chapter 19: Custom methods of selection of UITableViewCells

Advance ways to manage selections of UITableViewCell. Examples when simple `didSelect...` form `UITableViewDelegate` is not enough to achieve something.

## Section 19.1: Distinction between single and double selection on row

An example of implementation which give a possibility to detect if user single or double tap on UITableViewCell.

```swift
override func viewDidLoad() {
    viewDidLoad()

    let doubleTapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleDoubleTap(sender:)))
    doubleTapGestureRecognizer.numberOfTapsRequired = 2
    tableView.addGestureRecognizer(doubleTapGestureRecognizer)

    let tapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(handleTapGesture(sender:)))
    tapGestureRecognizer.numberOfTapsRequired = 1
    tapGestureRecognizer.require(toFail: doubleTapGestureRecognizer)
    tableView.addGestureRecognizer(tapGestureRecognizer)
}

func handleTapGesture(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}

func handleDoubleTap(sender: UITapGestureRecognizer) {
    let touchPoint = sender.location(in: tableView)
    if let indexPath = tableView.indexPathForRow(at: touchPoint) {
        print(indexPath)
    }
}
```

# Chapter 20: UIView

## Section 20.1: Make the view rounded

To make a rounded `UIView`, specify a `cornerRadius` for the view's `layer`.

*This also applies any class which inherits from UIView, such as UIImageView.*

**Programmatically**

**Swift Code**

```swift
someImageView.layoutIfNeeded()
someImageView.clipsToBounds = true
someImageView.layer.cornerRadius = 10
```

**Objective-C Code**

```objc
[someImageView layoutIfNeeded];
someImageView.clipsToBounds = YES;
someImageView.layer.cornerRadius = 10;
```

**Example**

```swift
//Swift code
topImageView.layoutIfNeeded()
bottomImageView.layoutIfNeeded()
topImageView.clipsToBounds = true
topImageView.layer.cornerRadius = 10
bottomImageView.clipsToBounds = true
bottomImageView.layer.cornerRadius = bottomImageView.frame.width / 2

//Objective-C code
[topImageView layoutIfNeeded]
[bottomImageView layoutIfNeeded];
topImageView.clipsToBounds = YES;
topImageView.layer.cornerRadius = 10;
bottomImageView.clipsToBounds = YES;
bottomImageView.cornerRadius = CGRectGetWidth(bottomImageView.frame) / 2;
```

Here is the result, showing the rounded view effect using the specified corner radius:

**Note**

To do this you need to include the QuartzCore framework.

```
#import <QuartzCore/QuartzCore.h>
```

**Storyboard Configuration**

A rounded view effect can also be achieved `non-programmatically` by setting the corresponding properties in **Storyboard**.
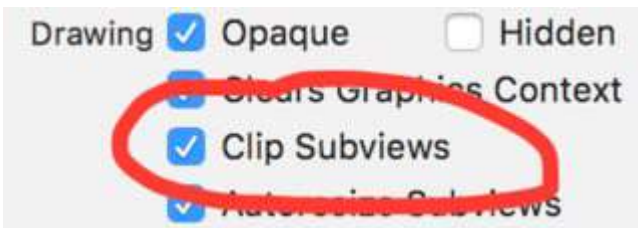


*Since `layer` properties aren't exposed in Storyboard, you have to modify the `cornerRadius` attribute via the User Defined Runtime Attributes section.*

**Swift Extension**

You can use this handy extension to apply rounded view as long as it has same width and height.

```
extension UIView {
    @discardableResult
    public func setAsCircle() -> Self {
        self.clipsToBounds = true
        let frameSize = self.frame.size
        self.layer.cornerRadius = min(frameSize.width, frameSize.height) / 2.0
        return self
    }
}
```

To use it:

```
yourView.setAsCircle()
```

# Section 20.2: Using IBInspectable and IBDesignable

One (or two) of the coolest new features in recent Xcode releases are the `IBInspectable` properties and `IBDesignable` `UIView`s. These have nothing to do with the functionality of your application but instead impact the developer experience in Xcode. The goal is to be able to visually inspect custom views in your iOS application without running it. So assume that you have a custom view creatively named `CustomView` that inherits from `UIView`. In this custom view, it will display a string of text with a designated color. You can also choose not to display any text. We'll need three properties:

```
var textColor: UIColor = UIColor.blackColor()
var text: String?
var showText: Bool = true
```

We can then override the `drawRect` function in the class:

```
if showText {
    if let text = text {
        let s = NSString(string: text)
        s.drawInRect(rect,
            withAttributes: [
                NSForegroundColorAttributeName: textColor,
                NSFontAttributeName: UIFont(name: "Helvetica Neue", size: 18)!
            ])
    }
}
```

Assuming that the `text` property is set, this will draw a string in the upper left hand corner of the view when the application is run. The problem is we won't know what it looks like without running the application. This is where `IBInspectable` and `IBDesignable` come in. `IBInspectable` allows us to visually set property values of the view in Xcode, just like with the built in controls. `IBDesignable` will show us a visual preview in the storyboard. Here is how the class should look:

```
@IBDesignable
class CustomView: UIView {
    @IBInspectable var textColor: UIColor = UIColor.blackColor()
    @IBInspectable var text: String?
    @IBInspectable var showText: Bool = true
```

```
    override func drawRect(rect: CGRect) {
        // ...
    }
}
```

Or in Objective C:

```objc
IB_DESIGNABLE
@interface CustomView: UIView

@property (nonatomic, strong) IBInspectable UIColor* textColor;
@property (nonatomic, strong) IBInspectable NSString* text;
@property (nonatomic, assign) IBInspectable BOOL showText;

@end

@implementation CustomView

- (instancetype)init {
    if(self = [super init]) {
        self.textColor = [UIColor blackColor];
        self.showText = YES;
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
    //...
}

@end
```

The next screenshots show what happens in Xcode. The first one is what happens after adding the revised class. Notice that there are three new UI elements for the three properties. The *Text Color* will display a color picker, *Text* is just an input box and *Show Text* will give us the options for Off and On which are `false` and `true` respectively.
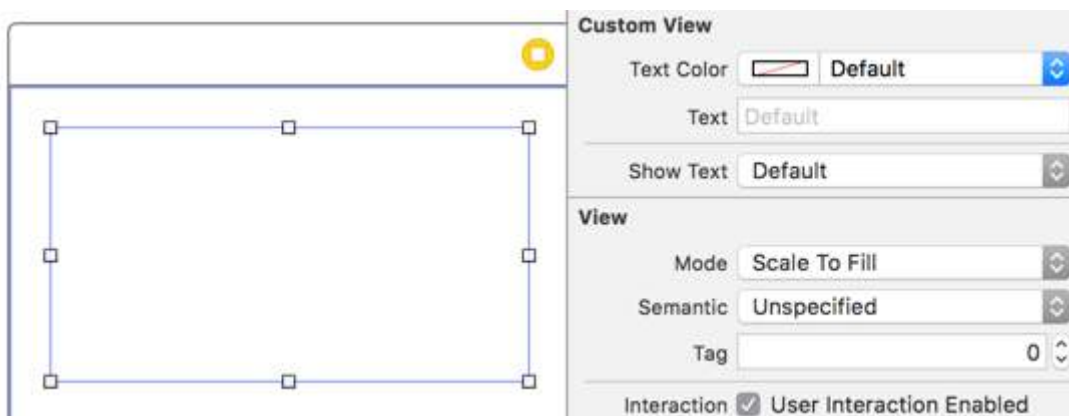


The next is after changing the *Text Color* to red using the color picker. Also, some text has been provided to make the `drawRect` function display it. Notice that the view in Interface Builder has been updated as well.
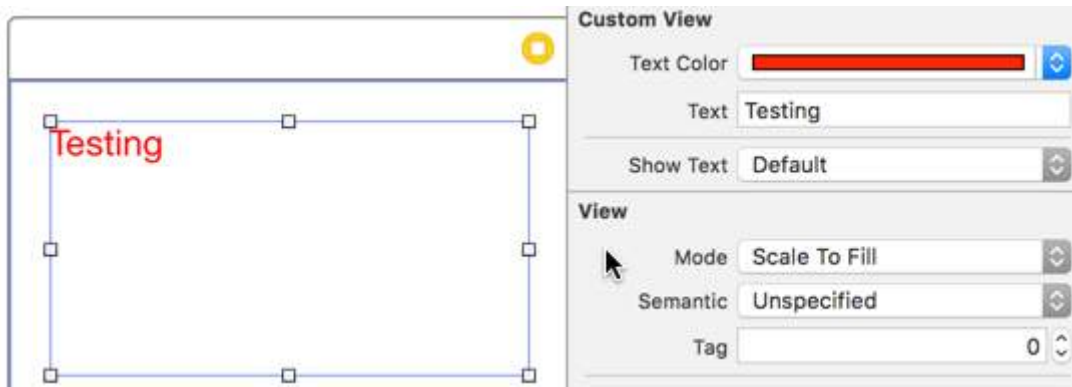
Finally, setting *Show Text* to Off in the property inspector makes the text display in Interface Builder disappear.
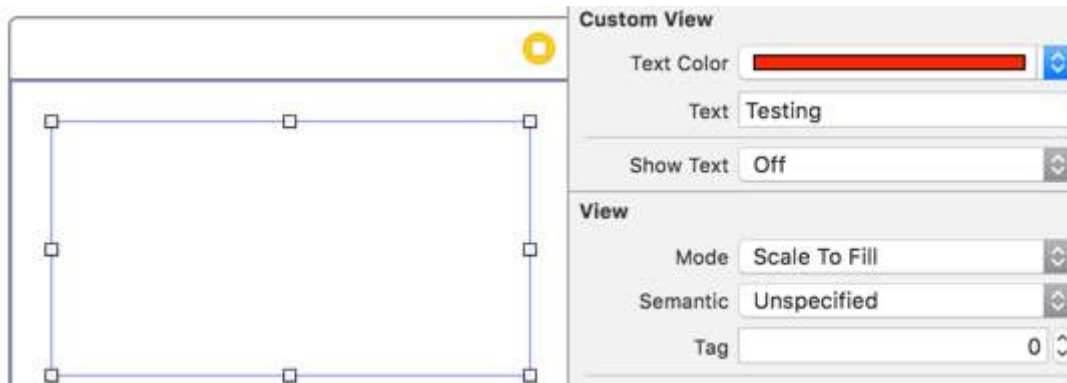


However, We all come up situation when we need to create rounded `UIView` at multiple views in your `Storyboard`.Instead of declaring `IBDesignable` to every views of `Storyboard`, its better to create an `Extension` of `UIView` and get a user interface built just for your every `UIView` through out the project to create rounded view by setting corner radius.A configurable border radius on any UIView you create in storyboard.

```swift
extension UIView {

    @IBInspectable var cornerRadius:CGFloat {
        set {
            layer.cornerRadius = newValue
            clipsToBounds = newValue > 0
        }
        get {
            return layer.cornerRadius
        }
    }

}
```
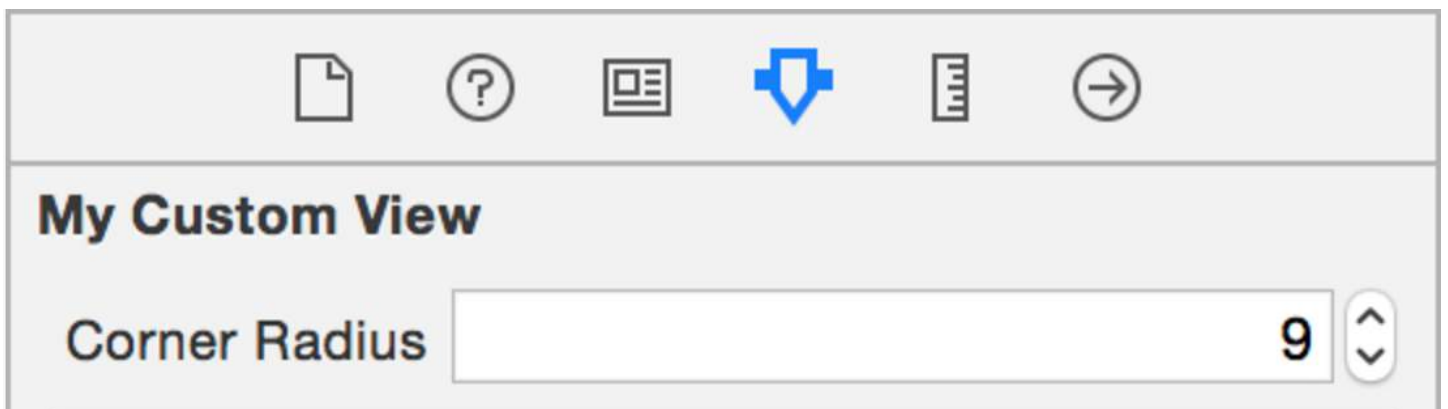
# Section 20.3: Taking a snapshot

You can take a snapshot from a `UIView` like this:

**Swift**

```
let snapshot = view.snapshotView(afterScreenUpdates: true)
```

**Objective-C**

```
UIView *snapshot = [view snapshotViewAfterScreenUpdates: YES];
```

# Section 20.4: Create a UIView

**Objective-C**

```
CGRect myFrame = CGRectMake(0, 0, 320, 35)
UIView *view = [[UIView alloc] initWithFrame:myFrame];

//Alternative way of defining the frame
UIView *view = [[UIView alloc] init];
CGRect myFrame = view.frame;
myFrame.size.width = 320;
myFrame.size.height = 35;
myFrame.origin.x = 0;
myFrame.origin.y = 0;
view.frame = myFrame;
```

**Swift**

```
let myFrame = CGRect(x: 0, y: 0, width: 320, height: 35)
let view = UIView(frame: myFrame)
```

# Section 20.5: Shake a View

```
extension UIView {
    func shake() {
        let animation = CAKeyframeAnimation(keyPath: "transform.translation.x")
        animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)
        animation.duration = 0.6
        animation.values = [-10.0, 10.0, -7.0, 7.0, -5.0, 5.0, 0.0 ]
        layer.add(animation, forKey: "shake")
    }
}
```

This function can be used to draw attention to a specific view by shaking it a bit.

# Section 20.6: Utilizing Intrinsic Content Size

When creating a UIView subclass, intrinsic content size helps to avoid setting hardcoded height and width constraints

a basic glimpse into how a class can utilize this

```
class ImageView: UIView {
    var image: UIImage {
```

```
        didSet {
            invalidateIntrinsicContentSize()
        }
    }
    // omitting initializers
    // convenience init(image: UIImage)

    override func intrinsicContentSize() -> CGSize {
        return CGSize(width: image.size.width, height: image.size.height)
    }
}
```

If you only want to provide one size intrinsically, you can provide the value `UIViewNoIntrinsicMetric` for the value that you wish to ignore.

```
override func intrinsicContentSize() -> CGSize {
    return CGSize(width: UIViewNoIntrinsicMetric, height: image.size.width)
}
```

**Benefits when using with AutoLayout and Interface Builder**

One could take this ImageView (or UIImageView) and set the horizontal alignment to superview center X and the vertical alignment to superview center Y.
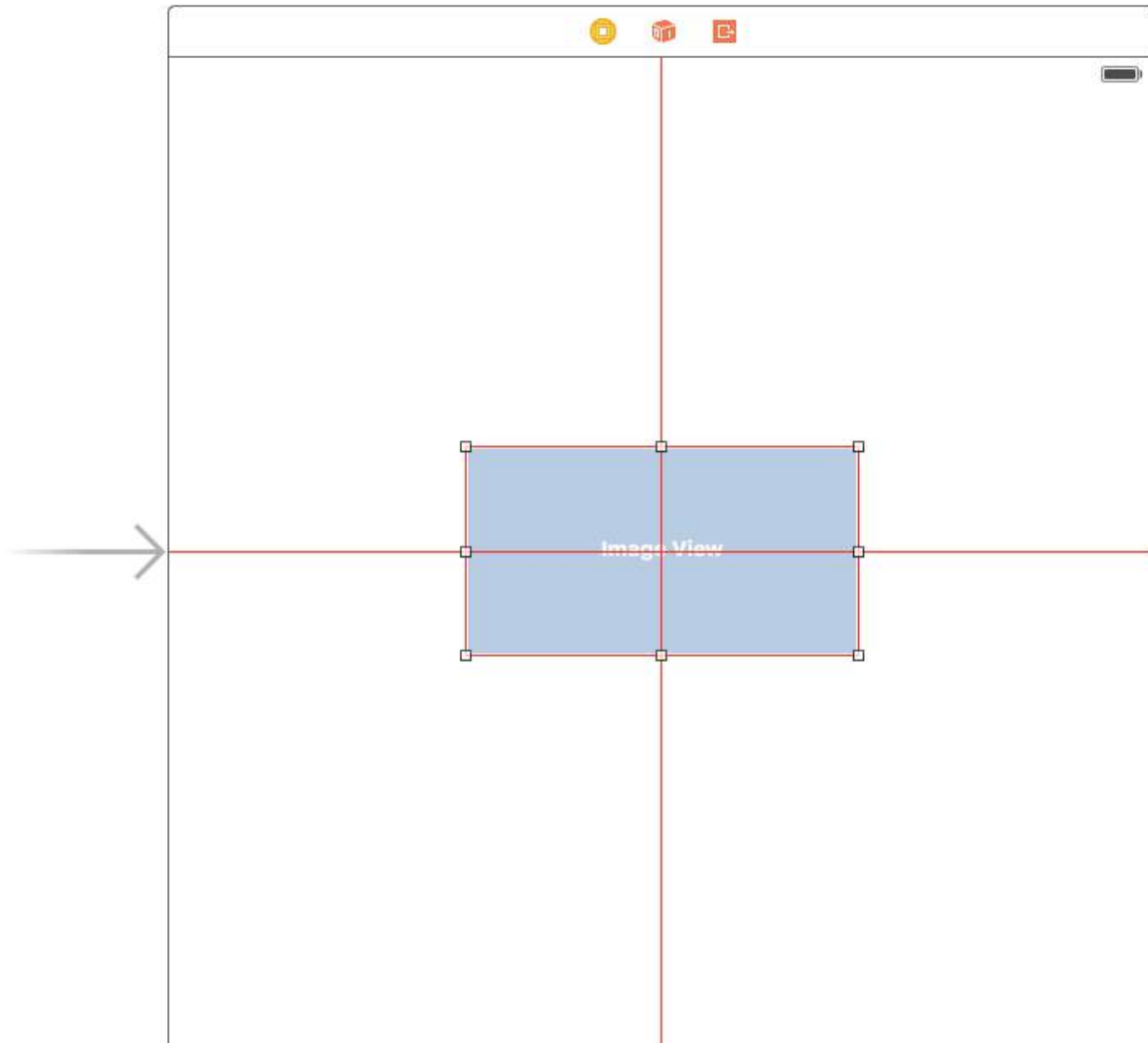
Interface builder will complain to you at this point giving the following warning:



This is where `Placeholder Intrinsic Size` comes in.

Going into the Size inspector panel, and down to the Intrinsic Size dropdown, you can switch this value from Default to Placeholder.
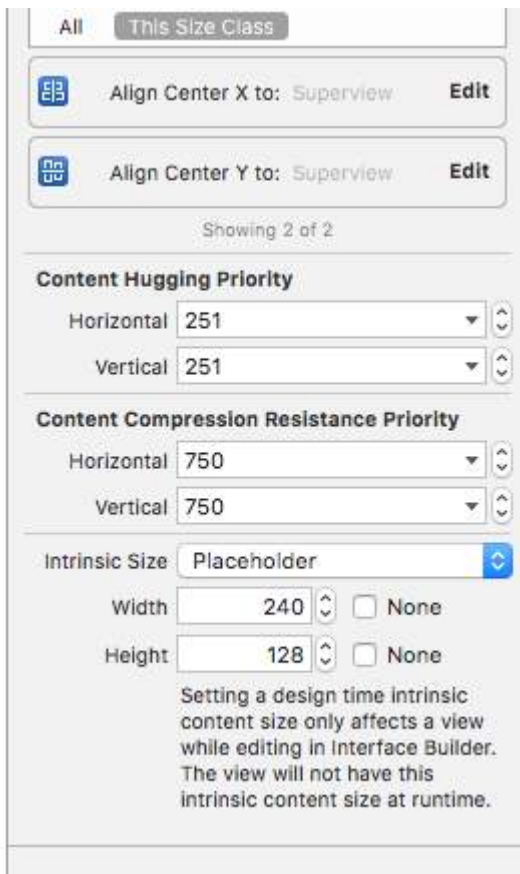
and now interface builder will remove the previous warnings and you can use this size to have dynamically sized views laid out in interface builder.

# Section 20.7: Programmatically manage UIView insertion and deletion into and from another UIView

Suppose you have a `parentView` into which you want to insert a new `subView` programmatically (eg. when you want to insert an `UIImageView` into a `UIViewController`'s view), than you can do it as below.

**Objective-C**

```
[parentView addSubview:subView];
```

**Swift**

```
parentView.addSubview(subView)
```

You can also add the subView below another `subView2`, which is already a sub view of parentView using the following code:

**Objective-C**

```
[parentView insertSubview:subView belowSubview:subView2];
```

**Swift**

```
parentView.insertSubview(subView, belowSubview: subView2)
```

If you want to insert it above `subView2` you can do it this way:

**Objective-C**

```objc
[parentView insertSubview:subView aboveSubview:subView2];
```

**Swift**

```swift
parentView.insertSubview(subView, aboveSubview: subView2)
```

If somewhere in your code you need to bring a certain `subView` to front, so above all the others `parentView`'s subviews, you can do it like this:

**Objective-C**

```objc
[parentView bringSubviewToFront:subView];
```

**Swift**

```swift
parentView.bringSubviewToFront(subView)
```

Finally, if you want to remove `subView` from `parentView`, you can do as below:

**Objective-C**

```objc
[subView removeFromSuperview];
```

**Swift**

```swift
subView.removeFromSuperview()
```

# Section 20.8: Create UIView using Autolayout

```objc
UIView *view = [[UIView alloc] init];

[self.view addSubview:view];

//Use the function if you want to use height as constraint
[self addView:view onParentView:self.view withHeight:200.f];


//Use this function if you want to add view with respect to parent and should resize with it
[self addFullResizeConstraintForSubview:view addedOnParentView:self.view];
```

**Functions**

**Function to add view with fixed height using autolayout constraints**

```objc
-(void)addView:(UIView*)subView onParentView:(UIView*)parentView withHeight:(CGFloat)height{

subView.translatesAutoresizingMaskIntoConstraints = NO;

NSLayoutConstraint *trailing =[NSLayoutConstraint
                               constraintWithItem:subView
                               attribute:NSLayoutAttributeTrailing
                               relatedBy:NSLayoutRelationEqual
                               toItem:parent
                               attribute:NSLayoutAttributeTrailing
```

```objc
                                multiplier:1.0
                                constant:10.f];


NSLayoutConstraint *top = [NSLayoutConstraint
                                constraintWithItem:subView
                                attribute:NSLayoutAttributeTop
                                relatedBy:NSLayoutRelationEqual
                                toItem:parent
                                attribute:NSLayoutAttributeTop
                                multiplier:1.0
                                constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
                                constraintWithItem:subView
                                attribute:NSLayoutAttributeLeading
                                relatedBy:NSLayoutRelationEqual
                                toItem:parent
                                attribute:NSLayoutAttributeLeading
                                multiplier:1.0
                                constant:10.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];

NSLayoutConstraint *heightConstraint =[NSLayoutConstraint
                                   constraintWithItem:subView
                                   attribute:NSLayoutAttributeHeight
                                   relatedBy:NSLayoutRelationEqual
                                   toItem:nil
                                   attribute:0
                                   multiplier:0.0
                                   constant:height];

[subView addConstraint:heightConstraint];

}
```

**Function add full resize constraint for created UIView.**

```objc
-(void)addFullResizeConstraintForSubview:(UIView*)subView addedOnParentView:(UIView*)parentView{

subView.translatesAutoresizingMaskIntoConstraints = NO;

NSLayoutConstraint *trailing =[NSLayoutConstraint
                                constraintWithItem:subView
                                attribute:NSLayoutAttributeTrailing
                                relatedBy:NSLayoutRelationEqual
                                toItem:parent
                                attribute:NSLayoutAttributeTrailing
                                multiplier:1.0
                                constant:10.f];


NSLayoutConstraint *top = [NSLayoutConstraint
                                constraintWithItem:subView
                                attribute:NSLayoutAttributeTop
                                relatedBy:NSLayoutRelationEqual
                                toItem:parent
                                attribute:NSLayoutAttributeTop
                                multiplier:1.0
```

```
                                 constant:10.f];

NSLayoutConstraint *leading = [NSLayoutConstraint
                              constraintWithItem:subView
                              attribute:NSLayoutAttributeLeading
                              relatedBy:NSLayoutRelationEqual
                              toItem:parent
                              attribute:NSLayoutAttributeLeading
                              multiplier:1.0
                              constant:10.f];

NSLayoutConstraint *bottom =[NSLayoutConstraint
                            constraintWithItem:subView
                            attribute:NSLayoutAttributeBottom
                            relatedBy:NSLayoutRelationEqual
                            toItem:parent
                            attribute:NSLayoutAttributeBottom
                            multiplier:1.0
                            constant:0.f];
[parent addConstraint:trailing];
[parent addConstraint:top];
[parent addConstraint:leading];
[parent addConstraint:bottom];
}
```

# Section 20.9: Animating a UIView

```
let view = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
view.backgroundColor = UIColor.orange
self.view.addSubview(view)
UIView.animate(withDuration: 0.75, delay: 0.5, options: .curveEaseIn, animations: {
    //This will cause view to go from (0,0) to
    // (self.view.frame.origin.x,self.view.frame.origin.y)
    view.frame.origin.x = self.view.frame.origin.x
    view.frame.origin.y = self.view.frame.origin.y
}) { (finished) in
    view.backgroundColor = UIColor.blueColor()
}
```

# Section 20.10: UIView extension for size and frame attributes

If we want to get the x-cordinate of origin of the view, then we need to write like:

```
view.frame.origin.x
```

For width, we need to write:

```
view.frame.size.width
```

But if we add a simple extension to an UIView, we can get all the attributes very simply, like:

```
view.x
view.y
view.width
view.height
```

It will also help setting these attributes like:

```
view.x = 10
view.y = 10
view.width = 100
view.height = 200
```

And the simple extension would be:

```
extension UIView {

    var x: CGFloat {
        get {
            return self.frame.origin.x
        }
        set {
            self.frame = CGRect(x: newValue, y: self.frame.origin.y, width: self.frame.size.width,
height: self.frame.size.height)
        }
    }

    var y: CGFloat {
        get {
            return self.frame.origin.y
        }
        set {
            self.frame = CGRect(x: self.frame.origin.x, y: newValue, width: self.frame.size.width,
height: self.frame.size.height)
        }
    }

    var width: CGFloat {
        get {
            return self.frame.size.width
        }
        set {
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width: newValue,
height: self.frame.size.height)
        }
    }

    var height: CGFloat {
        get {
            return self.frame.height
        }
        set {
            self.frame = CGRect(x: self.frame.origin.x, y: self.frame.origin.y, width:
self.frame.size.width, height: newValue)
        }
    }
}
```

We need to add this class file in a project and it'll be available to use throughout the project!

# Chapter 21: Snapshot of UIView

## Section 21.1: Getting the Snapshot

```
- (UIImage *)getSnapshot
{
UIScreen *screen = [UIScreen mainScreen];
CGRect bounds = [self.view bounds];
UIGraphicsBeginImageContextWithOptions(bounds.size, false, screen.scale);
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetInterpolationQuality(context, kCGInterpolationHigh);
[self.view drawViewHierarchyInRect:bounds afterScreenUpdates:YES];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
return image;
}
```

**Swift**

```
var screenshot: UIImage
{
UIGraphicsBeginImageContext(self.bounds.size);
let context = UIGraphicsGetCurrentContext();
self.layer.render(in: context)
let screenShot = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
return screenShot
}
```

## Section 21.2: Snapshot with subview with other markup and text

- Support Portrait and Landscape both type of image
- Drawing and other subviews can be merged in my case I'm adding label to draw

```
{
    CGSize fullSize = getImageForEdit.size;
    CGSize sizeInView = AVMakeRectWithAspectRatioInsideRect(imgViewFake.image.size,
imgViewFake.bounds).size;
    CGFloat orgScale = orgScale = fullSize.width/sizeInView.width;
    CGSize newSize = CGSizeMake(orgScale * img.image.size.width, orgScale * img.image.size.height);
    if(newSize.width <= fullSize.width && newSize.height <= fullSize.height){
        newSize = fullSize;
    }
    CGRect offsetRect;
    if (getImageForEdit.size.height > getImageForEdit.size.width){
        CGFloat scale = newSize.height/fullSize.height;
        CGFloat offset = (newSize.width - fullSize.width*scale)/2;
        offsetRect = CGRectMake(offset, 0, newSize.width-offset*2, newSize.height);
    }
    else{
        CGFloat scale = newSize.width/fullSize.width;
        CGFloat offset = (newSize.height - fullSize.height*scale)/2;
        offsetRect = CGRectMake(0, offset, newSize.width, newSize.height-offset*2);
    }
    UIGraphicsBeginImageContextWithOptions(newSize, NO, getImageForEdit.scale);
    [getImageForEdit drawAtPoint:offsetRect.origin];
    //        [img.image drawInRect:CGRectMake(0,0,newSize.width,newSize.height)];
```

```
    CGFloat oldScale = img.contentScaleFactor;
    img.contentScaleFactor = getImageForEdit.scale;
    [img drawViewHierarchyInRect:CGRectMake(0, 0, newSize.width, newSize.height)
afterScreenUpdates:YES];
    img.contentScaleFactor = oldScale;
    UIImage *combImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    imageData = UIImageJPEGRepresentation(combImage, 1);
}
```

# Chapter 22: UIAlertController

## Section 22.1: AlertViews with UIAlertController

`UIAlertView` and `UIActionSheet` are Deprecated in `iOS 8` and Later. So Apple introduced a new controller for AlertView and ActionSheet called `UIAlertController`, changing the `preferredStyle`, you can switch between AlertView and ActionSheet. There is no delegate method for it because all button events are handled in their blocks.

**Simple AlertView**

*Swift:*

```swift
let alert = UIAlertController(title: "Simple", message: "Simple alertView demo with Cancel and
OK.", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "Cancel", style: .cancel) { _ in
        print("Cancel")
})
alert.addAction(UIAlertAction(title: "OK", style: .default) { _ in
        print("OK")
})

present(alert, animated: true)
```

*Objective-C:*

```objc
 UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Simple"
message:@"Simple alertView demo with Cancel and OK." preferredStyle:UIAlertControllerStyleAlert];
    UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {
        NSLog(@"Cancel");
    }];
    UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
        NSLog(@"OK");
    }];

    [alertController addAction:cancelAction];
    [alertController addAction:okAction];
    [self presentViewController:alertController animated: YES completion: nil];
```
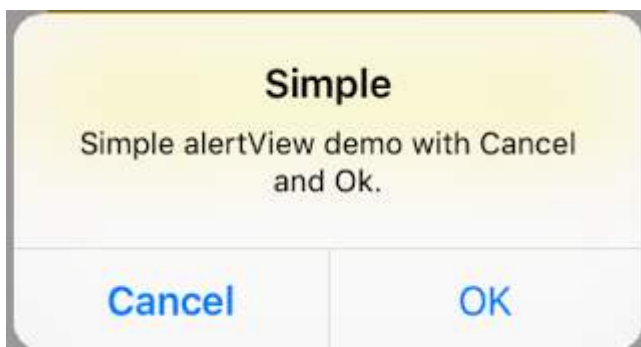


**Destructive AlertView**

*Swift:*

```
let alert = UIAlertController(title: "Simple", message: "Simple alertView demo with Cancel and
OK.", preferredStyle: .alert)

alert.addAction(UIAlertAction(title: "Destructive", style: .destructive) { _ in
        print("Destructive")
})
alert.addAction(UIAlertAction(title: "OK", style: .default) { _ in
        print("OK")
})

present(alert, animated: true)
```

***Objective-C:***

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Destructive"
message:@"Simple alertView demo with Destructive and OK."
preferredStyle:UIAlertControllerStyleAlert];
    UIAlertAction *destructiveAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
        NSLog(@"Destructive");
    }];
    UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"OK" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
        NSLog(@"OK");
    }];

    [alertController addAction:destructiveAction];
    [alertController addAction:okAction];
    [self presentViewController:alertController animated: YES completion: nil];
```
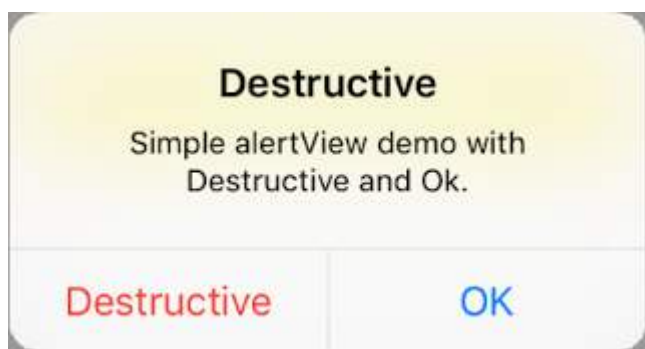


# Section 22.2: Action Sheets with UIAlertController

With `UIAlertController`, action sheets like the deprecated `UIActionSheet` are created with the same API as you use for AlertViews.

**Simple Action Sheet with two buttons**
**Swift**

```
let alertController = UIAlertController(title: "Demo", message: "A demo with two buttons",
preferredStyle: UIAlertControllerStyle.actionSheet)
```

**Objective-C**

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Demo"
message:@"A demo with two buttons" preferredStyle:UIAlertControllerStyleActionSheet];
```

Create the buttons "Cancel" and "Okay"

**Swift**

```
let cancelAction = UIAlertAction(title: "Cancel", style: .cancel) { (result : UIAlertAction) ->
Void in
    //action when pressed button
}
let okAction = UIAlertAction(title: "Okay", style: .default) { (result : UIAlertAction) -> Void in
    //action when pressed button
}
```

**Objective-C**

```
UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"Cancel"
style:UIAlertActionStyleCancel handler:^(UIAlertAction * action) {
        //action when pressed button
    }];

UIAlertAction * okAction = [UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
        //action when pressed button
    }];
```

And add them to the action sheet:

**Swift**

```
alertController.addAction(cancelAction)
alertController.addAction(okAction)
```

**Objective-C**

```
[alertController addAction:cancelAction];
[alertController addAction:okAction];
```

Now present the UIAlertController:

**Swift**

```
self.present(alertController, animated: true, completion: nil)
```

**Objective-C**

```
[self presentViewController:alertController animated: YES completion: nil];
```

This should be the result:
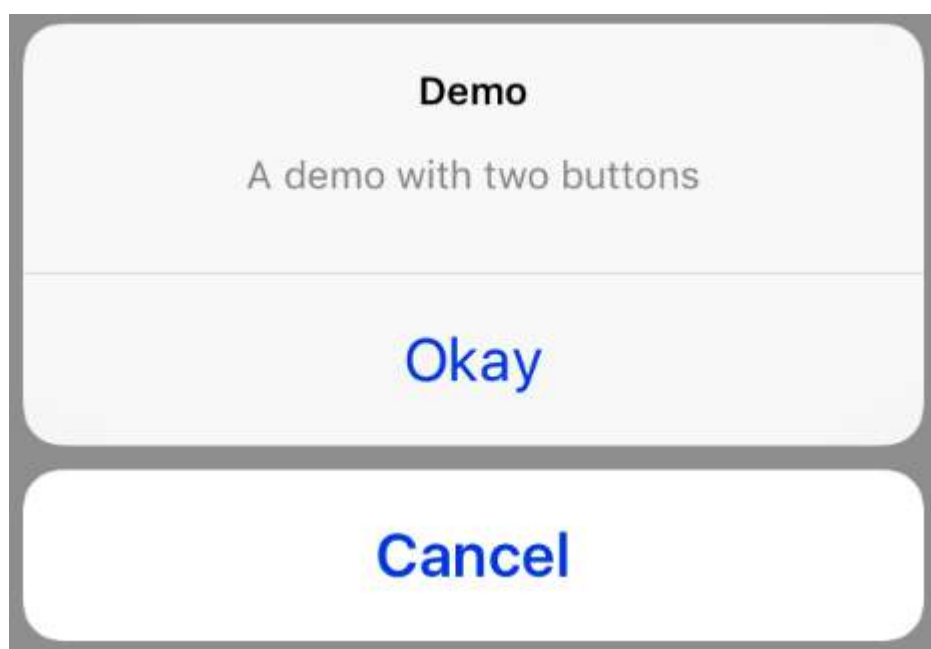
**Action Sheet with destructive button**

Using the UIAlertActionStyle `.destructive` for an `UIAlertAction` will create a button with red tint color.
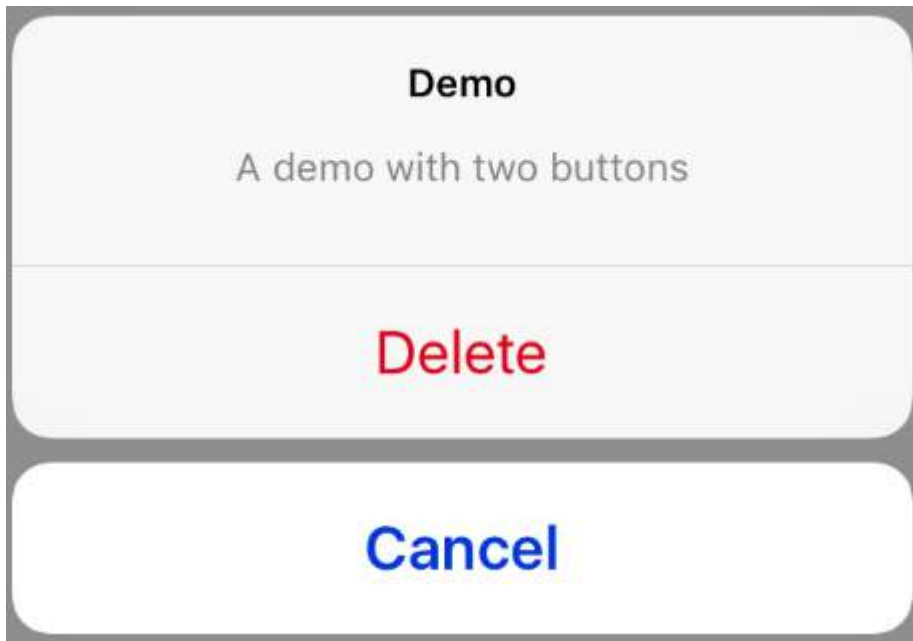


For this example, the `okAction` from above was replaced by this `UIAlertAction`:

**Swift**

```
let destructiveAction = UIAlertAction(title: "Delete", style: .destructive) { (result :
UIAlertAction) -> Void in
    //action when pressed button
}
```

**Objective-C**

```
UIAlertAction * destructiveAction = [UIAlertAction actionWithTitle:@"Delete"
style:UIAlertActionStyleDestructive handler:^(UIAlertAction * action) {
        //action when pressed button
    }];
```

# Section 22.3: Adding Text Field in UIAlertController like a prompt Box

**Swift**

```
let alert = UIAlertController(title: "Hello",
                             message: "Welcome to the world of iOS",
                             preferredStyle: UIAlertControllerStyle.alert)
let defaultAction = UIAlertAction(title: "OK", style: UIAlertActionStyle.default) { (action) in

}
defaultAction.isEnabled = false
alert.addAction(defaultAction)

alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.delegate = self
}

present(alert, animated: true, completion: nil)
```

**Objective-C**

```
UIAlertController* alert = [UIAlertController alertControllerWithTitle:@"Hello"
```

```
                                         message:@"Welcome to the world of
iOS"

preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
                                          style:UIAlertActionStyleDefault
                                     handler:^(UIAlertAction * action) {}];

defaultAction.enabled = NO;
[alert addAction:defaultAction];

[alert addTextFieldWithConfigurationHandler:^(UITextField *textField) {
    textField.delegate = self;
}];

[self presentViewController:alert animated:YES completion:nil];
```
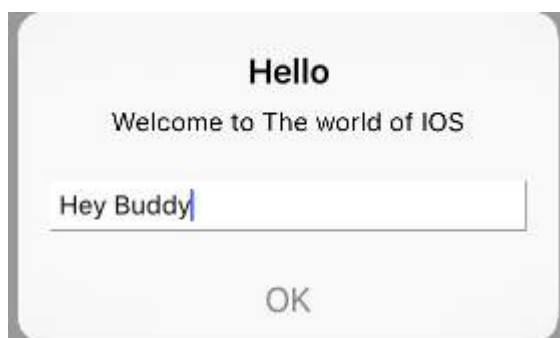


# Section 22.4: Highlighting an action button

Alert controller has a property which is used to put emphases on an action added in the alert controller. This property can be used to highlight a particular action for user attention.For objective C;

```
@property(nonatomic, strong) UIAlertAction *preferredAction
```

An action **which is already added in alert controller** can be assigned to this property.The Alert Controller will highlight this action.

**This property can only be used with UIAlertControllerStyleAlert.**

Following example shows how to use it.

```
UIAlertController *alertController = [UIAlertController alertControllerWithTitle:@"Cancel edit"
message:@"Are you really want to cancel your edit?" preferredStyle:UIAlertControllerStyleAlert];

UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"Cancel" style:UIAlertActionStyleCancel
handler:^(UIAlertAction * action) {
    NSLog(@"Cancel");
}];

UIAlertAction *no = [UIAlertAction actionWithTitle:@"NO" style:UIAlertActionStyleDefault
handler:^(UIAlertAction * action) {
    NSLog(@"Highlighted button is pressed.");
}];

[alertController addAction:cancel];
[alertController addAction:no];
```

```
//add no action to preffered action.
//Note
//the action should already be added to alert controller
alertController.preferredAction = no;

[self presentViewController:alertController animated: YES completion: nil];
```

Alert Controller with **preferred action set**.The **NO** button is highlighted.
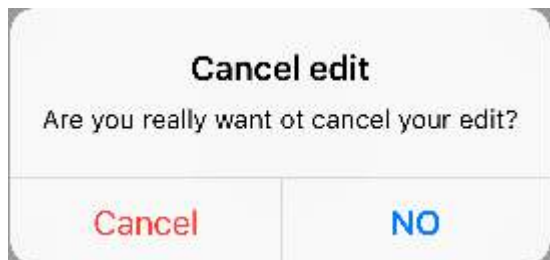


Alert Controller with **preferred action not set**.The **NO** button is not highlighted.
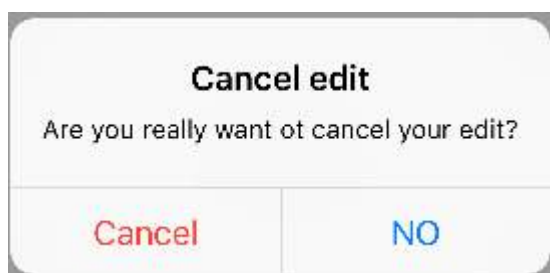


# Section 22.5: Displaying and handling alerts

**One Button**
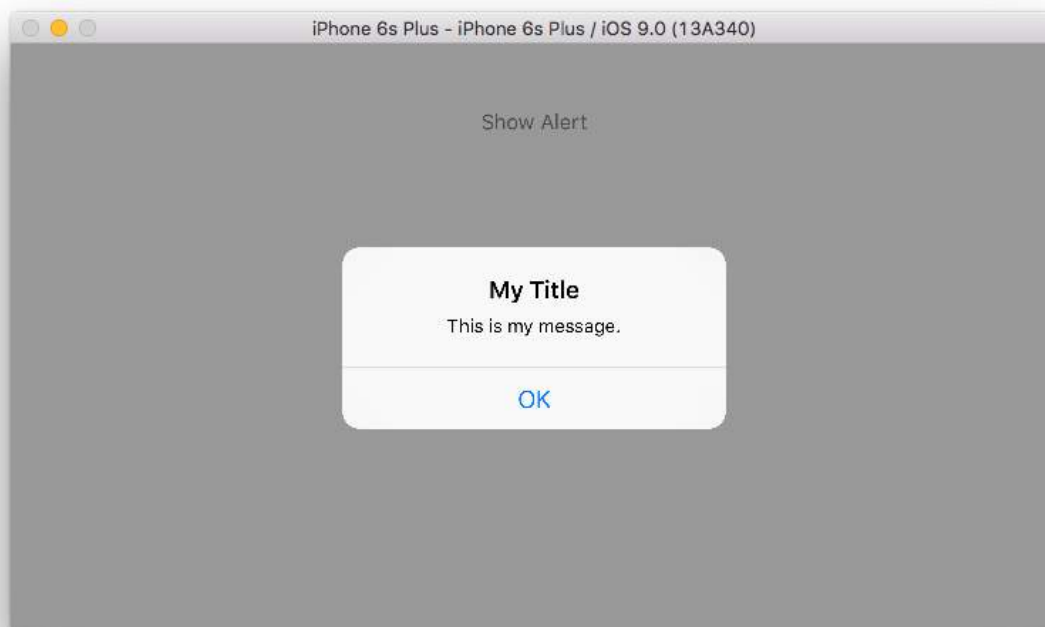


**Swift**

```
class ViewController: UIViewController {
```

```
    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // create the alert
        let alert = UIAlertController(title: "My Title", message: "This is my message.",
preferredStyle: UIAlertControllerStyle.Alert)

        // add an action (button)
        alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.Default, handler:
nil))

        // show the alert
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

**Two Buttons**
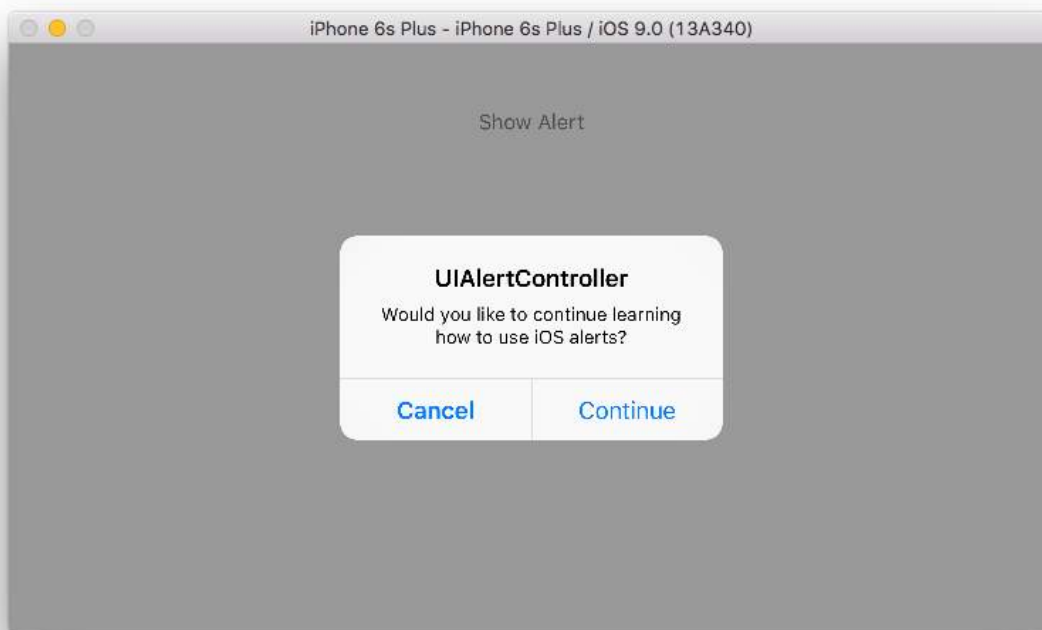


**Swift**

```
class ViewController: UIViewController {

    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // create the alert
        let alert = UIAlertController(title: "UIAlertController", message: "Would you like to
continue learning how to use iOS alerts?", preferredStyle: UIAlertControllerStyle.Alert)

        // add the actions (buttons)
        alert.addAction(UIAlertAction(title: "Continue", style: UIAlertActionStyle.Default,
handler: nil))
        alert.addAction(UIAlertAction(title: "Cancel", style: UIAlertActionStyle.Cancel, handler:
nil))

        // show the alert
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

**Three Buttons**
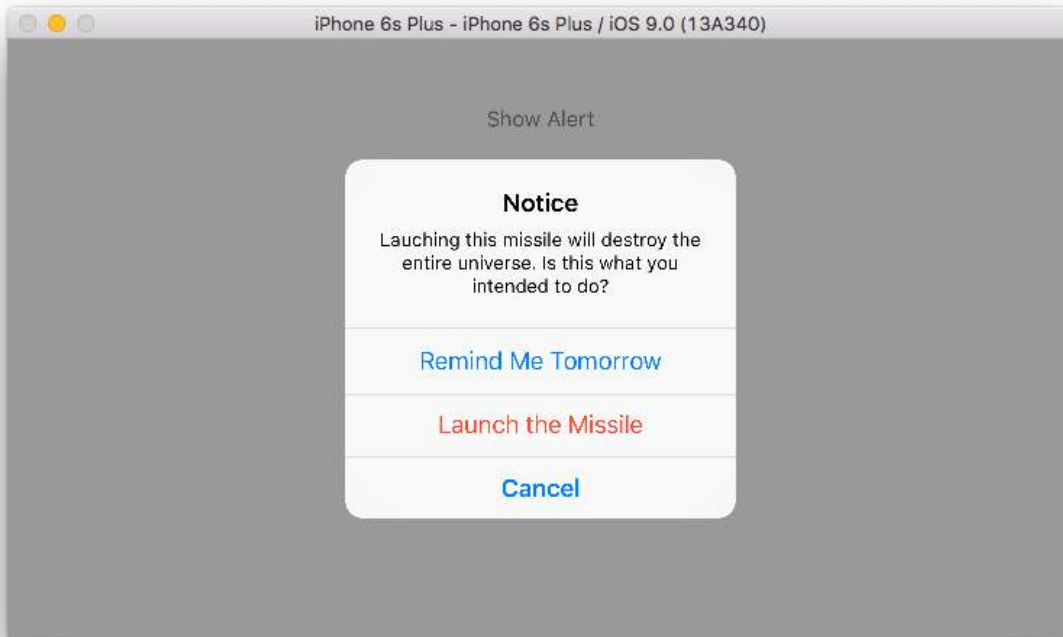
**Swift**

```swift
class ViewController: UIViewController {

    @IBAction func showAlertButtonTapped(sender: UIButton) {

        // create the alert
        let alert = UIAlertController(title: "Notice", message: "Lauching this missile will destroy
the entire universe. Is this what you intended to do?", preferredStyle:
UIAlertControllerStyle.Alert)

        // add the actions (buttons)
        alert.addAction(UIAlertAction(title: "Remind Me Tomorrow", style:
UIAlertActionStyle.Default, handler: nil))
        alert.addAction(UIAlertAction(title: "Cancel", style: UIAlertActionStyle.Cancel, handler:
nil))
        alert.addAction(UIAlertAction(title: "Launch the Missile", style:
UIAlertActionStyle.Destructive, handler: nil))

        // show the alert
        self.presentViewController(alert, animated: true, completion: nil)
    }
}
```

**Handling Button Taps**

The handler was nil in the above examples. You can replace nil with a [closure](#) to do something when the user taps a button, like the example below:

**Swift**

```swift
alert.addAction(UIAlertAction(title: "Launch the Missile", style: UIAlertActionStyle.Destructive,
handler: { action in

    // do something like...
    self.launchMissile()

}))
```

**Notes**

- Multiple buttons do not necessarily need to use different `UIAlertActionStyle` types. They could all be `.Default`.
- For more than three buttons consider using an Action Sheet. The setup is very similar. [Here is an example.](#)

# Chapter 23: UIColor

## Section 23.1: Creating a UIColor

There are many ways you can create a `UIColor`:

**Swift**

- Using one of the predefined colors:

```swift
let redColor = UIColor.redColor()
let blueColor: UIColor = .blueColor()

// In Swift 3, the "Color()" suffix is removed:
let redColor = UIColor.red
let blueColor: UIColor = .blue
```

If the compiler already knows that the variable is an instance of `UIColor` you can skip the type all together:

```swift
let view = UIView()
view.backgroundColor = .yellowColor()
```

- Using the grayscale value and the alpha:

```swift
let grayscaleColor = UIColor(white: 0.5, alpha: 1.0)
```

- Using hue, saturation, brightness and alpha:

```swift
let hsbColor = UIColor(
    hue: 0.4,
    saturation: 0.3,
    brightness: 0.7,
    alpha: 1.0
)
```

- Using the RGBA values:

```swift
let rgbColor = UIColor(
    red: 30.0 / 255,
    green: 70.0 / 255,
    blue: 200.0 / 255,
    alpha: 1.0
)
```

- Using a pattern image:

```swift
let patternColor = UIColor(patternImage: UIImage(named: "myImage")!)
```

**Objective-C**

- Using one of the predefined colors:

```objc
UIColor *redColor = [UIColor redColor];
```

- Using the grayscale value and the alpha:

```
UIColor *grayscaleColor = [UIColor colorWithWhite: 0.5 alpha: 1.0];
```

- Using hue, saturation, brightness and alpha:

```
UIColor *hsbColor = [UIColor
    colorWithHue: 0.4
    saturation: 0.3
    brightness: 0.7
    alpha: 1.0
];
```

- Using the RGBA values:

```
UIColor *rgbColor = [UIColor
    colorWithRed: 30.0 / 255.0
    green: 70.0 / 255.0
    blue: 200.0 / 255.0
    alpha: 1.0
];
```

- Using a pattern image:

```
UIColor *pattenColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"myImage.png"]];
```

# Section 23.2: Creating a UIColor from hexadecimal number or string

You can create a `UIColor` from a hexadecimal number or string, e.g. 0xff00cc, "#FFFFFF"

**Swift**

*Int Value*

```
extension UIColor {
    convenience init(hex: Int, alpha: CGFloat = 1.0) {
        let r = CGFloat((hex >> 16) & 0xff) / 255
        let g = CGFloat((hex >> 08) & 0xff) / 255
        let b = CGFloat((hex >> 00) & 0xff) / 255
        self.init(red: r, green: g, blue: b, alpha: alpha)
    }
}
```

Example:

```
let color = UIColor(hex: 0xff00cc, alpha: 1.0)
```

Note that for `alpha` the default value of `1.0` is provided, so it can be used as follows:

```
let color = UIColor(hex: 0xff00cc)
```

*String Value*

```swift
extension UIColor {
    convenience init(hexCode: String) {
        let hex =
hexCode.stringByTrimmingCharactersInSet(NSCharacterSet.alphanumericCharacterSet().invertedSet)
        var int = UInt32()
        NSScanner(string: hex).scanHexInt(&int)
        let a, r, g, b: UInt32

        switch hex.characters.count {
        case 3:
            (a, r, g, b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
        case 6:
            (a, r, g, b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
        case 8:
            (a, r, g, b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
        default:
            (a, r, g, b) = (1, 1, 1, 0)
        }

        self.init(red: CGFloat(r) / 255, green: CGFloat(g) / 255, blue: CGFloat(b) / 255, alpha:
CGFloat(a) / 255)
    }
}
```

Example Usage:

Hex with alpha

```swift
let color = UIColor("#80FFFFFF")
```

Hex with no alpha (`color` alpha will equal 1.0)

```swift
let color = UIColor("#FFFFFF")
let color = UIColor("#FFF")
```

**Objective-C**

*Int Value*

```objc
@interface UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha;
@end

@implementation UIColor (Hex)
+ (UIColor *)colorWithHex:(NSUInteger)hex alpha:(CGFloat)alpha {
    return [UIColor colorWithRed:((CGFloat)((hex & 0xFF0000) >> 16))/255.0
                           green:((CGFloat)((hex & 0xFF00) >> 8))/255.0
                            blue:((CGFloat)(hex & 0xFF))/255.0
                           alpha:alpha];
}
@end
```

Example:

```objc
UIColor *color = [UIColor colorWithHex:0xff00cc alpha:1.0];
```

*String Value*

```objc
- (UIColor*) hex:(NSString*)hexCode {

    NSString *noHashString = [hexCode stringByReplacingOccurrencesOfString:@"#" withString:@""];
    NSScanner *scanner = [NSScanner scannerWithString:noHashString];
    [scanner setCharactersToBeSkipped:[NSCharacterSet symbolCharacterSet]];

    unsigned hex;
    if (![scanner scanHexInt:&hex]) return nil;
    int a;
    int r;
    int g;
    int b;

    switch (noHashString.length) {
        case 3:
            a = 255;
            r = (hex >> 8) * 17;
            g = ((hex >> 4) & 0xF) * 17;
            b = ((hex >> 0) & 0xF) * 17;
            break;
        case 6:
            a = 255;
            r = (hex >> 16);
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;
        case 8:
            a = (hex >> 24);
            r = (hex >> 16) & 0xFF;
            g = (hex >> 8) & 0xFF;
            b = (hex) & 0xFF;
            break;

        default:
            a = 255.0;
            r = 255.0;
            b = 255.0;
            g = 255.0;
            break;
    }

    return [UIColor colorWithRed:r / 255.0f green:g / 255.0f blue:b / 255.0f alpha:a / 255];
}
```

Example usage:

Hex with alpha

```objc
UIColor* color = [self hex:@"#80FFFFFF"];
```

Hex with no alpha (`color` alpha will equal 1)

```objc
UIColor* color = [self hex:@"#FFFFFF"];
UIColor* color = [self hex:@"#FFF"];
```

# Section 23.3: Color with Alpha component

You can set the opacity to a certain `UIColor` without creating a new one using the
`init(red:_,green:_,blue:_,alpha:_)` initializer.

**Swift**

```
let colorWithAlpha = UIColor.redColor().colorWithAlphaComponent(0.1)
```

**Swift 3**

```
//In Swift Latest Version
_ colorWithAlpha = UIColor.red.withAlphaComponent(0.1)
```

**Objective-C**

```
UIColor * colorWithAlpha = [[UIColor redColor] colorWithAlphaComponent:0.1];
```

# Section 23.4: Undocumented Methods

There are a variety of undocumented methods on `UIColor`which expose alternate colors or functionality. These can be found in the [UIColor private header file](). I will document the use of two private methods, `styleString()` and `_systemDestructiveTintColor()`.

**styleString**

Since iOS 2.0 there is a private instance method on `UIColor` called `styleString` which returns an RGB or RGBA string representation of the color, even for colors like `whiteColor` outside the RGB space.

Objective-C:

```
@interface UIColor (Private)

- (NSString *)styleString;

@end

// ...

[[UIColor whiteColor] styleString]; // rgb(255,255,255)
[[UIColor redColor] styleString]; // rgb(255,0,0)
[[UIColor lightTextColor] styleString]; // rgba(255,255,255,0.600000)
```

In Swift you could use a bridging header to expose the interface. With pure Swift, you will need to create an `@objc` protocol with the private method, and `unsafeBitCast UIColor` with the protocol:

```
@objc protocol  UIColorPrivate {
    func styleString() -> String
}

let white = UIColor.whiteColor()
let red = UIColor.redColor()
let lightTextColor = UIColor.lightTextColor()

let whitePrivate = unsafeBitCast(white, UIColorPrivate.self)
let redPrivate = unsafeBitCast(red, UIColorPrivate.self)
let lightTextColorPrivate = unsafeBitCast(lightTextColor, UIColorPrivate.self)

whitePrivate.styleString() // rgb(255,255,255)
redPrivate.styleString() // rgb(255,0,0)
lightTextColorPrivate.styleString() // rgba(255,255,255,0.600000)
```
**_systemDestructiveTintColor()**

There is an undocumented class method on `UIColor` called `_systemDestructiveTintColor` which will return the red color used by destructive system buttons:

```swift
let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue()
```

It returns an unmanaged object, which you must call `.takeUnretainedValue()` on, since the color ownership has not been transferred to our own object.

As with any undocumented API, you should take caution when trying to use this method:

```swift
if UIColor.respondsToSelector("_systemDestructiveTintColor") {
    if let red = UIColor.performSelector("_systemDestructiveTintColor").takeUnretainedValue() as?
UIColor {
        // use the color
    }
}
```

or by using a protocol:

```swift
@objc protocol UIColorPrivateStatic {
    func _systemDestructiveTintColor() -> UIColor
}

let privateClass = UIColor.self as! UIColorPrivateStatic
privateClass._systemDestructiveTintColor() // UIDeviceRGBColorSpace 1 0.231373 0.188235 1
```

## Section 23.5: UIColor from an image pattern

You can create a `UIColor` object using an image pattern by using the `UIColor(patternImage:_)` method.

```swift
btn.backgroundColor = UIColor(patternImage: UIImage(named: "image")!)
```
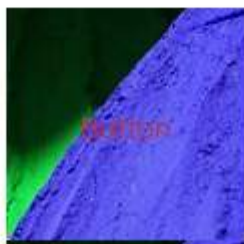
## Section 23.6: Lighter and Darker Shade of a given UIColor

The code example below demonstrate how you can get a lighter and darker shade of a given color, useful in applications having dynamic themes

For **Darker Color**

```
+ (UIColor *)darkerColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
        return [UIColor colorWithRed:MAX(r - 0.2, 0.0)
                               green:MAX(g - 0.2, 0.0)
                                blue:MAX(b - 0.2, 0.0)
                               alpha:a];
    return nil;
}
```

For **Lighter Color**

```
+ (UIColor *)lighterColorForColor:(UIColor *)c
{
    CGFloat r, g, b, a;
    if ([c getRed:&r green:&g blue:&b alpha:&a])
```

```
        return [UIColor colorWithRed:MIN(r + 0.2, 1.0)
                               green:MIN(g + 0.2, 1.0)
                                blue:MIN(b + 0.2, 1.0)
                               alpha:a];
    return nil;
}
```

See Visual differences below, considering given color is [UIColor orangeColor]



## Section 23.7: Make user defined attributes apply the CGColor datatype

By default, Interface Builder doesn't accept the CGColor datatype, so to allow adding a CGColor using user defined attributes in interface builder; one may want to use an extension like this:

Swift Extension :

```
extension CALayer {
    func borderUIColor() -> UIColor? {
        return borderColor != nil ? UIColor(CGColor: borderColor!) : nil
    }

    func setBorderUIColor(color: UIColor) {
        borderColor = color.CGColor
    }
}
```

**The new user defined attribute (borderUIColor) will be recognized and applied without problems.**

# Chapter 24: UITextView

## Section 24.1: Set attributed text

```swift
// Modify some of the attributes of the attributed string.
let attributedText = NSMutableAttributedString(attributedString: textView.attributedText!)

// Use NSString so the result of rangeOfString is an NSRange.
let text = textView.text! as NSString

// Find the range of each element to modify.
let tintedRange = text.range(of: NSLocalizedString("tinted", comment: ""))
let highlightedRange = text.range(of: NSLocalizedString("highlighted", comment: ""))

// Add tint.
attributedText.addAttribute(NSForegroundColorAttributeName, value: UIColor.blue, range:
tintedRange)

// Add highlight.
attributedText.addAttribute(NSBackgroundColorAttributeName, value: UIColor.yellow, range:
highlightedRange)

textView.attributedText = attributedText
```

## Section 24.2: Change font

**Swift**

```swift
//System Font
textView.font = UIFont.systemFont(ofSize: 12)

//Font of your choosing
textView.font = UIFont(name: "Font Name", size: 12)
```

**Objective-C**

```objectivec
//System Font
textView.font = [UIFont systemFontOfSize:12];

//Font of your choosing
textView.font = [UIFont fontWithName:@"Font Name" size:12];
```

## Section 24.3: Auto Detect Links, Addresses, Dates, and more

UITextView has built in support to auto detect a variety of data. The data that is able to be auto-detected currently
includes:

```
enum {
   UIDataDetectorTypePhoneNumber   = 1 << 0,
   UIDataDetectorTypeLink          = 1 << 1,
   UIDataDetectorTypeAddress       = 1 << 2,
   UIDataDetectorTypeCalendarEvent = 1 << 3,
   UIDataDetectorTypeNone          = 0,
   UIDataDetectorTypeAll           = NSUIntegerMax
};
```

**Enabling auto-detection**

```
// you may add as many as you like by using the `|` operator between options
textView.dataDetectorTypes = (UIDataDetectorTypeLink | UIDataDetectorTypePhoneNumber);
```

If enabled, the text will appear as a hyperlink on the `UITextView`

**Clickable data**

To allow the link to be clicked (which will result in different actions depending on the data type) you must ensure that the `UITextView` is selectable but not editable and that user interaction is enabled

```
textView.editable = NO;
textView.selectable = YES;
textView.userInteractionEnabled = YES; // YES by default
```

# Section 24.4: Change text

**Swift**

```
textView.text = "Hello, world!"
```

**Objective-C:**

```
textView.text = @"Hello, world!";
```

# Section 24.5: Change text alignment

**Swift**

```
textView.textAlignment = .left
```

**Objective-C**

```
textView.textAlignment = NSTextAlignmentLeft;
```

# Section 24.6: UITextViewDelegate methods

**Responding to Editing Notifications**

- textViewShouldBeginEditing(_:)
- textViewDidBeginEditing(_:)
- textViewShouldEndEditing(_:)
- textViewDidEndEditing(_:)

**Responding to Text Changes**

- textView(_:shouldChangeTextIn:replacementText:)
- textViewDidChange(_:)

**Responding to URL**

- textView(_: UITextView, shouldInteractWithURL: NSURL, inRange: NSRange) -> Bool

# Section 24.7: Change text color

**Swift**

```swift
textView.textColor = UIColor.red
```

**Objective-C**

```objectivec
textView.textColor = [UIColor redColor];
```

# Section 24.8: Remove extra paddings to fit to a precisely measured text

`UITextView` has extra paddings by default. Sometimes it's annoying especially if you want to measure some text without view instance and place them at some area precisely.

Do this to remove such paddings.

```swift
messageTextView.textContainerInset = UIEdgeInsetsZero
messageTextView.textContainer.lineFragmentPadding = 0
```

Now you can measure text size using `NSAttributedString.boundingRectWithSize(...)`, and resize a `UITextView` just to fit it to the text.

```swift
let budget = getSomeCGSizeBudget()
let text = getSomeAttributedString()
let textSize = text.boundingRectWithSize(budget, options: [.UsesLineFragmentOrigin,
.UsesFontLeading], context: nil).size
messageTextView.frame.size = textSize // Just fits.
```

# Section 24.9: Getting and Setting the Cursor Postition

**Useful information**

The very beginning of the text field text:

```swift
let startPosition: UITextPosition = textView.beginningOfDocument
```

The very end of the text field text:

```swift
let endPosition: UITextPosition = textView.endOfDocument
```

The currently selected range:

```swift
let selectedRange: UITextRange? = textView.selectedTextRange
```

**Get cursor position**

```swift
if let selectedRange = textView.selectedTextRange {

    let cursorPosition = textView.offsetFromPosition(textView.beginningOfDocument, toPosition:
selectedRange.start)

    print("\(cursorPosition)")
}
```

**Set cursor position**

In order to set the position, all of these methods are actually setting a range with the same start and end values.

**To the beginning**

```
let newPosition = textView.beginningOfDocument
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To the end**

```
let newPosition = textView.endOfDocument
textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To one position to the left of the current cursor position**

```
// only if there is a currently selected range
if let selectedRange = textView.selectedTextRange {

    // and only if the new position is valid
    if let newPosition = textView.positionFromPosition(selectedRange.start, inDirection:
UITextLayoutDirection.Left, offset: 1) {

        // set the new position
        textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:
newPosition)
    }
}
```

**To an arbitrary position**

Start at the beginning and move 5 characters to the right.

```
let arbitraryValue: Int = 5
if let newPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: arbitraryValue) {

    textView.selectedTextRange = textView.textRangeFromPosition(newPosition, toPosition:
newPosition)
}
```

**Related**

**Select all text**

```
textView.selectedTextRange = textView.textRangeFromPosition(textView.beginningOfDocument,
toPosition: textView.endOfDocument)
```

**Select a range of text**

```
// Range: 3 to 7
let startPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: 3)
let endPosition = textView.positionFromPosition(textView.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: 7)

if startPosition != nil && endPosition != nil {
    textView.selectedTextRange = textView.textRangeFromPosition(startPosition!, toPosition:
endPosition!)
}
```

**Insert text at the current cursor position**

```
textView.insertText("Hello")
```

**Notes**

- This example originally comes from an adaptation of [this Stack Overflow answer](#).

- This answer uses a text field, but the same concepts apply to `UITextView`.

- Use `textView.becomeFirstResponder()` to give focus to the text field and make the keyboard appear.

- See [this answer](#) for how to get the text at some range.

**Related**

- [How to Create a Range in Swift](#) (Deals indirectly with the issue of why we have to use `selectedTextRange` here rather than just `selectedRange`)

# Section 24.10: UITextView with HTML text

```
NSString *htmlString = @"<p> This is an <b>HTML</b> text</p>";
NSAttributedString *attributedString = [[NSMutableAttributedString alloc]
                                            initWithData: [htmlString
dataUsingEncoding:NSUnicodeStringEncoding]

                                            options: @{
NSDocumentTypeDocumentAttribute: NSHTMLTextDocumentType }
                                            documentAttributes: nil
                                            error: nil
                                            ];
            _yourTextView.attributedText = attributedString;
            // If you want to modify the font
            field.font = [UIFont fontWithName:@"Raleway-Regular" size:15];
```

# Section 24.11: Check to see if empty or nil

**Swift**

```
if let text = self.textView.text where !text.isEmpty {
    // Do stuff for text
} else {
    // Do stuff for nil text or empty string
}
```

**Objective-C**

```
if (self.textView.text.length > 0){
    // Do stuff for text
}   else {
    // Do stuff for nil text or empty string
}
```

# Chapter 25: UITextField Delegate

## Section 25.1: Actions when a user has started/ended interacting with a textfield

For Swift 3.1:

In the first example one can see how you would intercept the user interacting with a textfield while writing. Similarly, there are methods in the UITextFieldDelegate that are called when a user has started and ended his interaction with a TextField.

To be able to access these methods, you need to conform to the UITextFieldDelegate protocol, and for each textfield you want to be notified about, assign the parent class as the delegate:

```
class SomeClass: UITextFieldDelegate {

    @IBOutlet var textField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        textField.delegate = self
    }

}
```

Now you will be able to implement all the UITextFieldDelegate methods.

To be notified when a user has started editing a textfield, you can implement textFieldDidBeginEditing(_:) method like so:

```
func textFieldDidBeginEditing(_ textField: UITextField) {
    // now you can perform some action
    // if you have multiple textfields in a class,
    // you can compare them here to handle each one separately
    if textField == emailTextField {
        // e.g. validate email
    }
    else if textField == passwordTextField {
        // e.g. validate password
    }
}
```

Similarly, being notified if a user has ended interaction with a textfield, you can use the textFieldDidEndEditing(_:) method like so:

```
func textFieldDidEndEditing(_ textField: UITextField) {
    // now you can perform some action
    // if you have multiple textfields in a class,
    // you can compare them here to handle each one separately
    if textField == emailTextField {
        // e.g. validate email
    }
    else if textField == passwordTextField {
        // e.g. validate password
    }
}
```

If you want to have control over whether a TextField should begin/end editing, the textFieldShouldBeginEditing(_:) and textFieldShouldEndEditing(_:) methods can be used by return true/false based on your needed logic.

# Section 25.2: UITextField - Restrict textfield to certain characters

If you want to perform a user input validation of your textfield use the following code snippet:

```
// MARK: - UITextFieldDelegate

let allowedCharacters =
CharacterSet(charactersIn:"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz").inverted


func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange, replacementString
string: String) -> Bool {

    let components = string.components(separatedBy: allowedCharacters)
    let filtered = components.joined(separator: "")

    if string == filtered {

        return true

    } else {

        return false
    }
}
```

**Objective-C**

```
#define ACCEPTABLE_CHARACTERS @"0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

- (BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
{
    NSCharacterSet *cs = [[NSCharacterSet
characterSetWithCharactersInString:ACCEPTABLE_CHARACTERS] invertedSet];

    NSString *filtered = [[string componentsSeparatedByCharactersInSet:cs]
componentsJoinedByString:@""];

    return [string isEqualToString:filtered];
}
```

In addition you can also use character sets provided by apple to perform validation:

Take a look at https://developer.apple.com/reference/foundation/nscharacterset

```
let allowedCharacters = CharacterSet.alphanumerics.inverted
let allowedCharacters = CharacterSet.capitalizedLetters.inverted
```

# Chapter 26: UINavigationController

## Section 26.1: Embed a view controller in a navigation controller programmatically

Swift

```
//Swift
let viewController = UIViewController()
let navigationController = UINavigationController(rootViewController: viewController)

//Objective-C
UIViewController *viewController = [[UIViewController alloc] init];
UINavigationController *navigationController = [[UINavigationController alloc]
initWithRootViewController:viewController];
```

## Section 26.2: Popping in a Navigation Controller

**To previous view controller**

To pop back to the previous page you can do this:

Swift

```
navigationController?.popViewControllerAnimated(true)
```

Objective-C

```
[self.navigationController popViewControllerAnimated:YES];
```

**To root view controller**

To pop to the root of the navigation stack, you can do this:

Swift

```
navigationController?.popToRootViewControllerAnimated(true)
```

Objective C

```
[self.navigationController popToRootViewControllerAnimated:YES];
```

## Section 26.3: Purpose

UINavigationController is used to form a tree-like hierarchy of view controllers, which is known as a `navigation stack`.

**From developers perspective:**

You can connect independently made controller and get all the benefits of a free hierarchy manager and common UI presenter gratis. UINavigationController animates the transition to new controllers and provides the back functionality for you automatically. UINavigationController also gives access to all the other controllers in the `navigation stack` which can help access to some functionality or data.

**From user's perspective:**

`UINavigationController` helps to remember where user is at the moment (navigation bar title) and how he can go back (embedded back button) to one of the previous screens.

# Section 26.4: Pushing a view controller onto the navigation stack

```swift
//Swift
let fooViewController = UIViewController()
navigationController?.pushViewController(fooViewController, animated: true)

//Objective-C
UIViewController *fooViewController = [[UIViewController alloc] init];
[navigationController pushViewController:fooViewController animated:YES];
```

# Section 26.5: Creating a NavigationController

In your storyboard select the ViewController that you want to embed into a Navigation Controller.

Then navigate to Editor > Embed In > Navigation Controller
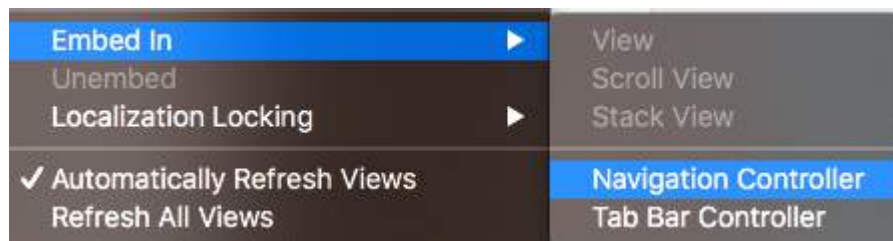


And that will create your navigation controller

# Chapter 27: UIGestureRecognizer

## Section 27.1: UITapGestureRecognizer

Initialize the `UITapGestureRecognizer` with a target, `self` in this case, and an `action` which is a method that has a single parameter: a `UITapGestureRecognizer`.

After initialization, add it to the view that it should recognize taps in.

**Swift**

```swift
override func viewDidLoad() {
    super.viewDidLoad()
    let recognizer = UITapGestureRecognizer(target: self,
                                            action: #selector(handleTap(_:)))
    view.addGestureRecognizer(recognizer)
}

func handleTap(recognizer: UITapGestureRecognizer) {

}
```

**Objective-C**

```objectivec
- (void)viewDidLoad {
    [super viewDidLoad];
    UITapGestureRecognizer *recognizer =
        [[UITapGestureRecognizer alloc] initWithTarget:self
                                        action:@selector(handleTap:)];
    [self.view addGestureRecognizer:recognizer];
}

- (void)handleTap:(UITapGestureRecognizer *)recognizer {

}
```

**Example of keyboard dismissal through UITapGestureRecognizer:**

First, you create the function for dismissing the keyboard:

```swift
func dismissKeyboard() {
        view.endEditing(true)
    }
```

Then, you add a tap gesture recognizer in your view controller, calling the method we just made

```swift
let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action: "dismissKeyboard")
        view.addGestureRecognizer(tap)
```

**Example of getting gesture location UITapGestureRecognizer (Swift 3):**

```swift
func handleTap(gestureRecognizer: UITapGestureRecognizer) {
print("tap working")
if gestureRecognizer.state == UIGestureRecognizerState.recognized
    {
      print(gestureRecognizer.location(in: gestureRecognizer.view))
```

```
    }
}
```

# Section 27.2: UITapGestureRecognizer (Double Tap)

The double tap, like a single tap, also uses the `UITapGestureRecognizer`. You simply set the `numberOfTapsRequired` to 2.

**Swift**

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    // Double Tap
    let doubleTapGesture = UITapGestureRecognizer(target: self, action: #selector(handleDoubleTap))
    doubleTapGesture.numberOfTapsRequired = 2
    doubleTapView.addGestureRecognizer(doubleTapGesture)
}

// Double tap action
func handleDoubleTap() {
    label.text = "Double tap recognized"
}
```

**Notes**

- A sample project can be found [here](#).
- You could recognize a triple tap by setting the `numberOfTapsRequired` to 3.

# Section 27.3: Adding a Gesture recognizer in the Interface Builder

Drag a gesture recognizer from the object library onto your view.
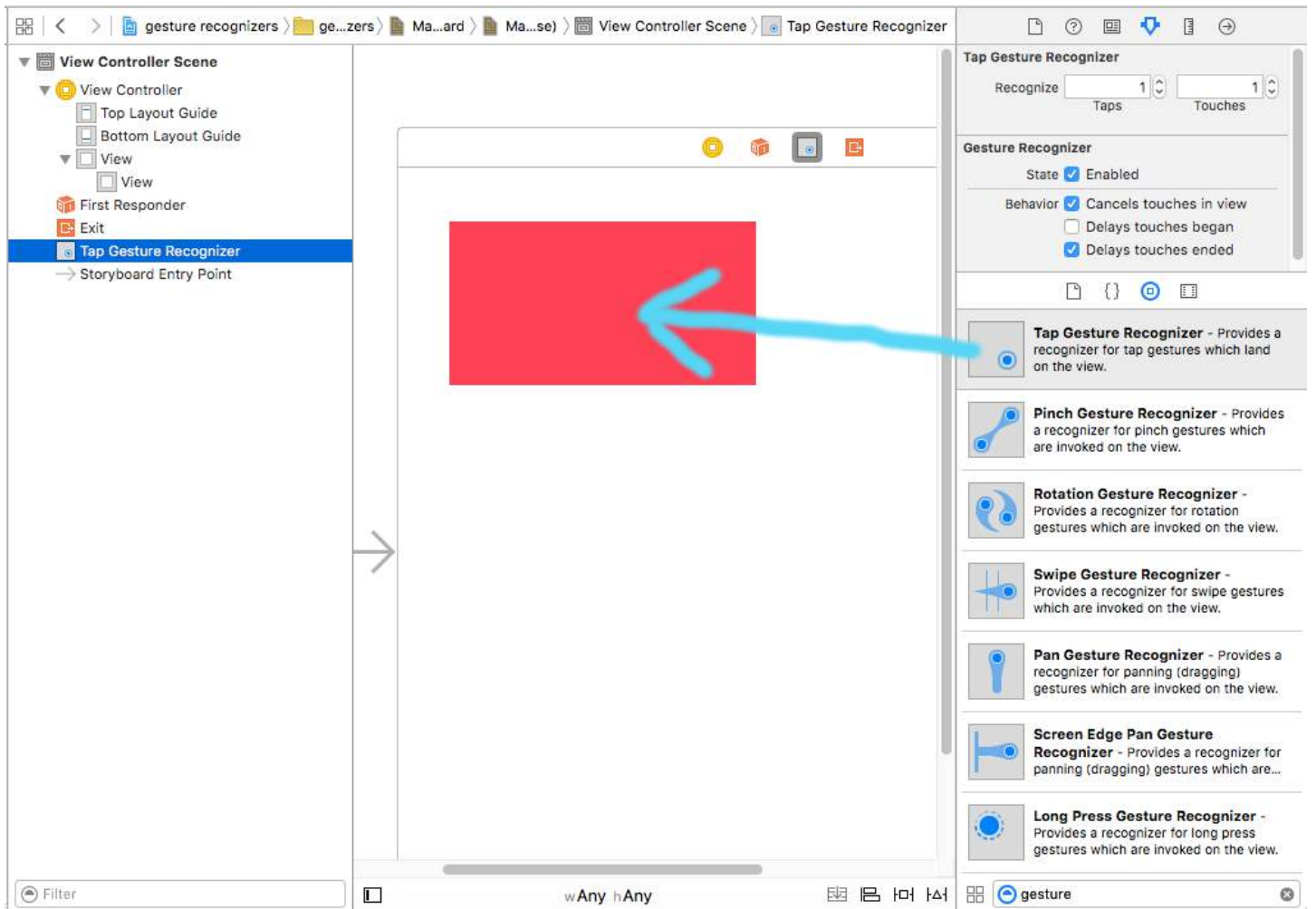
Control drag from the gesture in the Document Outline to your View Controller code in order to make an Outlet and an Action.
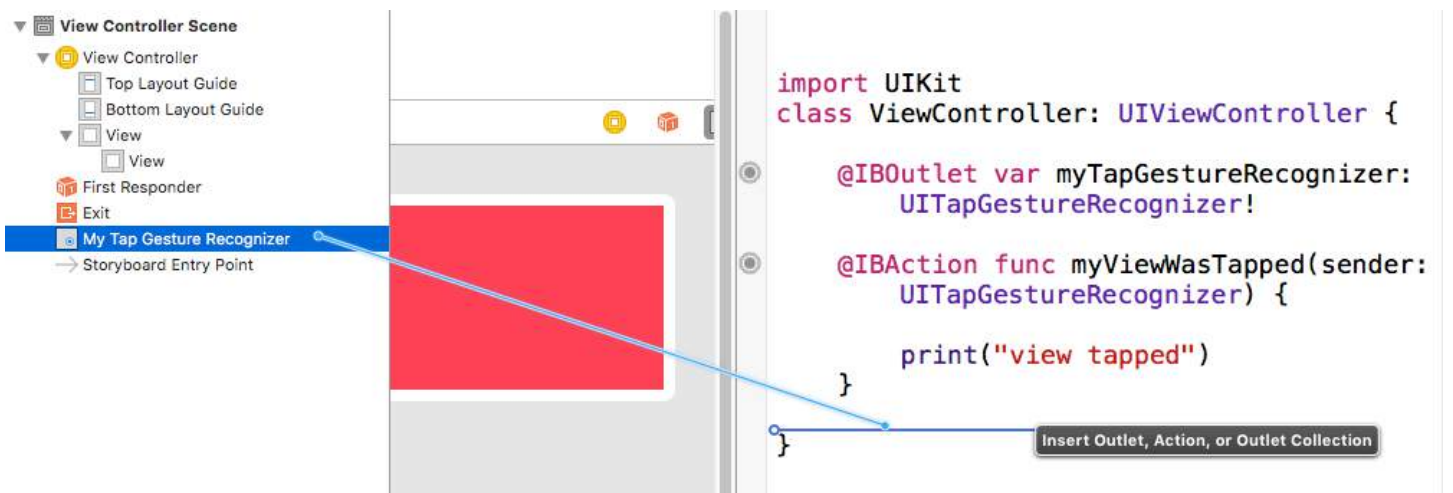


**Notes**

- This example comes from [this fuller sample project](#) demonstrating gesture recognizers.

# Section 27.4: UILongPressGestureRecognizer

The `UILongPressGestureRecognizer` lets you listen for a long press on a view. You can set the length of delay before the action method is called.

**Swift**

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Long Press
    let longPressGesture = UILongPressGestureRecognizer(target: self, action:
#selector(handleLongPress(_:)))
    longPressView.addGestureRecognizer(longPressGesture)
}

// Long press action
func handleLongPress(gesture: UILongPressGestureRecognizer) {
    if gesture.state == UIGestureRecognizerState.Began {
        label.text = "Long press recognized"
    }
}
```

**Notes**

- A fuller sample project can be found [here](#).

- Change the `minimumPressDuration` to set the length of long press.

# Section 27.5: UISwipeGestureRecognizer

Swipe gestures allow you to listen for the user moving their finger across the screen quickly in a certain direction.

**Swift**

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Swipe (right and left)
    let swipeRightGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    let swipeLeftGesture = UISwipeGestureRecognizer(target: self, action:
#selector(handleSwipe(_:)))
    swipeRightGesture.direction = UISwipeGestureRecognizerDirection.Right
    swipeLeftGesture.direction = UISwipeGestureRecognizerDirection.Left
    swipeView.addGestureRecognizer(swipeRightGesture)
    swipeView.addGestureRecognizer(swipeLeftGesture)
}

// Swipe action
func handleSwipe(gesture: UISwipeGestureRecognizer) {
    label.text = "Swipe recognized"

    // example task: animate view off screen
    let originalLocation = swipeView.center
    if gesture.direction == UISwipeGestureRecognizerDirection.Right {
        label.text = "Swipe right"
    } else if gesture.direction == UISwipeGestureRecognizerDirection.Left {
        label.text = "Swipe left"
    }
}
```

**Objective-C**

```
- (void)viewDidLoad
{
[super viewDidLoad];
```

```
UISwipeGestureRecognizer *swipeLeft = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];
UISwipeGestureRecognizer *swipeRight = [[UISwipeGestureRecognizer alloc] initWithTarget:self
action:@selector(handleSwipe:)];

// Setting the swipe direction.
[swipeLeft setDirection:UISwipeGestureRecognizerDirectionLeft];
[swipeRight setDirection:UISwipeGestureRecognizerDirectionRight];

// Adding the swipe gesture on image view
[self.view addGestureRecognizer:swipeLeft];
[self.view addGestureRecognizer:swipeRight];

}
//Handling Swipe Gesture Events

- (void)handleSwipe:(UISwipeGestureRecognizer *)swipe {

    if (swipe.direction == UISwipeGestureRecognizerDirectionLeft) {
        NSLog(@"Left Swipe");
    }

    if (swipe.direction == UISwipeGestureRecognizerDirectionRight) {
        NSLog(@"Right Swipe");
    }

}
```

**Notes**

- A fuller project example can be found here.

# Section 27.6: UIPinchGestureRecognizer

Pinches are a two fingered gesture where the fingers move closer or farther from each other. This gesture is generally used for resizing a view.

**Swift**

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Pinch
    let pinchGesture = UIPinchGestureRecognizer(target: self, action: #selector(handlePinch(_:)))
    pinchView.addGestureRecognizer(pinchGesture)
}

// Pinch action
func handlePinch(gesture: UIPinchGestureRecognizer) {
    label.text = "Pinch recognized"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeScale(gesture.scale, gesture.scale)
        pinchView.transform = transform
    }
}
```
**Notes**

- A fuller project example can be found here.

# Section 27.7: UIRotationGestureRecognizer

Two fingers rotating around a center can be listened for with the `UIRotationGestureRecognizer`. This is generally used for rotating a view.

**Swift**

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    // Rotate
    let rotateGesture = UIRotationGestureRecognizer(target: self, action:
#selector(handleRotate(_:)))
    rotateView.addGestureRecognizer(rotateGesture)
}

// Rotate action
func handleRotate(gesture: UIRotationGestureRecognizer) {
    label.text = "Rotate recognized"

    if gesture.state == UIGestureRecognizerState.Changed {
        let transform = CGAffineTransformMakeRotation(gesture.rotation)
        rotateView.transform = transform
    }
}
```

**Notes**

- A sample project can be found [here](#).

# Chapter 28: UIBarButtonItem

| Parameter | Description |
| --- | --- |
| title | The UIBarButtonItem title |
| style | The style of the UIBarButtonItem |
| target | The object to receive the UIBarButtonItem action |
| action | The selector (method) to be performed when the UIBarButtonItem is pressed |

## Section 28.1: Creating a UIBarButtonItem in the Interface Builder

The example below shows how to add a navigation bar button (called a `UIBarButtonItem`) in the Interface Builder.

**Add a Navigation Controller to your Storyboard**

Select your View Controller and then in the Xcode menu choose **Editor > Embed In > Navigation Controller**.
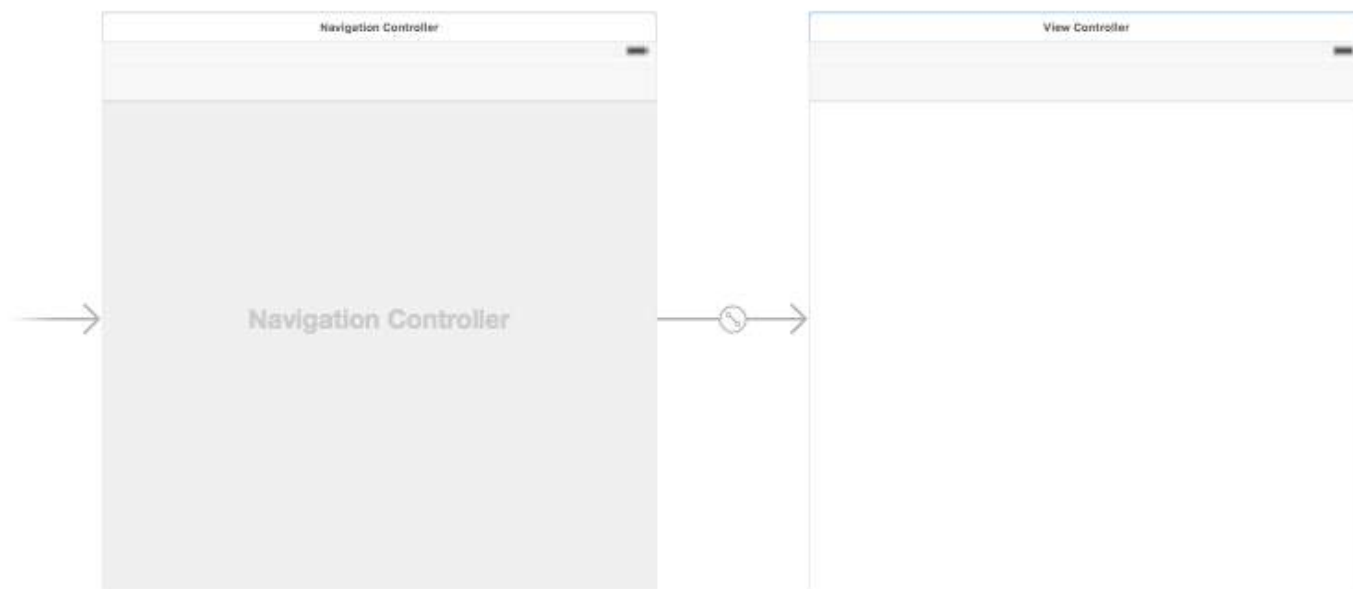


Alternatively, you could add a `UINavigationBar` from the Object Library.

**Add a Bar Button Item**

Drag a `UIBarButtonItem` from the Object Library to the top navigation bar.
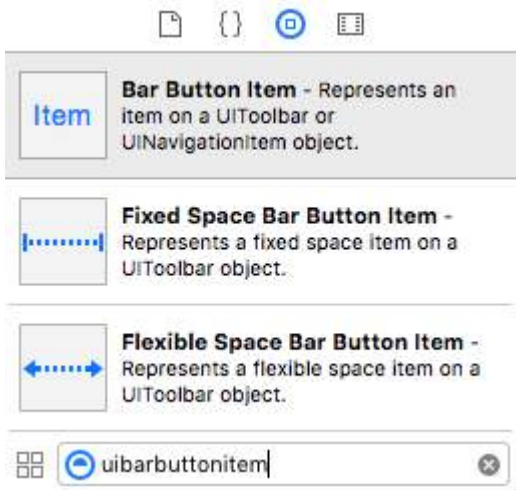
It should look like this:



**Set the Attributes**

You could double-click "Item" to change the text to something like "Refresh", but there is an actual icon for *Refresh* that you can use. Just select the Attributes Inspector for the `UIBarButtonItem` and for **System Item** choose **Refresh**.
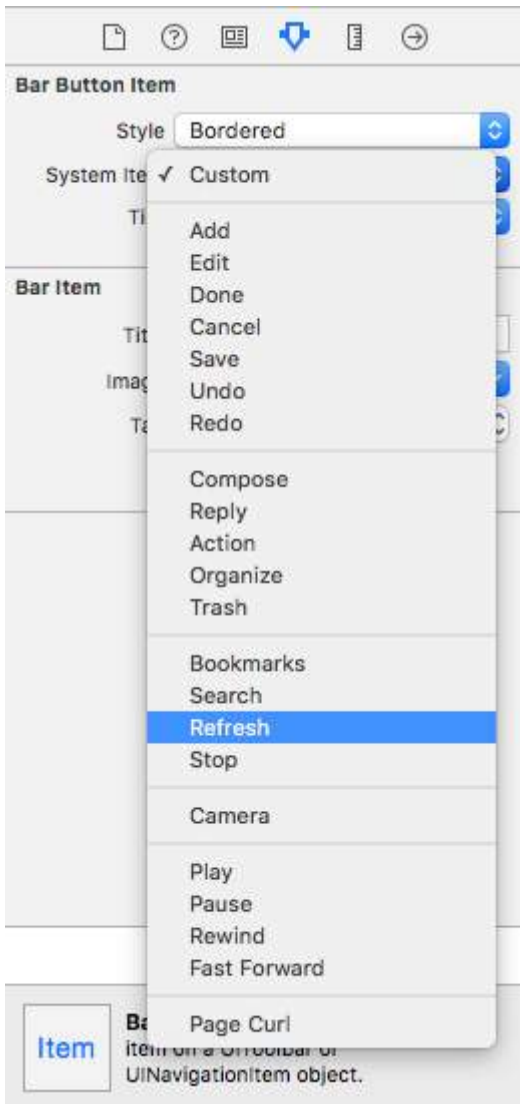
That will give you the default Refresh icon.



**Add an IB Action**

Control drag from the `UIBarButtonItem` to the View Controller to add an `@IBAction`.

```
class ViewController: UIViewController {

    @IBAction func refreshBarButtonItemTap(sender: UIBarButtonItem) {

        print("How refreshing!")
    }

}
```

That's it.

**Notes**

- This example originally comes from [this Stack Overflow answer](.).

# Section 28.2: Creating a UIBarButtonItem

```
//Swift
let barButtonItem = UIBarButtonItem(title: "Greetings!", style: .Plain, target: self, action:
#selector(barButtonTapped))
self.navigationItem.rightBarButtonItem = barButtonItem

//Objective-C
UIBarButtonItem *barButtonItem = [[UIBarButtonItem alloc] initWithTitle:@"Greetings!"
style:UIBarButtonItemStylePlain target:self action:@selector(barButtonTaped)];
self.navigationItem.rightBarButtonItem = barButtonItem;
```
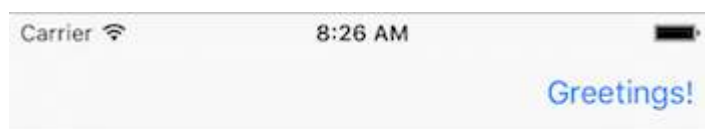


# Section 28.3: Bar Button Item Original Image with no Tint Color

Provided that `barButtonItem` has a non-null image property (e.g. set in the Interface Builder).

**Objective-C**

```
    barButtonItem.image = [barButtonItem.image
imageWithRenderingMode:UIImageRenderingModeAlwaysOriginal];
```

# Chapter 29: UIScrollView

## Section 29.1: Scrolling content with Auto Layout enabled

This project is a self-contained example done completely in the Interface Builder. You should be able to work through it in 10 minutes or less. Then you can apply the concepts you learned to your own project.
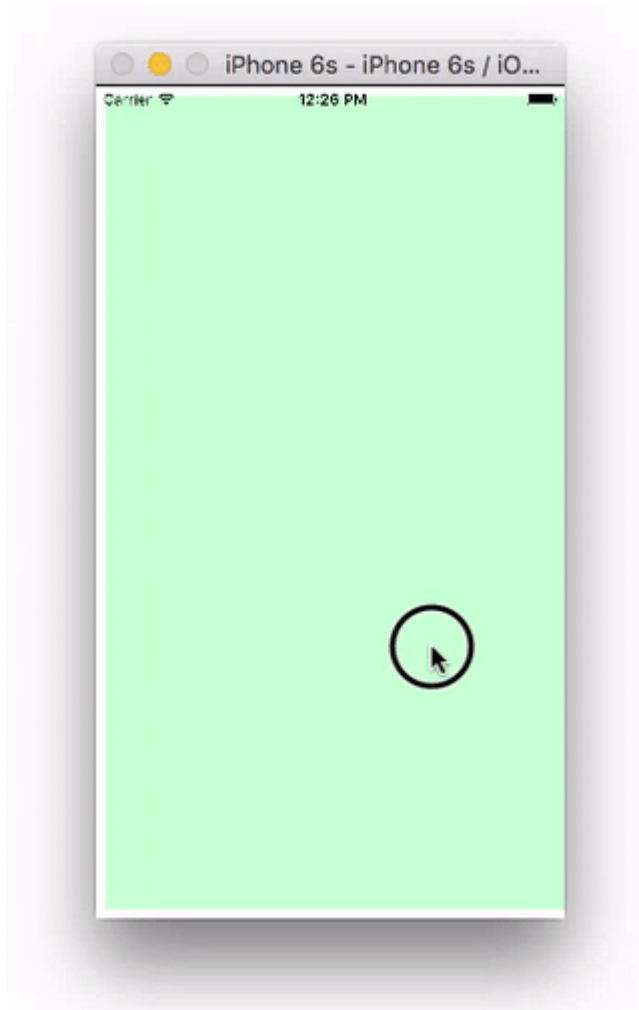


Here I just use `UIView`s but they can represent whatever view you like (ie, button, label, etc). I also chose horizontal scrolling because the storyboard screenshots are more compact for this format. The principles are the same for vertical scrolling, though.

**Key concepts**

- The `UIScrollView` should only use one subview. This is a 'UIView' that serves as the content view to hold everything you wish to scroll.
- Make the content view and the scroll view's *parent* have equal heights for horizontal scrolling. (Equal widths for vertical scrolling)
- Make sure that all of the scrollable content has a set width and is pinned on all sides.

**Start a new project**

It can be just a single view application.

**Storyboard**

In this example, we will make a horizontal scroll view. Select the View Controller and then choose Freeform in the

Size Inspector. Make the width `1,000` and the height `300`. This just gives us the room on the storyboard to add content that will scroll.



## Add a Scroll View

Add a `UIScrollView` and pin all four sides to the root view of the view controller.
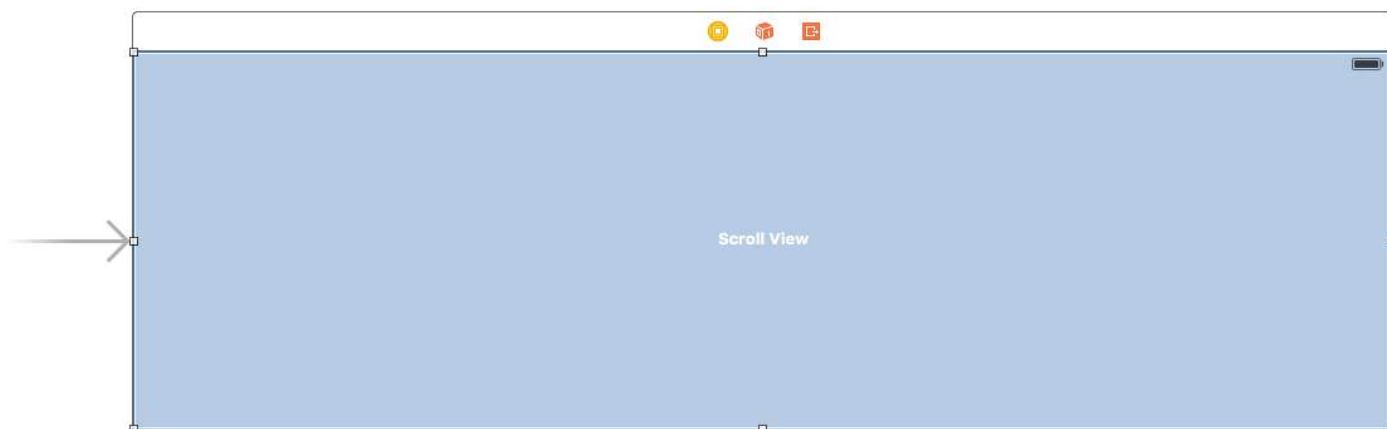


## Add a Content View

Add a `UIView` as a subview to the scroll view. *This is key.* Don't try to add lots of subviews to the scroll view. Just add a single `UIView`. This will be your content view for the other views you want to scroll. Pin the content view to the scroll view on all four sides.
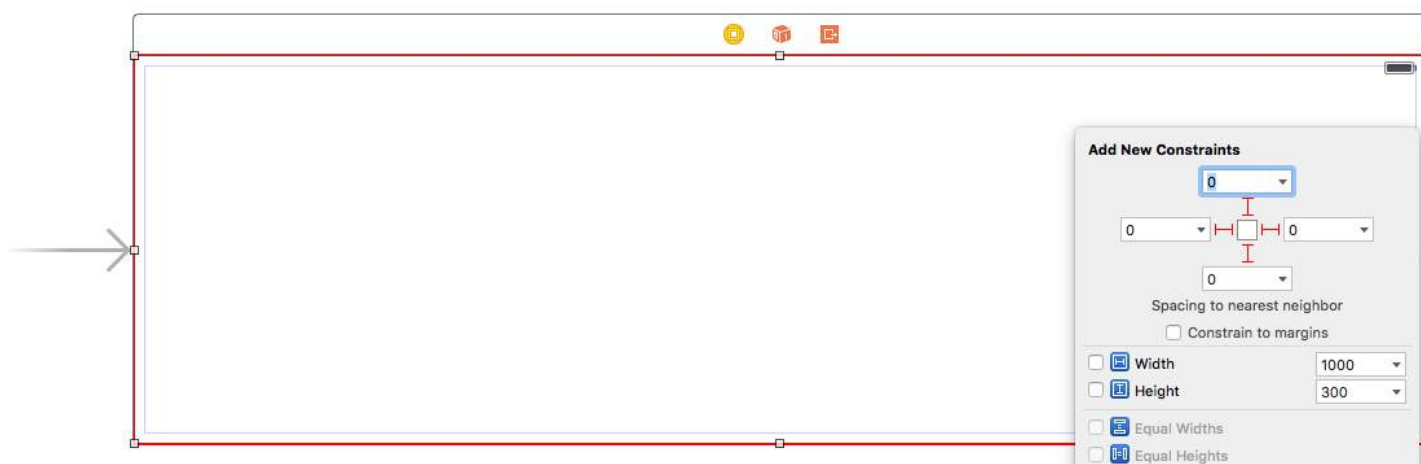


## Equal Heights

Now in the Document Outline, Command click both the content view and the scroll view's *parent view* in order to select them both. Then set the heights to be equal (Control</kbd drag from the Content View to the Scroll View>). *This is also key.* Because we are scrolling horizontally, the scroll view's content view won't know how high it should be unless we set it in this way.
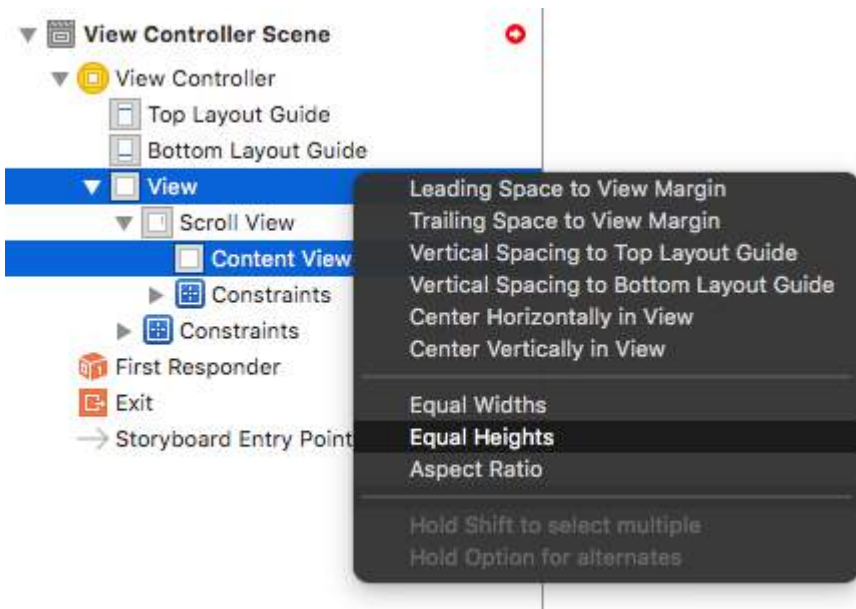
Note:

- If we were making the content scroll vertically, then we would set the content view's width to be equal to the scroll view's parent's width.

**Add content**

Add three `UIView`s and give them all constraints. I used 8 point margins for everything.
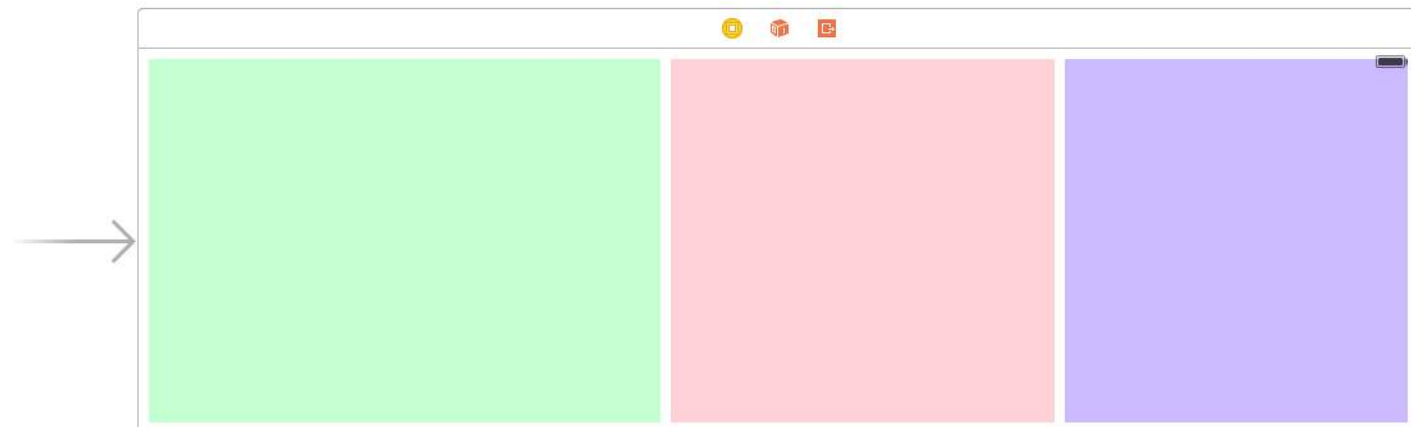


Constraints:

- Green view: pin the top, left, and bottom edges. Make the width 400.
- Red view: pin the top, left, and bottom edges. Make the width 300.
- Purple view: pin all four edges. Make the width whatever the remaining space is (268 in this case).

*Setting the width constraints is also key* so that the scroll view knows how wide its content view will be.

**Finished**

That's all. You can run your project now. It should behave like the scrolling image at the top of this answer.

**Further Study**

- iOS: How To Make AutoLayout Work On A ScrollView
- How to configure a UIScrollView with Auto Layout in Interface Builder

- YouTube video tutorial: [UIScrollView - How to keep your views on screen](#)

# Section 29.2: Create a UIScrollView

Create an instance of `UIScrollView` with a `CGRect` as frame.

**Swift**

```swift
let scrollview = UIScrollView.init(frame: CGRect(x: 0, y: 0, width: 320, height: 400))
```

**Objective-C**

```objc
UIScrollView *scrollview = [[UIScrollView alloc] initWithFrame:CGRectMake(0, 0, 320, 400)];
```

# Section 29.3: ScrollView with AutoLayout

Simple steps to use scrollview with autolayout.

- Create a new project with single view application
- Select the default viewcontroller and change its screen size to iPhone-4inch from attributes inspector.
- Add a scrollview to your viewcontroller's view as follows and set background color to blue
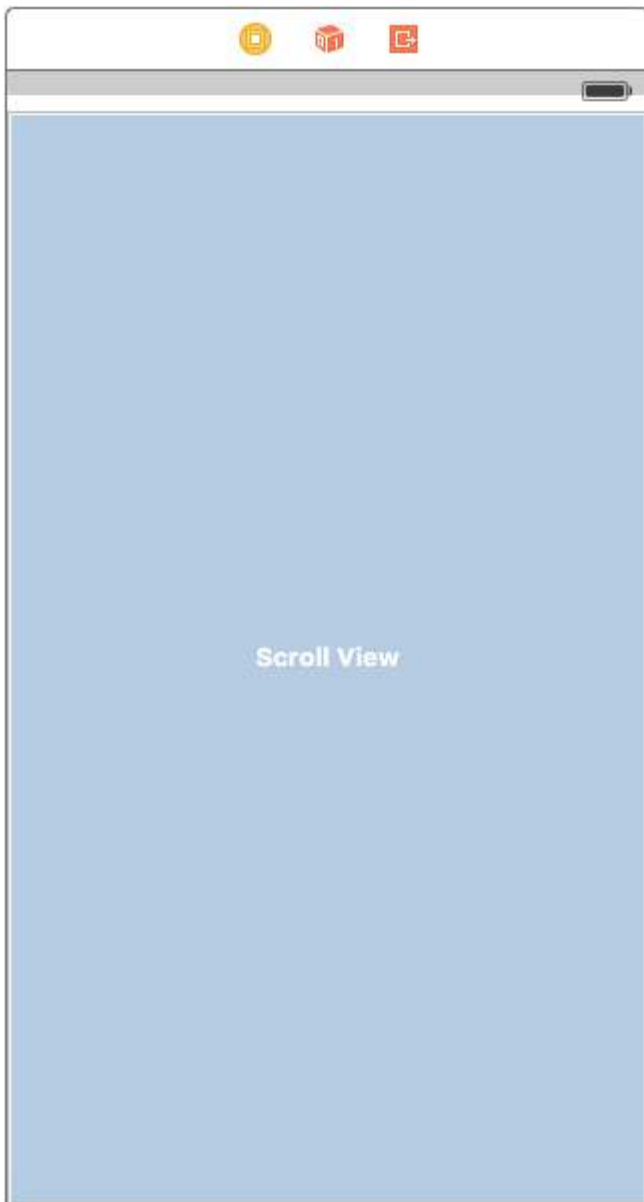
- Add constraints on it as shown in below image
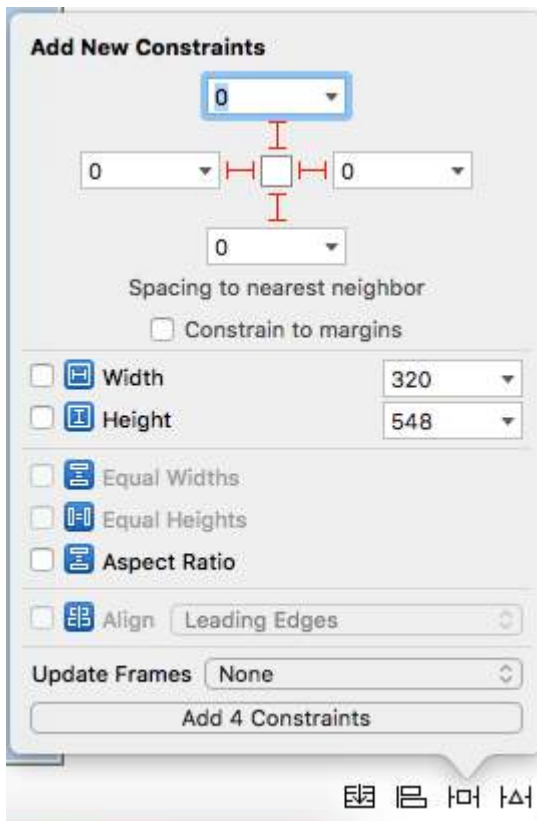


What this will do is,simply stick every edge of scrollview to viewcontroller's view

**Scenario 1:**

***Now lets say our content is huge,and we want it to scroll horizontally as well as vertically.***

For this,

- Add a UIView to the scrollview of frame(0,0,700,700).Lets give it orange background color to identify it differently.
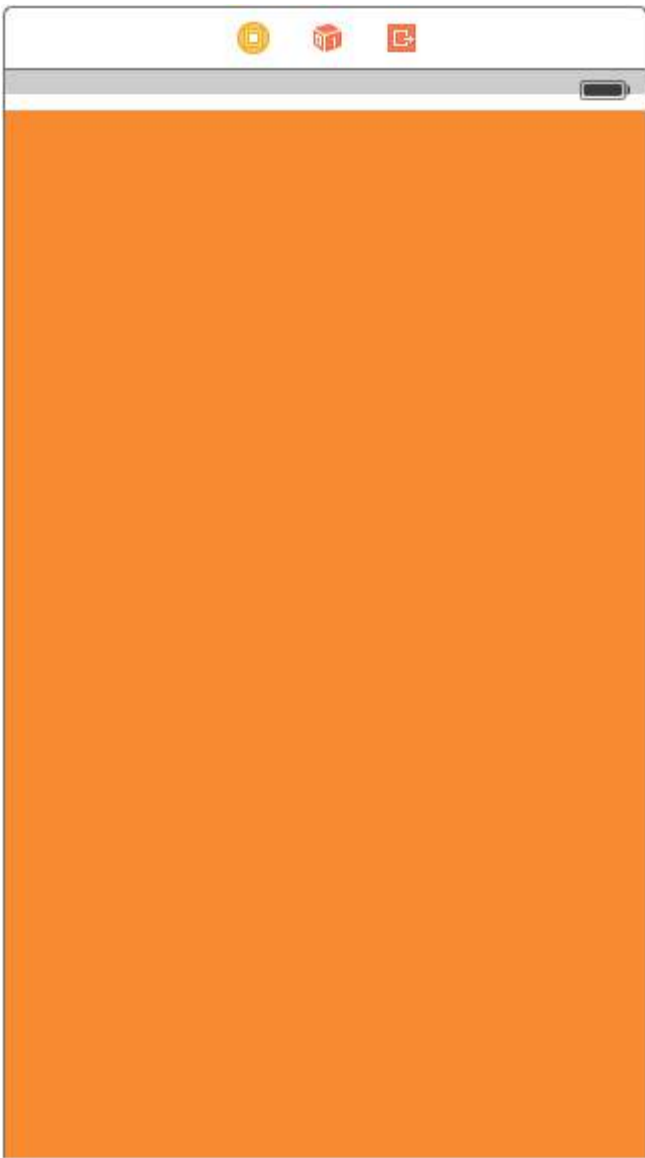
Next comes the important part,we need it to scroll horizontally and vertically.

- Select the orange view and add the following constraints
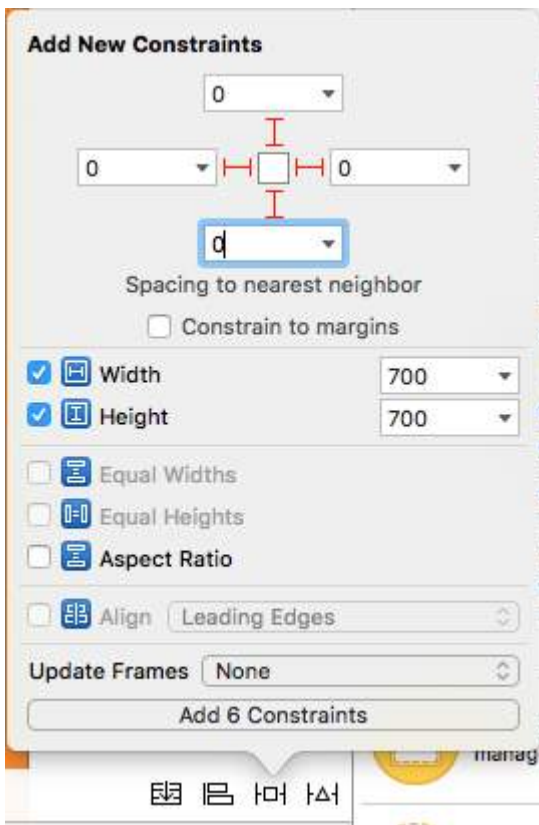
Let me explain what we did in above step.

- We fixed the height and width to 700.
- We set trailing space to scrollview = 0 which tells the scrollview that content is horizontally scrollable.
- We set bottom space to scrollview = 0 which tells the scrollview that content is vertically scrollable.

Now run the project and check.

**Scenario 2:** *Lets consider a scenario where we know that content width is going to be same as scroll width width,but height is larger than scrollview.*

Follow the steps to scroll content vertically.

- Delete the width constraint in above case.
- Change the width of orange view to match to scrollview width.
- Ctrl-drag from orange view to scroll view and add **equal widths** constraint.
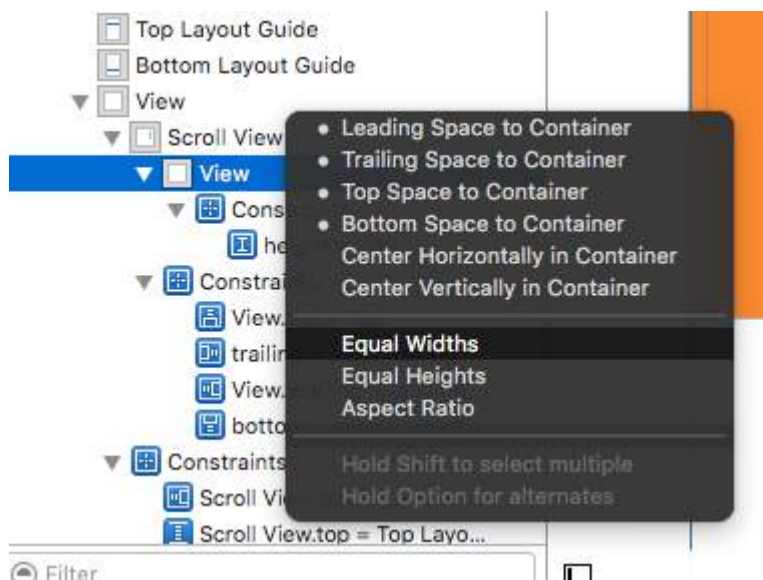
- And Done!!! Simply run and check if it scrolls vertically

**Scenario 3:**

***Now we want to scroll only horizontally and not vertically.***

Follow the steps to horizontally scroll the content.

- Undo all the changes to achieve constraints as below(i.e **restore original constraints which achieved vertical and horizontal scroll**)
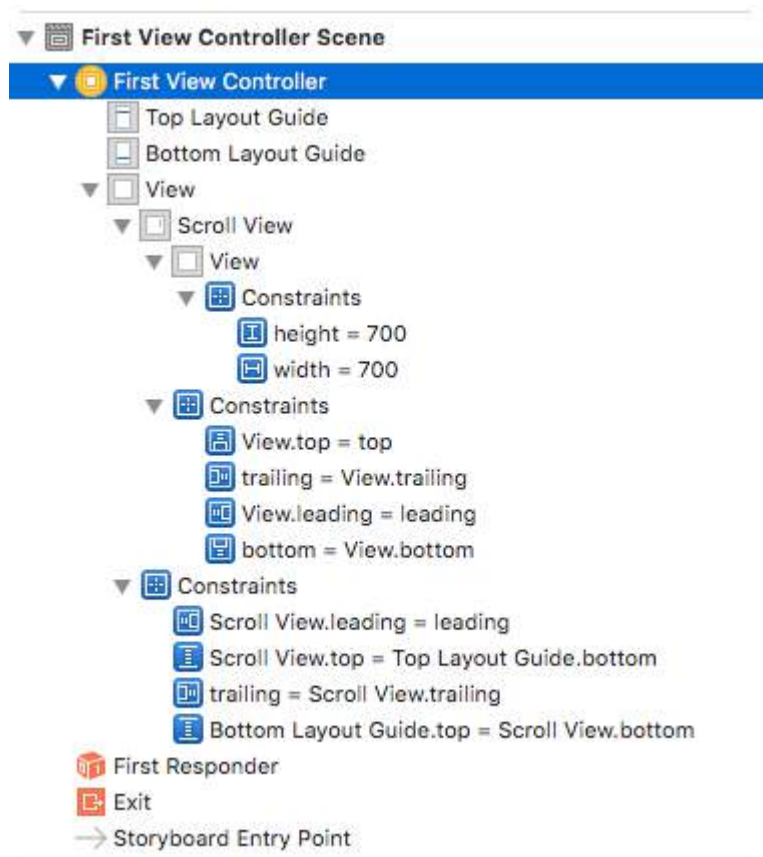


- Check frame of orange view,which should be (0,0,700,700)
- Delete height constraint of orange view.
- Change the height of orange view to match the scrollview height.
- Ctrl-drag from orange view to scroll view and add **equal heights** constraint.
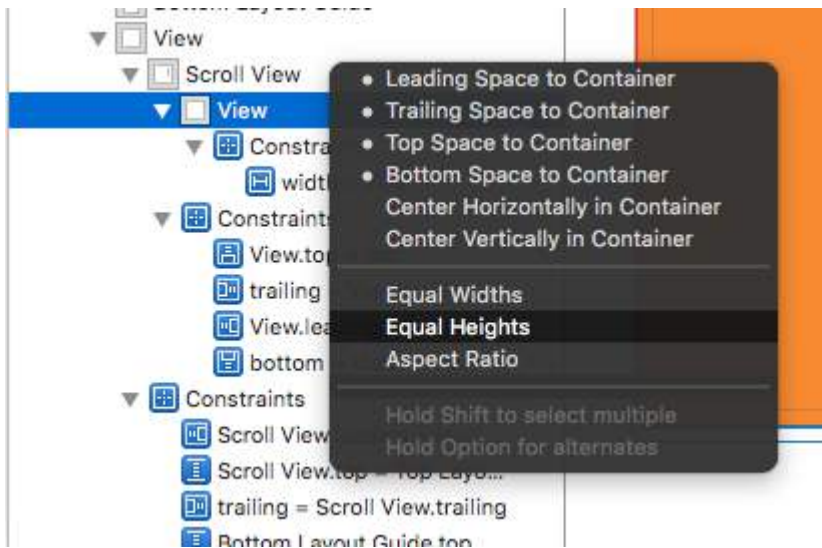
- And Done!!! Simply run and check if it scrolls vertically

## Section 29.4: Detecting when UIScrollView finished scrolling with delegate methods

**scrollViewDidEndDecelerating:** this tells the delegate that the scroll view has ended decelerating the scrolling movement.

**Objective C:**

```objc
- (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
    [self stoppedScrolling];
}

- (void)scrollViewDidEndDragging:(UIScrollView *)scrollView willDecelerate:(BOOL)decelerate {
    if (!decelerate) {
        [self stoppedScrolling];
    }
}

- (void)stoppedScrolling {
    // done, do whatever
}
```

**Swift:**

```swift
func scrollViewDidEndDragging(scrollView: UIScrollView, willDecelerate decelerate: Bool) {
    if !decelerate {
        stoppedScrolling()
    }
}

func scrollViewDidEndDecelerating(scrollView: UIScrollView) {
    stoppedScrolling()
}

func stoppedScrolling() {
    // done, do whatever
}
```

## Section 29.5: Enable/Disable Scrolling

Property `scrollEnabled` stores a `Boolean` value that determines whether scrolling is enabled or not.
If the value of this property is true/YES, scrolling is enabled, otherwise not. The default value is `true`

**Swift**

```
scrollview.isScrollEnabled = true
```

**Objective-C**

```
scrollview.scrollEnabled = YES;
```

# Section 29.6: Zoom In/Out UIImageView

**Create UIScrollView instance**

```
let scrollview = UIScrollView.init(frame: self.view.bounds)
```

And then set these properties:

```
scrollView.minimumZoomScale = 0.1
scrollView.maximumZoomScale = 4.0
scrollView.zoomScale = 1.0
scrollview.delegate = self as? UIScrollViewDelegate
```

To zoom in and out image we must specify the amount the user can zoom in and out. We do this by setting values of the scroll view's `minimumZoomScale` and `maximumZoomScale` properties. Both of these are set to 1.0 by default.

And `zoomScale` to 1.0 which specify the zoom factor for the minimum and maximum zooming.

To support zooming, we must set a delegate for your scroll view. The delegate object must conform to the `UIScrollViewDelegate` protocol. That delegate class must implement the `viewForZoomingInScrollView()` method and return the view to zoom.

Modify your ViewController as shown

```
class ViewController: UIViewController, UIScrollViewDelegate
```

Then add the following delegate function to the class.

```
func viewForZoomingInScrollView(scrollView: UIScrollView) -> UIView? {
    return imageView
}
```
**Now create UIImageView instance**

Make this variable as class variable

```
var imageView:UIImageView = UIImageView.init(image: UIImage.init(named: "someImage.jpg"))
```

And then add it to scrollview

```
scrollView?.addSubview(imageView)
```

Reference

- Scroll View Programming Guide for iOS
- UIScrollView Tutorial

# Section 29.7: Scroll View Content Size

The `contentSize` property must be set to the size of the scrollable content. This specifies the size of the scrollable area. Scrolling is visible when scrollable area i.e. `contentSize` is larger than the `UIScrollView` frame size.

**With Autolayout:**

When the scroll view's content is set up using autolayout, it must be explicitly sized both vertically and horizontally and have all 4 edges pinned to the containing scroll view. That way, the `contentSize` is calculated automatically based on the scroll view's contents and also gets updated when the content's layout is changed.

**Manually:**

**Swift**

```
scrollview.contentSize = CGSize(width: 640, height: 800)
```

**Objective-C**

```
scrollview.contentSize = CGSizeMake(640, 800);
```

# Section 29.8: Restrict scrolling direction

You can restrict the directions the user is able to scroll to using the following code:

```
func scrollViewDidScroll(_ scrollView: UIScrollView) {
    if scrollView.contentOffset.x != 0 {
        scrollView.contentOffset.x = 0
    }
}
```

Every time the user scrolls on the x-axis, the scrollView's content offset is set back to 0.
You can obviously change the `xs` to `ys` and therefor lock the direction to be horizontal-only.

You also need to make sure you put this code into the `scrollViewDidScroll(_ scrollView: UIScrollView)` delegate method. Otherwise, you won't get it to work.

Also, be sure to have imported the `UIScrollViewDelegate` in your class declaration, like so:

```
class ViewController: UIViewController, UIScrollViewDelegate
```

...and set the scrollView's delegate to self in some method like `viewDidLoad(_:)`

```
scrollView.delegate = self
```

# Chapter 30: UIStackView

## Section 30.1: Center Buttons with UIStackview

**Step 1:** take 4 button in your Storyboard. Button1 , Button2 , Button 3 , Button4

**Step 2:** Give Fixed Height and width to All buttons .
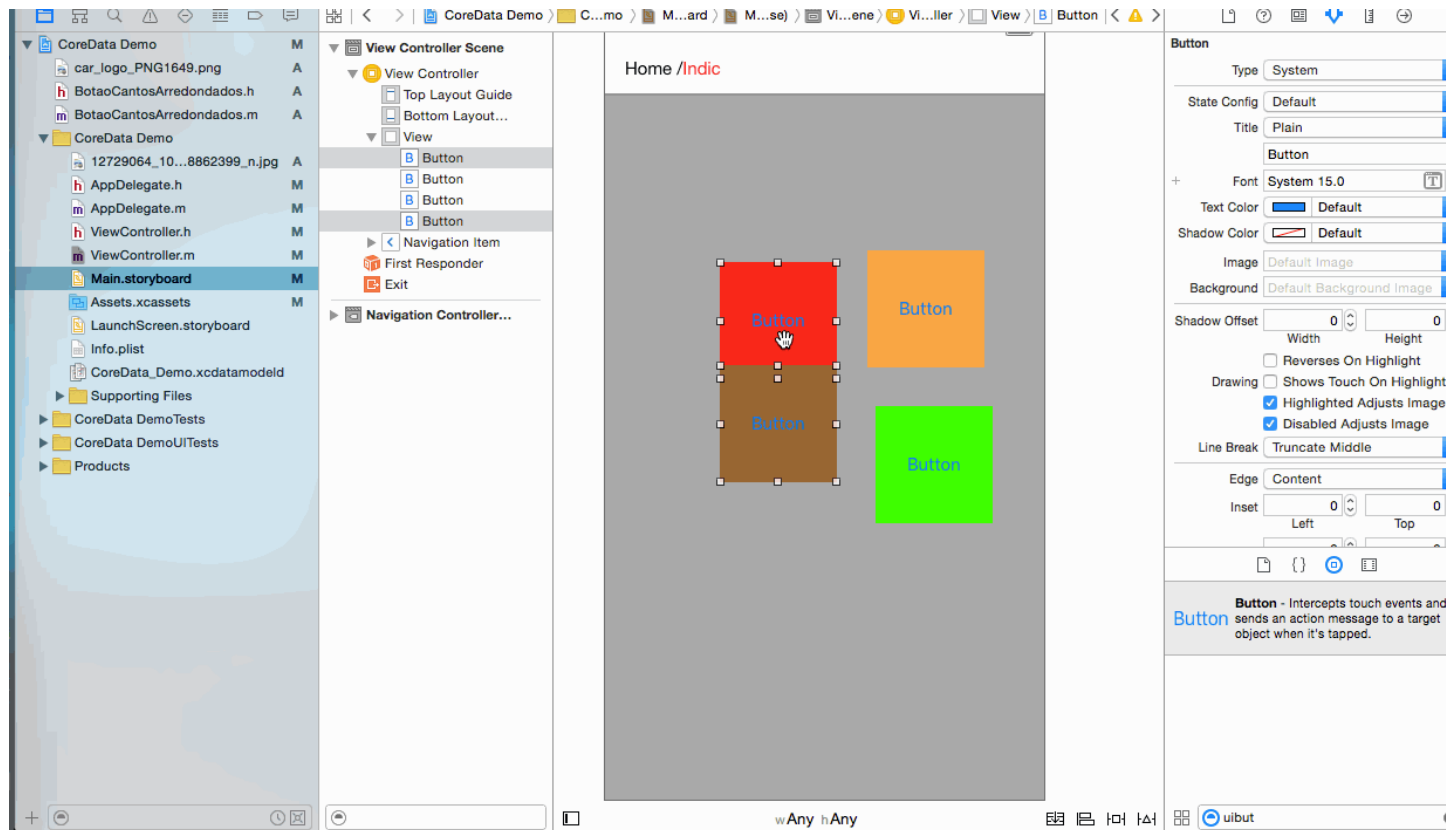


**Step 3:** All 2 - 2 button's pair in 2 stackview.
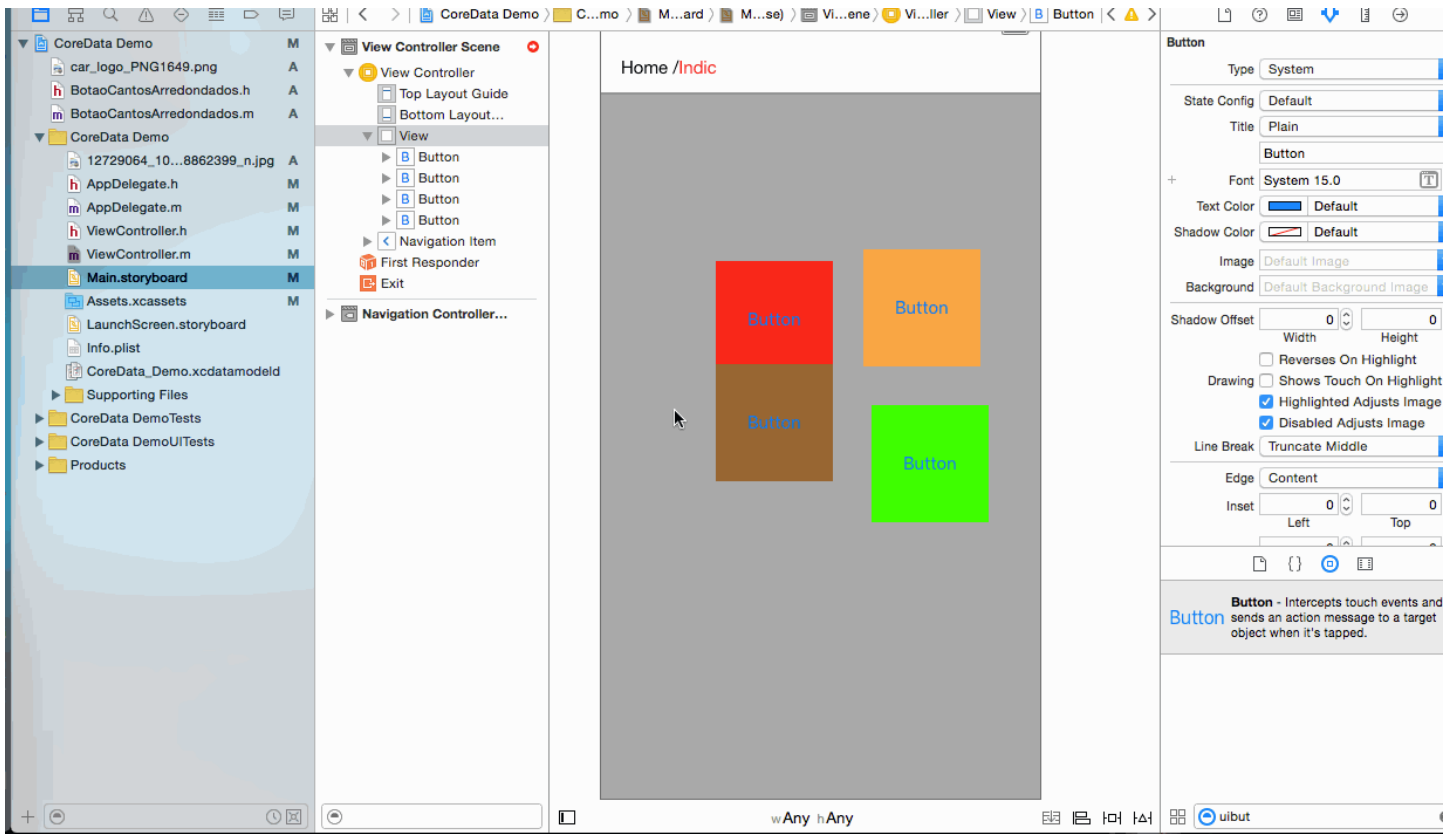
**Step 4:** Set UIStackview Property for both .

```
Distribution -> Fill Equally
        Spacing -> 5 (as per your requirement)
```
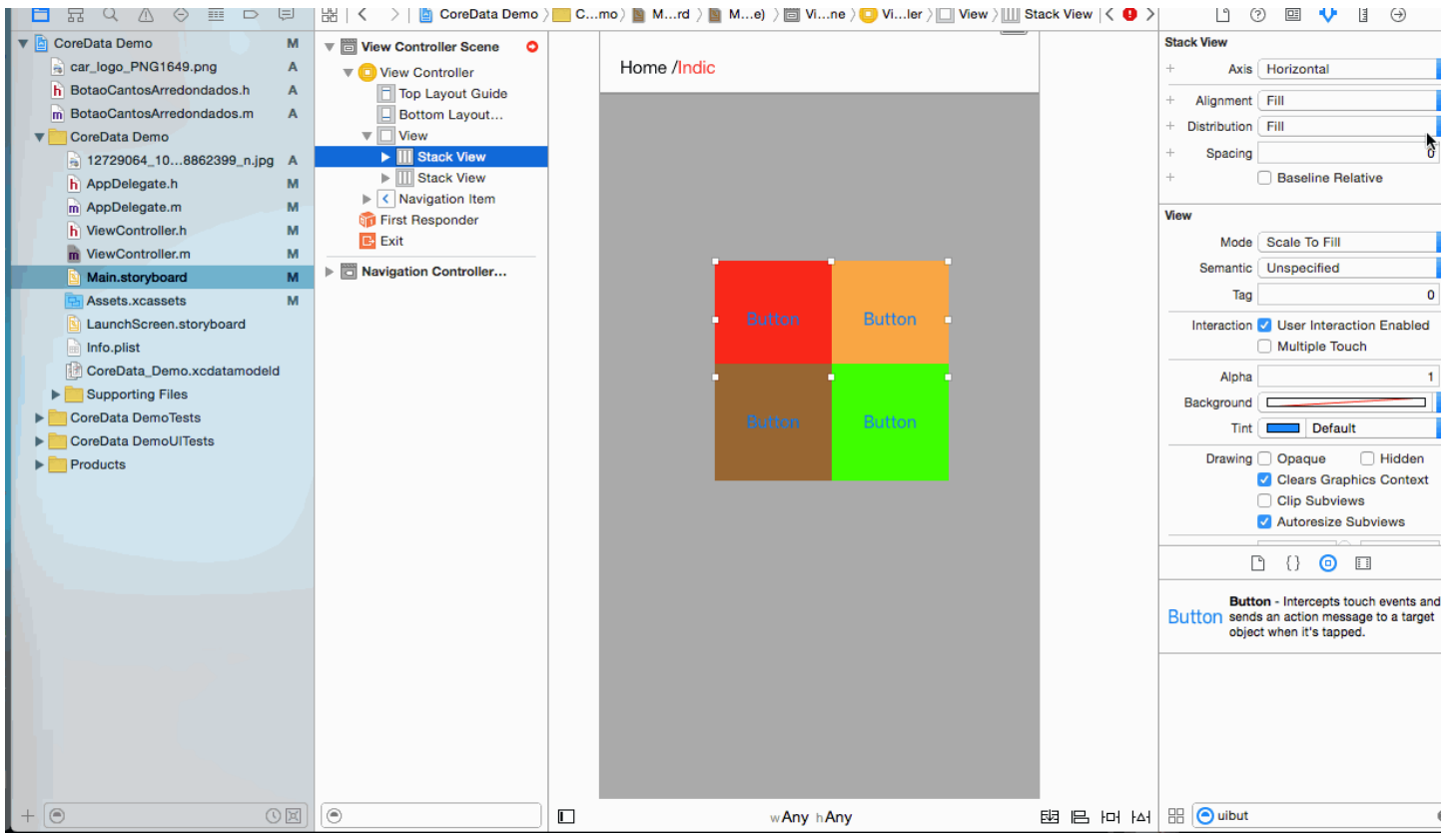
**Step 5:** Add both Stackview in one Stackview
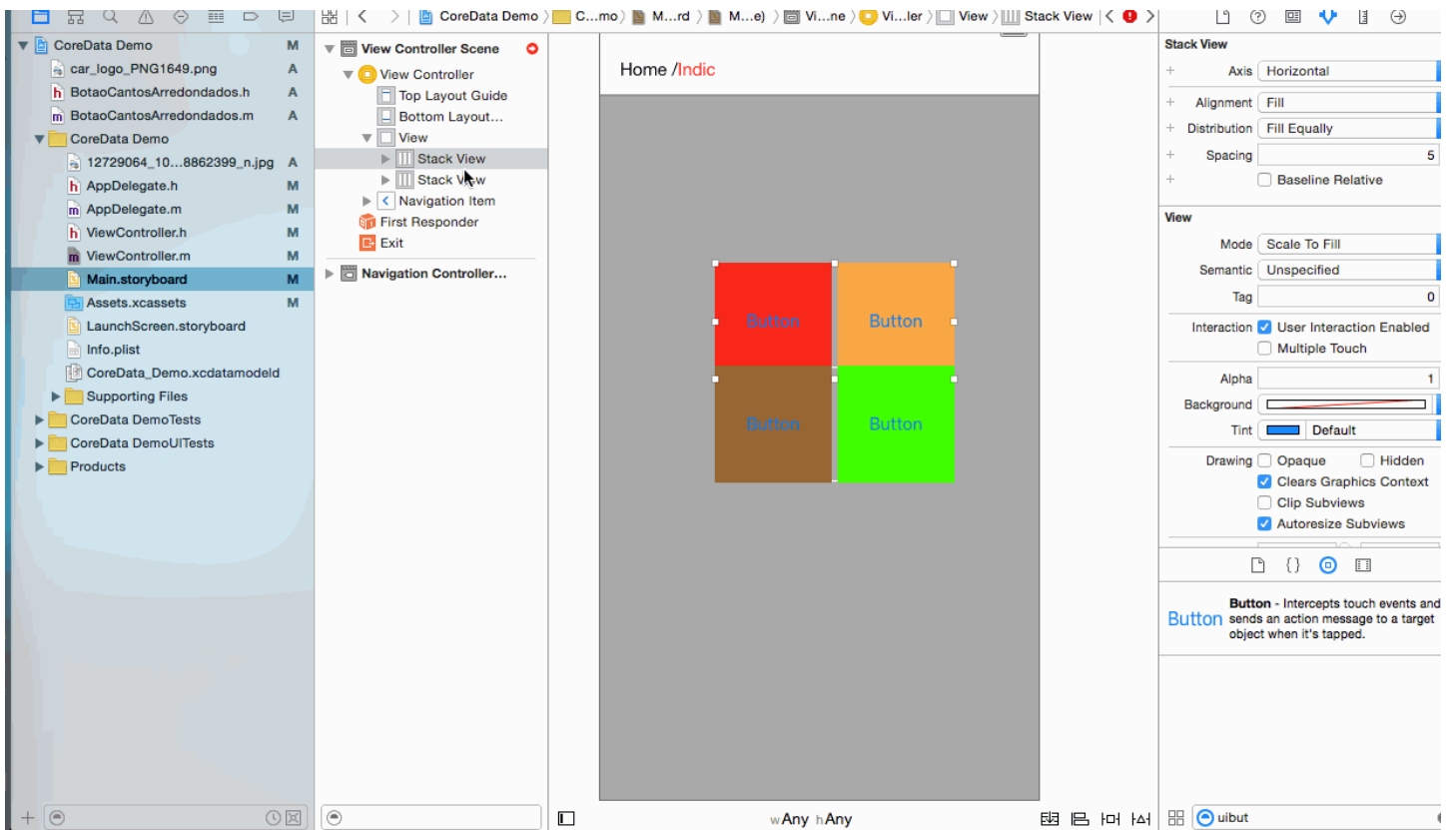


**Step 6:** Set `Distribution = Fill equally Spacing =5` in main stackview (set According to your requirement)
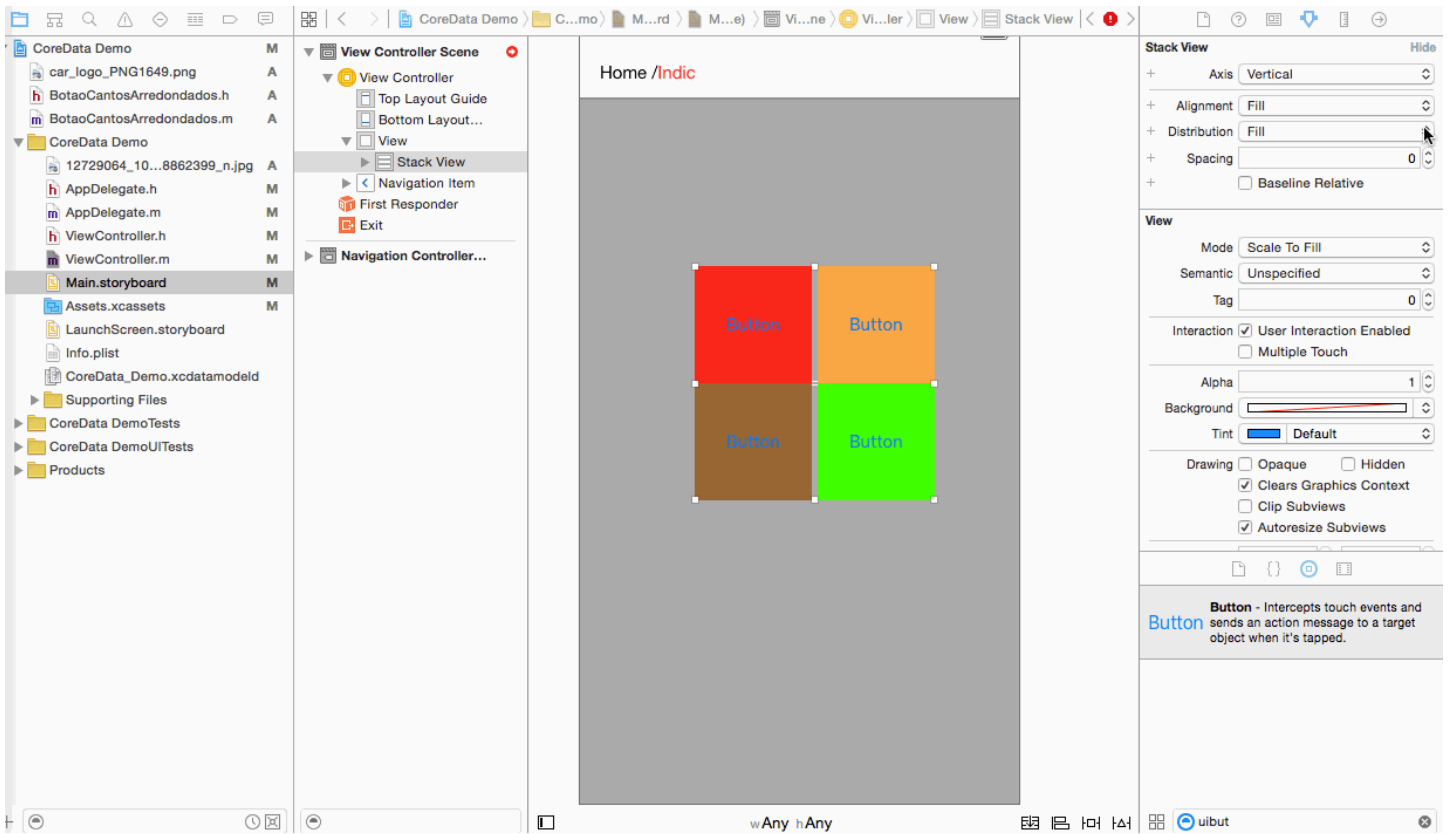
**Step 7:** Now set Constrain to main stackview

```
center Horizontally in container

center vertically in container

  and select Update Frame.
```
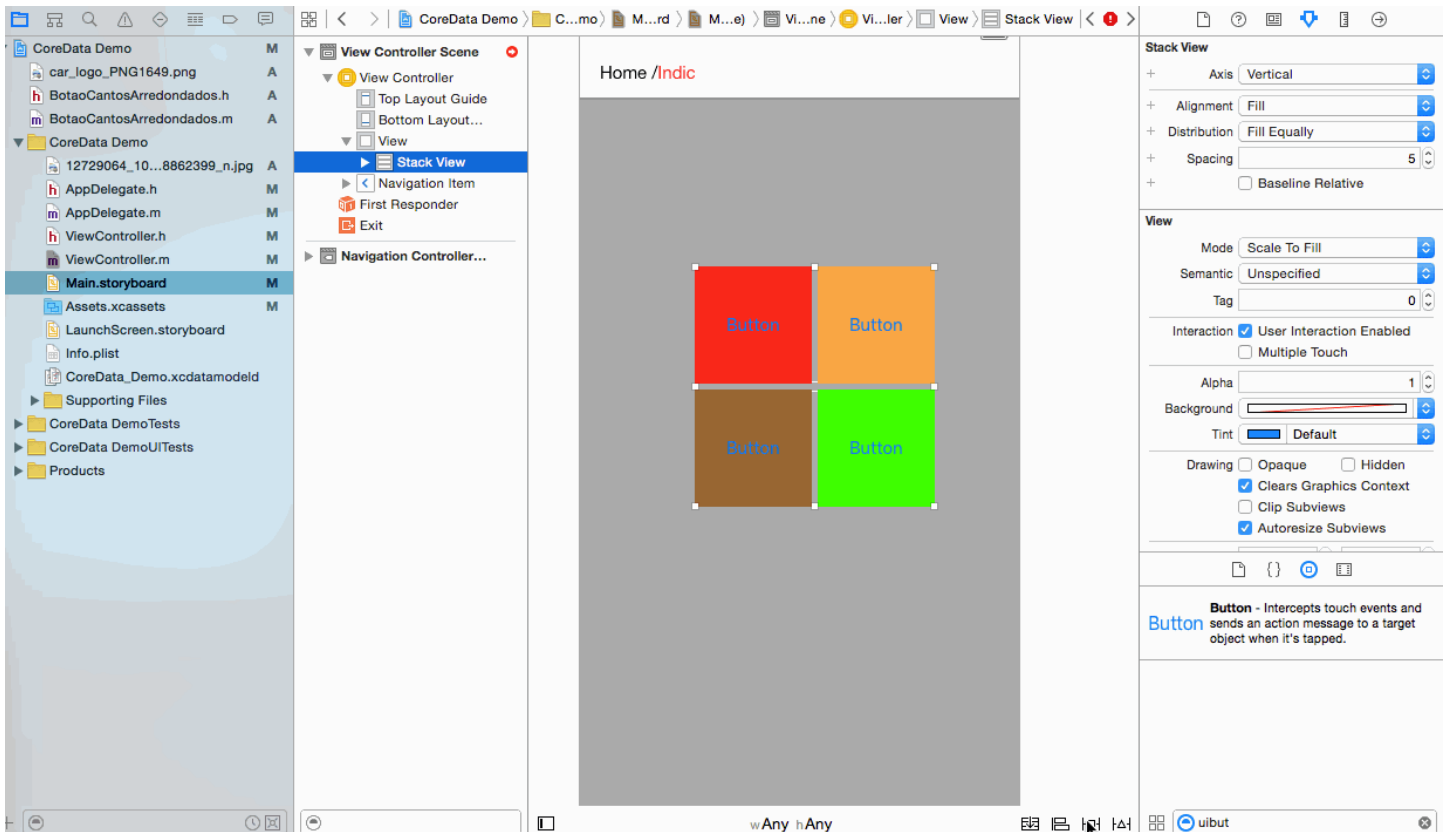


**Step 8:** It's time for Output for All device .

iPhone 4-inch

iPhone 5.5-inch

iPhone 4.7-inch

iPhone 3.5-inch
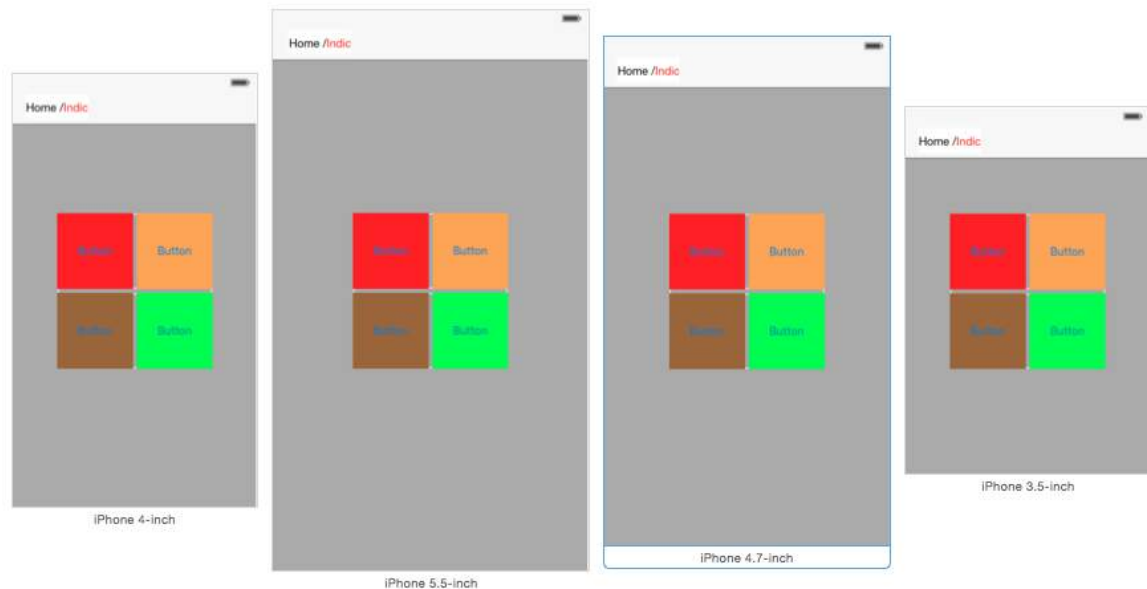
# Section 30.2: Create a horizontal stack view programmatically

**Swift 3**

```swift
let stackView = UIStackView()
stackView.axis = .horizontal
stackView.alignment = .fill // .leading .firstBaseline .center .trailing .lastBaseline
stackView.distribution = .fill // .fillEqually .fillProportionally .equalSpacing .equalCentering

let label = UILabel()
label.text = "Text"
stackView.addArrangedSubview(label)
// for horizontal stack view, you might want to add width constraint to label or whatever view
you're adding.
```

**Swift**

```swift
let stackView = UIStackView()
stackView.axis = .Horizontal
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering

let label = UILabel(frame: CGRectZero)
label.text = "Label"
stackView.addArrangedSubview(label)
// for horizontal stack view, you might want to add width constraint to label or whatever view
you're adding.
```

**Objective-C**

```objc
UIStackView *stackView = [[UIStackView alloc] init];
stackView.axis = UILayoutConstraintAxisHorizontal;
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,
UIStackViewAlignmentLastBaseline
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,
```

```
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,
UIStackViewDistributionEqualCentering

UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
label.text = @"Label";
[stackView addArrangedSubview:label];
//For horizontal stack view, you might want to add a width constraint to your label or whatever
view you are adding.
```

# Section 30.3: Create a vertical stack view programmatically

**Swift**

```
let stackView = UIStackView()
stackView.axis = .Vertical
stackView.alignment = .Fill // .Leading .FirstBaseline .Center .Trailing .LastBaseline
stackView.distribution = .Fill // .FillEqually .FillProportionally .EqualSpacing .EqualCentering

let label = UILabel(frame: CGRectZero)
label.text = "Label"
stackView.addArrangedSubview(label)
// for vertical stack view, you might want to add height constraint to label or whatever view
you're adding.
```

**Objective-C**

```
UIStackView *stackView = [[UIStackView alloc] init];
stackView.axis = UILayoutConstraintAxisVertical;
stackView.alignment = UIStackViewAlignmentFill; //UIStackViewAlignmentLeading,
UIStackViewAlignmentFirstBaseline, UIStackViewAlignmentCenter, UIStackViewAlignmentTrailing,
UIStackViewAlignmentLastBaseline
stackView.distribution = UIStackViewDistributionFill; //UIStackViewDistributionFillEqually,
UIStackViewDistributionFillProportionally, UIStackViewDistributionEqualSpacing,
UIStackViewDistributionEqualCentering

UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
label.text = @"Label";
[stackView addArrangedSubview:label];
//For vertical stack view, you might want to add a height constraint to your label or whatever view
you are adding.
```

# Chapter 31: Dynamically updating a UIStackView

## Section 31.1: Connect the UISwitch to an action we can animate switching between a horizontal or vertical layout of the image views

```
@IBAction func axisChange(sender: UISwitch) {
    UIView.animateWithDuration(1.0) {
        self.updateConstraintsForAxis()
    }
}
```

The updateConstraintForAxis function just sets the axis of the stack view containing the two image views:

```
private func updateConstraintsForAxis() {
    if (axisSwitch.on) {
        stackView.axis = .Horizontal
    } else {
        stackView.axis = .Vertical
    }
}
```

The animated gif below gives you an idea of how this appears:
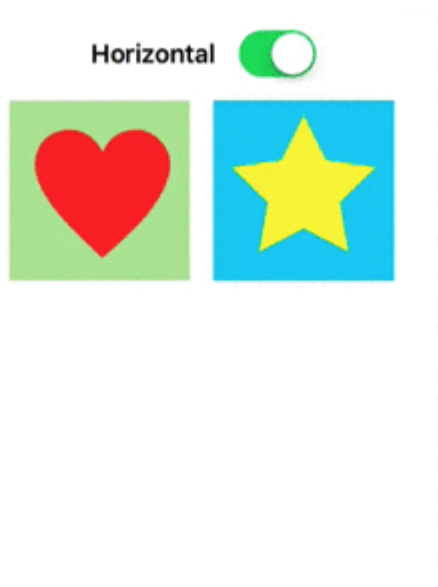
# Chapter 32: UIScrollView with StackView child

## Section 32.1: A complex StackView inside Scrollview Example

Here follows a example of what can be done with nested StackViews, giving the user the impression of a continuous scrolling experience using complex user interface elements or alignments.
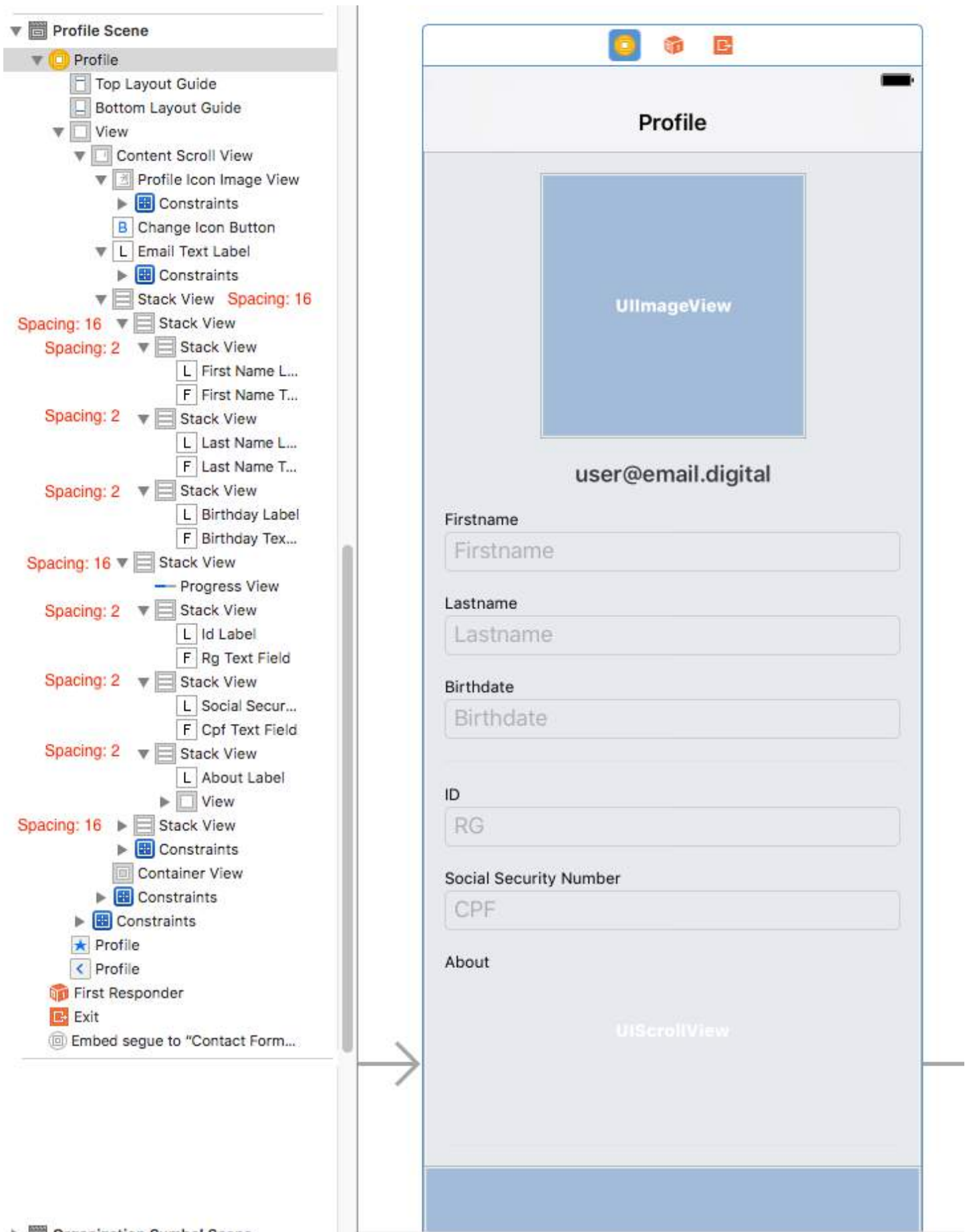
## Section 32.2: Preventing ambiguous layout

A frequent question about StackViews inside Scrollviews comes from ambiguous with/heigh alerts on interface builder. As this answer explained, it is necessary to:

1. Add in the UIScrollView a UIView (the contentScrollView);
2. In this contentScrollView, set top, bottom, left and right margins to 0
3. Set also align center horizontally and vertically;

# Section 32.3: Scrolling to content inside nested StackViews

The big gotcha about scrolling is to determine the offset necessary to present (for instance) a **Textfield inside a StackView with is inside the ScrollView**.

If you try to get **the position of** `Textfield.frame.minY` **can be 0**, because the minY frame is only considering the distance between the element and the top of the StackView. So you have to **consider all other parent stackviews/views.**

A good workaround for this is:

1 - Implement the ScrollView Extension

```swift
extension UIScrollView {

    func scrollToShowView(view: UIView){
        var offset = view.frame.minY
        var superview = view.superview
        while((superview  != nil)){
            offset += (superview?.frame.minY)!
            superview = superview?.superview
        }

        offset -= 100 //optional margin added on offset

        self.contentOffset = CGPoint.init(x: 0, y: offset)
    }

}
```

This will consider all parent view and sum the necessary offset for the scrollview present the necessary view on screen (for example a Textfield which cannot stay behind the user keyboard)

Usage example:

```swift
func textViewDidBeginEditing(_ textView: UITextView) {
    self.contentScrollView.scrollToShowView(view: textView)
}
```

# Chapter 33: UIScrollView AutoLayout

## Section 33.1: ScrollableController

When using Autolayout with a `UIScrollView`, it does NOT resize properly depending on the size of its contents or subviews.

In order to get a `UIScrollView` to automatically scroll when its contents become too large to fit in the visible area, we need to add a `ContentView` and some constraints that allow the `UIScrollView` to determine the size of its contents AND its width and height in its parent view.

```swift
import Foundation
import UIKit

class ScrollableController : UIViewController {

    private var scrollView: UIScrollView!
    private var contentView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        //Setup
        self.initControls()
        self.setTheme()
        self.layoutScrollView()
        self.layoutContentView()

        //Add child views
        self.addChildViews()
    }

    func initControls() {
        self.scrollView = UIScrollView()
        self.contentView = UIView()
    }

    func setTheme() {
        self.scrollView.backgroundColor = UIColor.blue()
        self.contentView.backgroundColor = UIColor.orange()
    }

    func layoutScrollView() {
        self.view.addSubview(self.scrollView)

        let views: NSDictionary = ["scrollView": self.scrollView]
        var constraints = Array<String>()

        //Constrain the scrollView to our controller's self.view.
        constraints.append("H:|-0-[scrollView]-0-|")
        constraints.append("V:|-0-[scrollView]-0-|")

        for constraint in constraints {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views as! [String : AnyObject]))
        }

        self.scrollView.translatesAutoresizingMaskIntoConstraints = false
    }
```

```swift
    func layoutContentView() {
        self.scrollView.addSubview(self.contentView)

        let views: NSDictionary = ["contentView": self.contentView, "view": self.view]
        var constraints = Array<String>()

        //Constrain the contentView to the scrollView.
        constraints.append("H:|-0-[contentView]-0-|")
        constraints.append("V:|-0-[contentView]-0-|")

        for constraint in constraints {
            self.scrollView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
constraint, options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views as! [String :
AnyObject]))
        }

        //Disable Horizontal Scrolling by making the contentView EqualWidth with our controller's
self.view (ScrollView's parentView).
        self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
"H:[contentView(==view)]", options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views
as! [String : AnyObject]))

        self.contentView.translatesAutoresizingMaskIntoConstraints = false
    }

    func addChildViews() {
        //Init
        let greenView = UIView()
        let whiteView = UIView()

        //Theme
        greenView.backgroundColor = UIColor.green()
        whiteView.backgroundColor = UIColor.orange()

        //Layout -- Child views are added to the 'ContentView'
        self.contentView.addSubview(greenView)
        self.contentView.addSubview(whiteView)

        let views: NSDictionary = ["greenView": greenView, "whiteView": whiteView];
        var constraints = Array<String>()

        //Constrain the greenView to the contentView with a height of 400 and 15 spacing all
around.
        constraints.append("H:|-15-[greenView]-15-|")
        constraints.append("V:|-15-[greenView(400)]")

        //Constrain the whiteView below the greenView with 15 spacing all around and a height of
500.
        constraints.append("H:|-15-[whiteView]-15-|")
        constraints.append("V:[greenView]-15-[whiteView(500)]-15-|")

        for constraint in constraints {
            self.contentView.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
constraint, options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views as! [String :
AnyObject]))
        }

        greenView.translatesAutoresizingMaskIntoConstraints = false
        whiteView.translatesAutoresizingMaskIntoConstraints = false
    }
}
```

Now we can see that the greenView (400 height) + the whiteView (500 height) is larger than our screen. This will cause the ScrollView's contentSize to grow to fit BOTH views, allowing it to scroll vertically.

We disabled horizontal scrolling using the `EqualWidth` constraint on the `contentView` and `self.view`

# Section 33.2: UIScrollView dynamic content size through Storyboard

While using scrollviews in storyboard it's better to calculate content size according to number of views present in scrollview rather than giving content size programatically with static value.

Here are the steps to get content size dynamically.

Step 1 :

Add Scrollview to view in storyboard and add leading, trailing, top and bottom constraints (All values are zero).

Step 2 :

Don't add directly views which you need on directly scrollview, First add one view to scrollview (that will be our content view for all UI elements). Add below constraints to this view.

1. Leading, trailing, top and bottom constraints (All values are zero).

2. Add equal height, equal width to Main view (i.e. which contains scrollview). For equal height set priority to low. (This is the important step for setting content size).

3. Height of this content view will be according to the number of views added to the view. let say if you added last view is one label and his Y position is 420 and height is 20 then your content view will be 440.

Step 3 : Add constraints to all of views which you added within content view as per your requirement.

# Chapter 34: UITextField

UITextField is part of UIKit framework and is used to display an area to collect text input from the user using the onscreen keyboard

## Section 34.1: Get Keyboard Focus and Hide Keyboard

**Get Focus**

**Swift**

```swift
textField.becomeFirstResponder()
```

**Objective-C**

```objc
[textField becomeFirstResponder];
```

**Resign**

**Swift**

```swift
textField.resignFirstResponder()
```

**Objective-C**

```objc
[textField resignFirstResponder];
```

## Section 34.2: Dismiss keyboard when user pushes the return button

Setup your view controller to manage editing of text for the text field.

```swift
class MyViewController: UITextFieldDelegate {

    override viewDidLoad() {
        super.viewDidLoad()

        textField.delegate = self
    }

}
```

textFieldShouldReturn is called every time the return button on the keyboard is pressed.

**Swift:**

```swift
func textFieldShouldReturn(textField: UITextField) -> Bool {
    textField.resignFirstResponder()
    return true;
}
```

**Objective-C:**

```objc
- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    [textField resignFirstResponder];
    return true;
}
```

# Section 34.3: Hide blinking caret

To hide the blinking caret, you need to override caretRectForPosition of a UITextField and return CGRectZero.

**Swift 2.3 <**

```swift
public override func caretRectForPosition(position: UITextPosition) -> CGRect {
    return CGRectZero
}
```

**Swift 3**

```swift
override func caretRect(for position: UITextPosition) -> CGRect {
    return CGRect.zero
}
```

**Objective-C**

```objc
- (CGRect) caretRectForPosition:(UITextPosition*) position{
return CGRectZero;
}
```

# Section 34.4: Input accessory view (toolbar)

Add an accessory view above the keyboard. This is commonly used for adding next/previous buttons, or additional buttons like Done/Submit (especially for the number/phone/decimal pad keyboard types which don't have a built-in return key).

**Swift**

```swift
let textField = UITextField() // initialized however

let toolbar = UIToolbar(frame: CGRect(x: 0, y: 0, width: view.frame.size.width, height: 0)

let flexibleSpace = UIBarButtonItem(barButtonSystemItem: .FlexibleSpace, target: nil, action: nil)

let doneButton = UIBarButtonItem(barButtonSystemItem: .Done, target: self, action:
Selector("done"))

let items = [flexibleSpace, doneButton]  // pushes done button to right side

toolbar.setItems(items, animated: false) // or toolbar.items = ...
toolbar.sizeToFit()

textField.inputAccessoryView = toolbar
```

**Objective-C**

```objc
UITextField *textField = [[UITextField alloc] init];

UIToolbar *toolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
0)];

UIBarButtonItem *flexibleSpace = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:nil];
UIBarButtonItem *doneButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemDone target:self action:@selector(done)];
NSArray *items = @[
                flexibleSpace,
                doneButton
                ];

[toolbar setItems:items];
[toolbar sizeToFit];
```

```
textField.inputAccessoryView = toolbar;
```

# Section 34.5: Moving scroll when UITextView becomes first responder

Observe the notifications UIKeyboardWillShowNotification and UIKeyboardWillHideNotification, update the scrollView content insets according to keyboard height, then scroll to the focused control.

```objc
- (void)viewDidLoad
{
    [super viewDidLoad];

    // register for keyboard notifications
    [[NSNotificationCenter defaultCenter] addObserver:self
                                     selector:@selector(keyboardWillShow:)
                                         name:UIKeyboardWillShowNotification
                                       object:self.view.window];
    // register for keyboard notifications
    [[NSNotificationCenter defaultCenter] addObserver:self
                                     selector:@selector(keyboardWillHide:)
                                         name:UIKeyboardWillHideNotification
                                       object:self.view.window];
}

// Called when UIKeyboardWillShowNotification is sent
- (void)keyboardWillShow:(NSNotification*)notification
{
    // if we have no view or are not visible in any window, we don't care
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }

    NSDictionary *userInfo = [notification userInfo];

    CGRect keyboardFrameInWindow;
    [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardFrameInWindow];

    // the keyboard frame is specified in window-level coordinates. this calculates the frame as if
it were a subview of our view, making it a sibling of the scroll view
    CGRect keyboardFrameInView = [self.view convertRect:keyboardFrameInWindow fromView:nil];

    CGRect scrollViewKeyboardIntersection = CGRectIntersection(_scrollView.frame,
keyboardFrameInView);
    UIEdgeInsets newContentInsets = UIEdgeInsetsMake(0, 0,
scrollViewKeyboardIntersection.size.height, 0);

    // this is an old animation method, but the only one that retains compaitiblity between
parameters (duration, curve) and the values contained in the userInfo-Dictionary.
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:[[userInfo objectForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
    [UIView setAnimationCurve:[[userInfo objectForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

    _scrollView.contentInset = newContentInsets;
    _scrollView.scrollIndicatorInsets = newContentInsets;

    /*
     * Depending on visual layout, _focusedControl should either be the input field
(UITextField,..) or another element
```

```objc
     * that should be visible, e.g. a purchase button below an amount text field
     * it makes sense to set _focusedControl in delegates like -textFieldShouldBeginEditing: if you
have multiple input fields
     */
    if (_focusedControl) {
        CGRect controlFrameInScrollView = [_scrollView convertRect:_focusedControl.bounds
fromView:_focusedControl]; // if the control is a deep in the hierarchy below the scroll view, this
will calculate the frame as if it were a direct subview
        controlFrameInScrollView = CGRectInset(controlFrameInScrollView, 0, -10); // replace 10
with any nice visual offset between control and keyboard or control and top of the scroll view.

        CGFloat controlVisualOffsetToTopOfScrollview = controlFrameInScrollView.origin.y -
_scrollView.contentOffset.y;
        CGFloat controlVisualBottom = controlVisualOffsetToTopOfScrollview +
controlFrameInScrollView.size.height;

        // this is the visible part of the scroll view that is not hidden by the keyboard
        CGFloat scrollViewVisibleHeight = _scrollView.frame.size.height -
scrollViewKeyboardIntersection.size.height;

        if (controlVisualBottom > scrollViewVisibleHeight) { // check if the keyboard will hide the
control in question
            // scroll up until the control is in place
            CGPoint newContentOffset = _scrollView.contentOffset;
            newContentOffset.y += (controlVisualBottom - scrollViewVisibleHeight);

            // make sure we don't set an impossible offset caused by the "nice visual offset"
            // if a control is at the bottom of the scroll view, it will end up just above the
keyboard to eliminate scrolling inconsistencies
            newContentOffset.y = MIN(newContentOffset.y, _scrollView.contentSize.height -
scrollViewVisibleHeight);

            [_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO because we
have created our own animation context around this code
        } else if (controlFrameInScrollView.origin.y < _scrollView.contentOffset.y) {
            // if the control is not fully visible, make it so (useful if the user taps on a
partially visible input field
            CGPoint newContentOffset = _scrollView.contentOffset;
            newContentOffset.y = controlFrameInScrollView.origin.y;

            [_scrollView setContentOffset:newContentOffset animated:NO]; // animated:NO because we
have created our own animation context around this code
        }
    }

    [UIView commitAnimations];
}


// Called when the UIKeyboardWillHideNotification is sent
- (void)keyboardWillHide:(NSNotification*)notification
{
    // if we have no view or are not visible in any window, we don't care
    if (!self.isViewLoaded || !self.view.window) {
        return;
    }

    NSDictionary *userInfo = notification.userInfo;

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:[[userInfo valueForKey:UIKeyboardAnimationDurationUserInfoKey]
doubleValue]];
```

```
    [UIView setAnimationCurve:[[userInfo valueForKey:UIKeyboardAnimationCurveUserInfoKey]
intValue]];

    // undo all that keyboardWillShow-magic
    // the scroll view will adjust its contentOffset apropriately
    _scrollView.contentInset = UIEdgeInsetsZero;
    _scrollView.scrollIndicatorInsets = UIEdgeInsetsZero;

    [UIView commitAnimations];
}
```

# Section 34.6: KeyboardType

To change the appearance of the keyboard, the following types can be set individually on every `UITextFields`
property: `keyboardType`

```
typedef NS_ENUM(NSInteger, UIKeyboardType) {
    UIKeyboardTypeDefault,                   // Default type for the current input method.
    UIKeyboardTypeASCIICapable,              // Displays a keyboard which can enter ASCII characters,
non-ASCII keyboards remain active
    UIKeyboardTypeNumbersAndPunctuation,     // Numbers and assorted punctuation.
    UIKeyboardTypeURL,                       // A type optimized for URL entry (shows . / .com
prominently).
    UIKeyboardTypeNumberPad,                 // A number pad (0-9). Suitable for PIN entry.
    UIKeyboardTypePhonePad,                  // A phone pad (1-9, *, 0, #, with letters under the
numbers).
    UIKeyboardTypeNamePhonePad,              // A type optimized for entering a person's name or phone
number.
    UIKeyboardTypeEmailAddress,              // A type optimized for multiple email address entry
(shows space @ . prominently).
    UIKeyboardTypeDecimalPad NS_ENUM_AVAILABLE_IOS(4_1),    // A number pad with a decimal point.
    UIKeyboardTypeTwitter NS_ENUM_AVAILABLE_IOS(5_0),       // A type optimized for twitter text
entry (easy access to @ #)
    UIKeyboardTypeWebSearch NS_ENUM_AVAILABLE_IOS(7_0),    // A default keyboard type with URL-
oriented addition (shows space . prominently).

    UIKeyboardTypeAlphabet = UIKeyboardTypeASCIICapable, // Deprecated
};
```

# Section 34.7: Change placeholder color and font

We can change the style of the placeholder by setting `attributedPlaceholder` (a `NSAttributedString`).

```
var placeholderAttributes = [String: AnyObject]()
placeholderAttributes[NSForegroundColorAttributeName] = color
placeholderAttributes[NSFontAttributeName] = font

if let placeholder = textField.placeholder {
    let newAttributedPlaceholder = NSAttributedString(string: placeholder, attributes:
placeholderAttributes)
    textField.attributedPlaceholder = newAttributedPlaceholder
}
```

In this example we change only the `color` and `font`. You could change other properties such as underline or
strikethrough style. Refer to `NSAttributedString` for the properties that can be changed.

# Section 34.8: Replace keyboard with UIPickerView

In some cases, you want to show your users a `UIPickerView` with predefined contents for a `UITextField` instead of a keyboard.

**Create a custom UIPickerView**

At first, you need a custom wrapper-class for `UIPickerView` conforming to the protocols `UIPickerViewDataSource` and `UIPickerViewDelegate`.

```
class MyPickerView: UIPickerView, UIPickerViewDataSource, UIPickerViewDelegate
```

You need to implement the following methods for the DataSource and Delegate:

```
public func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    if data != nil {
        return data!.count
    } else {
        return 0
    }
}

public func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
}

public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component:
Int) -> String? {
    if data != nil {
        return data![row]
    } else {
        return ""
    }
}
```

To handle the data, `MyPickerView` needs the properties `data`, `selectedValue` and `textFieldBeingEdited`:

```
/**
 The data for the `UIPickerViewDelegate`

 Always needs to be an array of `String`! The `UIPickerView` can ONLY display Strings
 */
public var data: [String]? {
    didSet {
        super.delegate = self
        super.dataSource = self
        self.reloadAllComponents()
    }
}

/**
 Stores the UITextField that is being edited at the moment
 */
public var textFieldBeingEdited: UITextField?

/**
 Get the selected Value of the picker
 */
public var selectedValue: String {
```

```
    get {
        if data != nil {
            return data![selectedRow(inComponent: 0)]
        } else {
            return ""
        }
    }
}
```

**Prepare your ViewController**

The `ViewController` that contains your textField, needs to have a property for your custom `UIPickerView`. (Assuming, that you already have another property or `@IBOutlet` containing your textField)

```
/**
 The picker view to present as keyboard
 */
var picker: MyPickerView?
```

In your `viewDidLoad()`, you need to initialize `picker` and configure it a bit:

```
picker = MyPickerView()
picker?.autoresizingMask = [.flexibleHeight, .flexibleWidth]
picker?.backgroundColor = UIColor.white()

picker?.data = ["One", "Two", "Three", "Four", "Five"] //The data shown in the picker
```

Now, you can add the `MyPicker` as `inputView` of your `UITextField`:

```
textField.inputView = picker
```

**Dismissing the picker-keyboard**

Now, you have replaced the keyboard by an `UIPickerView`, but there is no possibility to dismiss it. This can be done with a custom `.inputAccessoryView`:

Add the property `pickerAccessory` to your `ViewController`.

```
/**
 A toolbar to add to the keyboard when the `picker` is presented.
 */
var pickerAccessory: UIToolbar?
```

In `viewDidLoad()`, you need to create an `UIToolbar` for the `inputAccessoryView`:

```
pickerAccessory = UIToolbar()
pickerAccessory?.autoresizingMask = .flexibleHeight

//this customization is optional
pickerAccessory?.barStyle = .default
pickerAccessory?.barTintColor = UIColor.red()
pickerAccessory?.backgroundColor = UIColor.red()
pickerAccessory?.isTranslucent = false
```

You should set the frame of your toolbar. To fit in the design of iOS, it's recommended to use a height of `44.0`:

```
var frame = pickerAccessory?.frame
```

```
frame?.size.height = 44.0
pickerAccessory?.frame = frame!
```

For a good user experience, you should add two buttons ("Done" and "Cancel"), but it would also work with only one that dismisses the keyboard.

```
let cancelButton = UIBarButtonItem(barButtonSystemItem: .cancel, target: self, action:
#selector(ViewController.cancelBtnClicked(_:)))
cancelButton.tintColor = UIColor.white()
let flexSpace = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil) //a
flexible space between the two buttons
let doneButton = UIBarButtonItem(barButtonSystemItem: .done, target: self, action:
#selector(ViewController.doneBtnClicked(_:)))
doneButton.tintColor = UIColor.white()

//Add the items to the toolbar
pickerAccessory?.items = [cancelButton, flexSpace, doneButton]
```

Now you can add the toolbar as `inputAccessoryView`

```
textField.inputAccessoryView = pickerAccessory
```

Before you can build your project, you need to implement the methods, the buttons are calling:

```
/**
 Called when the cancel button of the `pickerAccessory` was clicked. Dismsses the picker
 */
func cancelBtnClicked(_ button: UIBarButtonItem?) {
    textField?.resignFirstResponder()
}

/**
 Called when the done button of the `pickerAccessory` was clicked. Dismisses the picker and puts
the selected value into the textField
 */
func doneBtnClicked(_ button: UIBarButtonItem?) {
    textField?.resignFirstResponder()
    textField.text = picker?.selectedValue
}
```

Run your project, tap the `textField` and you should see a picker like this instead of the keyboard:

**Select a value programmatically (optional)**

If you don't want to have the first row selected automatically, you can set the selected row as in `UIPickerView`:

```
picker?.selectRow(3, inComponent: 0, animated: false) //Will select the row at index 3
```

# Section 34.9: Create a UITextField

Initialize the `UITextField` with a CGRect as a frame:

**Swift**

```
let textfield = UITextField(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

**Objective-C**

```
UITextField *textField = [[UITextField alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

You can also create a `UITextField` in Interface Builder:

# Section 34.10: Getting and Setting the Cursor Position

**Useful information**

The very beginning of the text field text:

```
let startPosition: UITextPosition = textField.beginningOfDocument
```

The very end of the text field text:

```
let endPosition: UITextPosition = textField.endOfDocument
```

The currently selected range:

```
let selectedRange: UITextRange? = textField.selectedTextRange
```

**Get cursor position**

```
if let selectedRange = textField.selectedTextRange {

    let cursorPosition = textField.offsetFromPosition(textField.beginningOfDocument, toPosition:
selectedRange.start)

    print("\(cursorPosition)")
}
```

**Set cursor position**

In order to set the position, all of these methods are actually setting a range with the same start and end values.

**To the beginning**

```
let newPosition = textField.beginningOfDocument
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To the end**

```
let newPosition = textField.endOfDocument
textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition: newPosition)
```

**To one position to the left of the current cursor position**

```
// only if there is a currently selected range
if let selectedRange = textField.selectedTextRange {

    // and only if the new position is valid
    if let newPosition = textField.positionFromPosition(selectedRange.start, inDirection:
UITextLayoutDirection.Left, offset: 1) {

        // set the new position
        textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition:
newPosition)
    }
}
```

**To an arbitrary position**

Start at the beginning and move 5 characters to the right.

```
let arbitraryValue: Int = 5
if let newPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: arbitraryValue) {

    textField.selectedTextRange = textField.textRangeFromPosition(newPosition, toPosition:
newPosition)
}
```

**Related**

**Select all text**

```
textField.selectedTextRange = textField.textRangeFromPosition(textField.beginningOfDocument,
toPosition: textField.endOfDocument)
```

**Select a range of text**

```
// Range: 3 to 7
let startPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: 3)
let endPosition = textField.positionFromPosition(textField.beginningOfDocument, inDirection:
UITextLayoutDirection.Right, offset: 7)

if startPosition != nil && endPosition != nil {
    textField.selectedTextRange = textField.textRangeFromPosition(startPosition!, toPosition:
endPosition!)
}
```

**Insert text at the current cursor position**

```
textField.insertText("Hello")
```

**Notes**

- This example originally comes from [this Stack Overflow answer](#).

- This answer uses a text field, but the same concepts apply to `UITextView`.

- Use `textField.becomeFirstResponder()` to give focus to the text field and make the keyboard appear.

- See [this answer](#) for how to get the text at some range.

**Related**

- [How to Create a Range in Swift](#) (Deals indirectly with the issue of why we have to use `selectedTextRange` here rather than just `selectedRange`)

# Section 34.11: Dismiss Keyboard

**Swift**

Ctrl + Drag from the UItextfield in MainStoryboard to the ViewController Class and create a UITextField Outlet

After that select the UItextField again and Ctrl+drag in ViewController class but this time select **Action** connection and on storage select **Did End On Exit** then click connect.

in the action you just created type the name of your UItextField `.resignFirstResponder()`

```
@IBAction func textFieldResign(sender: AnyObject) {
    yourTextFieldName.resignFirstResponder()
}
```

This will take care of hiding the keyboard when pressing the return key on keyboard.

**Another example of hiding the keyboard when return key is pressed:**

we add `UITextFieldDelegate` protocol next to `UIViewController`

in the vieDidLoad function we add `self.yourTextFieldName.delegate = self`

And Finally we add this

```
func textFieldShouldReturn(textField: UITextField) -> Bool {
        yourTextFieldName.resignFirstResponder()
        return true
    }
```

The final code is this:

```
class ViewController: UIViewController, UITextFieldDelegate  {

@IBOutlet var textField: UITextField!

    func textFieldShouldReturn(textField: UITextField) -> Bool {
        textField.resignFirstResponder()
```

```
        return true
    }

override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?){
    view.endEditing(true)
    super.touchesBegan(touches, withEvent: event)
}


override func viewDidLoad() {
    super.viewDidLoad()
    self.textField.delegate = self
        }

}
```

**Objective-C**

```
[textField resignFirstResponder];
```

# Section 34.12: Initialize text field

**Swift**

```
let frame = CGRect(x: 0, y: 0, width: 100, height: 100)
let textField = UITextField(frame: frame)
```

**Objective-C**

```
CGRect *frame = CGRectMake(0, 0, 100, 100);
UITextField *textField = [[UITextField alloc] initWithFrame:frame];
```

**Interface Builder**

You can also add a UITextField to a storyboard by dragging it from Object Library.



# Section 34.13: Autocapitalization

**Swift**

```
textField.autocapitalizationType = .None
```

**Objective-C**

```
textField.autocapitalizationType = UITextAutocapitalizationTypeNone;
```

All options:

- .None \ UITextAutocapitalizationTypeNone : Don't autocapitalize anything
- .Words \ UITextAutocapitalizationTypeWords : Autocapitalize every word
- .Sentences \ UITextAutocapitalizationTypeSentences : Autocapitalize the first word in a sentence
- .AllCharacters \ UITextAutocapitalizationTypeAllCharacters : Autocapitalize every letter (i.e. caps lock)

# Section 34.14: Set Alignment

**Swift**

```swift
textField.textAlignment = .Center
```

**Objective-C**

```objc
[textField setTextAlignment: NSTextAlignmentCenter];
```

In the example, we have set the `NSTextAlignment` to center. You can also set to `.Left`, `.Right`, `.Justified` and `.Natural`.

`.Natural` is the default alignment for the current localization. That means for left-to-right languages (eg. English), the alignment is `.Left`; for right-to-left languages, it is `.Right`.

# Chapter 35: Custom UITextField

Using custom UITextField, we can manipulate the behavior of text field!

## Section 35.1: Custom UITextField for Filtering Input Text

Here is an example of custom `UITextField` that takes only numerical text and discards all other.

**NOTE:** For iPhone it is easy to do this using Number type keyboard, but for iPad there is no keyboard with Numbers only

```swift
class NumberTextField: UITextField {

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
    registerForTextFieldNotifications()
}

override init(frame: CGRect) {
    super.init(frame: frame)
}

override func awakeFromNib() {
    super.awakeFromNib()
    keyboardType = .numberPad//useful for iPhone only
}

private func registerForTextFieldNotifications() {
    NotificationCenter.default.addObserver(self, selector:
#selector(NumberTextField.textDidChange), name: NSNotification.Name(rawValue:
"UITextFieldTextDidChangeNotification"), object: self)
}

deinit {
    NotificationCenter.default.removeObserver(self)
}

func textDidChange() {
    text = filteredText()
}
private func filteredText() -> String {
    let inverseSet = CharacterSet(charactersIn:"0123456789").inverted
    let components = text!.components(separatedBy: inverseSet)
    return components.joined(separator: "")
}
}
```

So, wherever we want text field which would take only numbers as input text, then we can use this custom UITextField

## Section 35.2: Custom UITextField to Disallow All Actions like Copy, Paste, etc

If we want to disable all the actions like Copy, Paste, Replace, Select, etc from `UITextField` then we can use following custom text field:

```swift
class CustomTextField: UITextField {
```

```swift
var enableLongPressActions = false

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)!
}

override init(frame: CGRect) {
    super.init(frame: frame)
}

override func canPerformAction(_ action: Selector, withSender sender: Any?) -> Bool {
    return enableLongPressActions
}
}
```

Using enableLongPressActions property, we can enable all actions any time later, if needed.

# Chapter 36: UIViewController

## Section 36.1: Subclassing

Subclassing `UIControl` gives us access to the following methods:

- `beginTrackingWithTouch` is called when the finger first touches down within the control's bounds.
- `continueTrackingWithTouch` is called repeatedly as the finger slides across the control and even outside of the control's bounds.
- `endTrackingWithTouch` is called when the finger lifts off the screen.

*MyCustomControl.swift*

```swift
import UIKit

// These are out self-defined rules for how we will communicate with other classes
protocol ViewControllerCommunicationDelegate: class {
    func myTrackingBegan()
    func myTrackingContinuing(location: CGPoint)
    func myTrackingEnded()
}

class MyCustomControl: UIControl {

    // whichever class wants to be notified of the touch events must set the delegate to itself
    weak var delegate: ViewControllerCommunicationDelegate?

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {

        // notify the delegate (i.e. the view controller)
        delegate?.myTrackingBegan()

        // returning true means that future events (like continueTrackingWithTouch and
endTrackingWithTouch) will continue to be fired
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {

        // get the touch location in our custom control's own coordinate system
        let point = touch.locationInView(self)

        // Update the delegate (i.e. the view controller) with the new coordinate point
        delegate?.myTrackingContinuing(point)

        // returning true means that future events will continue to be fired
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {

        // notify the delegate (i.e. the view controller)
        delegate?.myTrackingEnded()
    }
}
```

*ViewController.swift*

This is how the view controller is set up to be the delegate and respond to touch events from our custom control.

```swift
import UIKit
class ViewController: UIViewController, ViewControllerCommunicationDelegate {

    @IBOutlet weak var myCustomControl: MyCustomControl!
    @IBOutlet weak var trackingBeganLabel: UILabel!
    @IBOutlet weak var trackingEndedLabel: UILabel!
    @IBOutlet weak var xLabel: UILabel!
    @IBOutlet weak var yLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        myCustomControl.delegate = self
    }

    func myTrackingBegan() {
        trackingBeganLabel.text = "Tracking began"
    }

    func myTrackingContinuing(location: CGPoint) {
        xLabel.text = "x: \(location.x)"
        yLabel.text = "y: \(location.y)"
    }

    func myTrackingEnded() {
        trackingEndedLabel.text = "Tracking ended"
    }
}
```

**Notes**

- Alternate methods of achieving the same result without subclassing include adding a target or using a gesture recognizer.

- It is not necessary to use a delegate with these methods if they are only being used within the custom control itself. We could have just added a `print` statement to show how the events are being called. In that case, the code would be simplified to

```swift
import UIKit
class MyCustomControl: UIControl {

    override func beginTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        print("Began tracking")
        return true
    }

    override func continueTrackingWithTouch(touch: UITouch, withEvent event: UIEvent?) -> Bool {
        let point = touch.locationInView(self)
        print("x: \(point.x), y: \(point.y)")
        return true
    }

    override func endTrackingWithTouch(touch: UITouch?, withEvent event: UIEvent?) {
        print("Ended tracking")
    }
}
```

# Section 36.2: Access the container view controller

When the view controller is presented within a tab bar controller, you can access the tab bar controller like this:

**Swift**

```swift
let tabBarController = viewController.tabBarController
```

**Objective-C**

```objc
UITabBarController *tabBarController = self.tabBarController;
```

When the view controller is part on an navigation stack, you can access the navigation controller like this:

**Swift**

```swift
let navigationController = viewController.navigationController
```

**Objective-C**

```objc
UINavigationController *navigationController = self.navigationController;
```

# Section 36.3: Set the view programmatically

Swift

```swift
class FooViewController: UIViewController {

  override func loadView() {
    view = FooView()
  }

}
```

# Section 36.4: Instantiate from a Storyboard

```objc
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
```

**With an Identifier**:

Give the scene a Storyboard ID within the identity inspector of the storyboard.
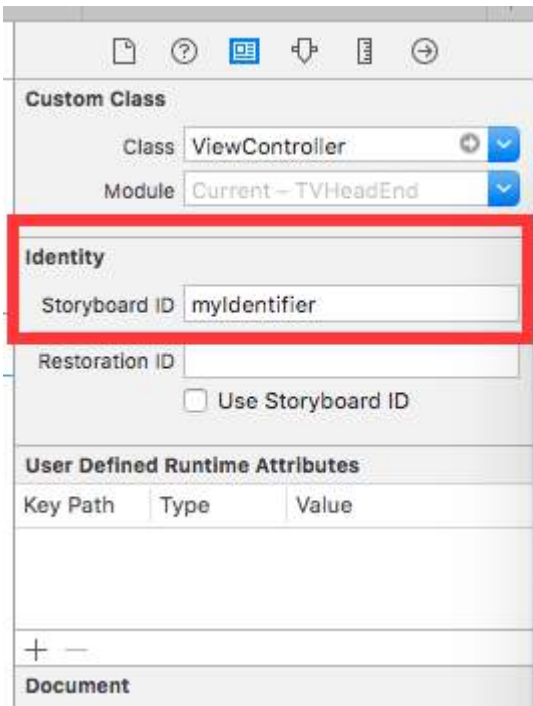
Instantiate in code:

```
UIViewController *controller = [storyboard
instantiateViewControllerWithIdentifier:@"myIdentifier"];
```

**Instantiate an initial viewcontroller**:

Within the storyboard select the view controller, then select the attribute inspector, check the "Is Initial View Controller" box.
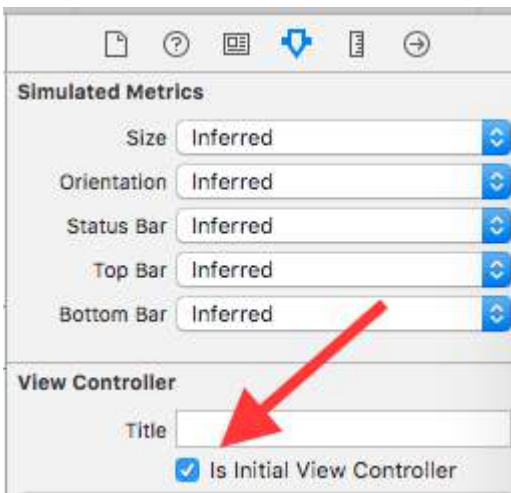


```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
UIViewController *controller = [storyboard instantiateInitialViewController];
```

# Section 36.5: Create an instance

**Swift**

```
let viewController = UIViewController()
```

**Objective-C**

```
UIViewController *viewController = [UIViewController new];
```

# Section 36.6: Adding/removing a child view controller

To add a child view controller:

```
- (void)displayContentController:(UIViewController *)vc {
    [self addChildViewController:vc];
    vc.view.frame = self.view.frame;
    [self.view addSubview:vc.view];
    [vc didMoveToParentViewController:self];
}
```

To remove a child view controller:

```
- (void)hideContentController:(UIViewController *)vc {
    [vc willMoveToParentViewController:nil];
    [vc.view removeFromSuperview];
    [vc removeFromParentViewController];
}
```

# Chapter 37: UISwitch

## Section 37.1: Set Image for On/Off state

**Objective-C**

```objc
//set off-image
mySwitch.offImage = [UIImage imageNamed:@"off_image"];
[mySwitch setOffImage:[UIImage imageNamed:@"off_image"]];

//set on-image
mySwitch.onImage = [UIImage imageNamed:@"on_image"];
[mySwitch setOnImage:[UIImage imageNamed:@"on_image"]];
```

**Swift**

```swift
//set off-image
mySwitch.offImage = UIImage(named: "off_image")

//set on-image
mySwitch.onImage = UIImage(named: "on_image")
```

## Section 37.2: Set On / Off

**Objective-C**

```objc
[mySwitch setOn:YES];
//or
[mySwitch setOn:YES animated:YES];
```

**Swift**

```swift
mySwitch.setOn(false)
//or
mySwitch.setOn(false, animated: false)
```

## Section 37.3: Set Background Color

**Objective-C**

```objc
mySwitch.backgroundColor = [UIColor yellowColor];
[mySwitch setBackgroundColor: [UIColor yellowColor]];
mySwitch.backgroundColor =[UIColor colorWithRed:255/255.0 green:0/255.0 blue:0/255.0 alpha:1.0];
mySwitch.backgroundColor= [UIColor colorWithWhite: 0.5 alpha: 1.0];
mySwitch.backgroundColor=[UIColor colorWithHue: 0.4 saturation: 0.3 brightness:0.7 alpha: 1.0];
```

**Swift**

```swift
mySwitch.backgroundColor = UIColor.yellow
mySwitch.backgroundColor = UIColor(red: 255.0/255, green: 0.0/255, blue: 0.0/255, alpha: 1.0)
mySwitch.backgroundColor = UIColor(white: 0.5, alpha: 1.0)
mySwitch.backgroundColor = UIColor(hue: 0.4,saturation: 0.3,brightness: 0.7,alpha: 1.0)
```

# Section 37.4: Set Tint Color

**Objective-C**

```objc
//for off-state
mySwitch.tintColor = [UIColor blueColor];
[mySwitch setTintColor: [UIColor blueColor]];

//for on-state
mySwitch.onTintColor = [UIColor cyanColor];
[mySwitch setOnTintColor: [UIColor cyanColor]];
```

**Swift**

```swift
//for off-state
mySwitch.tintColor = UIColor.blueColor()

//for on-state
mySwitch.onTintColor = UIColor.cyanColor()
```

# Chapter 38: UICollectionView

## Section 38.1: Create a UICollectionView

Initialize a `UICollectionView` with a `CGRect` frame:

**Swift:**

```swift
let collection = UICollectionView(frame: CGRect(x: 0, y: 0, width: 200, height: 21))
```

**Objective C:**

```objc
UICollectionView *collection = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0, 200, 21)];
```

You can also create a `UICollectionView` in Interface Builder
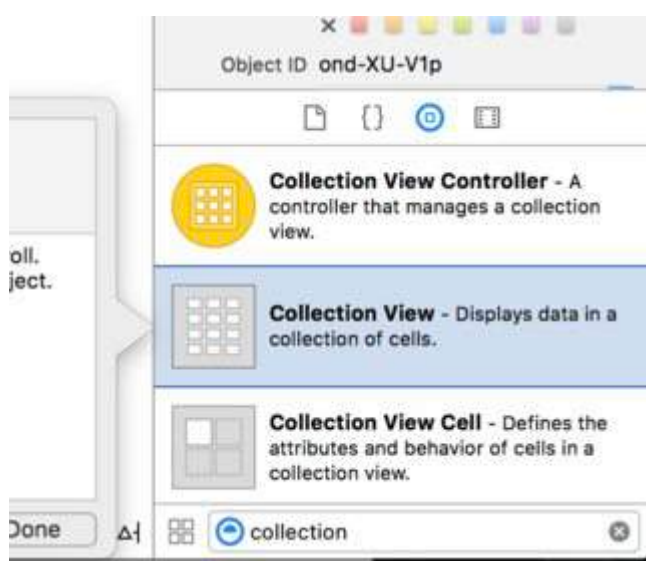


## Section 38.2: UICollectionView - Datasource

Every collection view must have a `Datasource` object. The `Datasource` object is the content that your app will display within the `UICollectionView`. At a minimum, all `Datasource` objects must implement the `collectionView:numberOfItemsInSection:` and `collectionView:cellForItemAtIndexPath:` methods.

*Required Methods*

**Swift**

```swift
func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
    // Return how many items in section
    let sectionArray = _data[section]
    return sectionArray.count
}

func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
NSIndexPath) -> UICollectionViewCell {

    let cell = collectionView.dequeueReusableCellWithReuseIdentifier(MyCellID)
    // If you use a custom cell class then cast the cell returned, like:
    // as! MyCollectionViewCellClass
    // or you will have errors when you try to use features of that class.
```

```
    //Customize your cell here, default UICollectionViewCells do not contain any inherent
    //text or image views (like UITableView), but some could be added,
    //or a custom UICollectionViewCell sub-class could be used
    return cell
}
```

**Objective C**

```
- (NSInteger)collectionView:(UICollectionView*)collectionView
numberOfItemsInSection:(NSInteger)section {
    // Return how many items in section
    NSArray *sectionArray = [_data objectAtIndex:section];
    return [sectionArray count];
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
                cellForItemAtIndexPath:(NSIndexPath *)indexPath {
    // Return a cell
    UICollectionViewCell *newCell = [self.collectionView
                                    dequeueReusableCellWithReuseIdentifier:MyCellID

                                                    forIndexPath:indexPath];
    //Customize your cell here, default UICollectionViewCells do not contain any inherent
    //text or image views (like UITableView), but some could be added,
    //or a custom UICollectionViewCell sub-class could be used
    return newCell;
}
```

# Section 38.3: Basic Swift example of a Collection View

**Create a new project**

It can be just a Single View Application.

**Add the code**

Create a new Cocoa Touch Class file (File > New > File... > iOS > Cocoa Touch Class). Name it `MyCollectionViewCell`. This class will hold the outlets for the views that you add to your cell in the storyboard.

```
import UIKit
class MyCollectionViewCell: UICollectionViewCell {

    @IBOutlet weak var myLabel: UILabel!
}
```

We will connect this outlet later.

Open ViewController.swift and make sure you have the following content:

```
import UIKit
class ViewController: UIViewController, UICollectionViewDataSource, UICollectionViewDelegate {

    let reuseIdentifier = "cell" // also enter this string as the cell identifier in the storyboard
    var items = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15",
"16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31",
"32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47",
"48"]
```

```swift
    // MARK: - UICollectionViewDataSource protocol

    // tell the collection view how many cells to make
    func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) ->
Int {
        return self.items.count
    }

    // make a cell for each cell index path
    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath:
NSIndexPath) -> UICollectionViewCell {

        // get a reference to our storyboard cell
        let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier,
forIndexPath: indexPath) as! MyCollectionViewCell

        // Use the outlet in our custom class to get a reference to the UILabel in the cell
        cell.myLabel.text = self.items[indexPath.item]
        cell.backgroundColor = UIColor.yellowColor() // make cell more visible in our example
project

        return cell
    }

    // MARK: - UICollectionViewDelegate protocol

    func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath) {
        // handle tap events
        print("You selected cell #\(indexPath.item)!")
    }
}
```

**Notes**

- UICollectionViewDataSource and UICollectionViewDelegate are the protocols that the collection view follows. You could also add the UICollectionViewDelegateFlowLayout protocol to change the size of the views programmatically, but it isn't necessary.
- We are just putting simple strings in our grid, but you could certainly do images later.

**Setup the storyboard**

Drag a Collection View to the View Controller in your storyboard. You can add constraints to make it fill the parent view if you like.
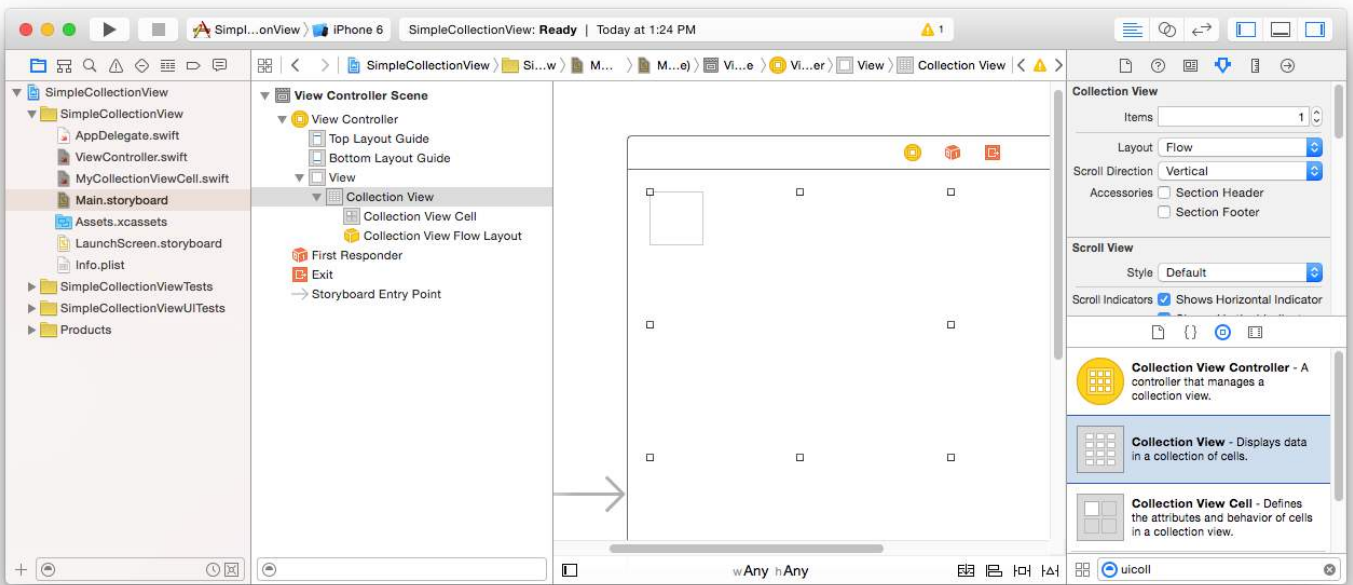
Make sure that your defaults in the Attribute Inspector are also

- Items: 1
- Layout: Flow

The little box in the top left of the Collection View is a Collection View Cell. We will use it as our prototype cell. Drag a Label into the cell and center it. You can resize the cell borders and add constraints to center the Label if you like.
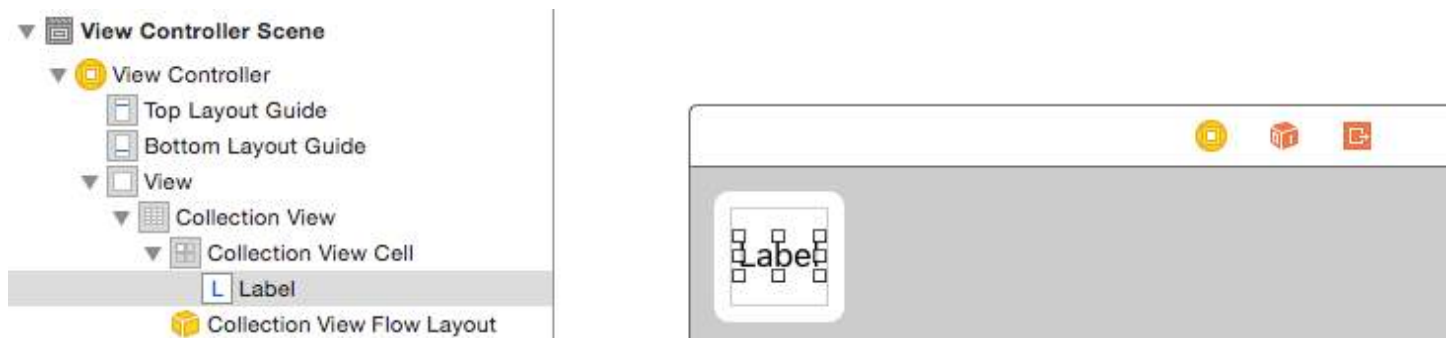


Write "cell" (without quotes) in the Identifier box of the Attributes Inspector for the Collection View Cell. Note that this is the same value as `let reuseIdentifier = "cell"` in ViewController.swift.
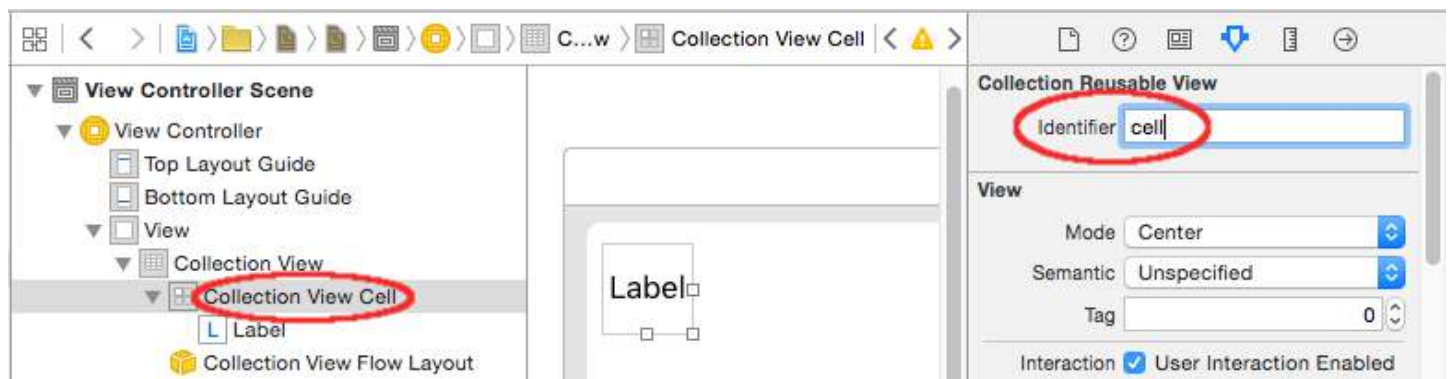


And in the Identity Inspector for the cell, set the class name to `MyCollectionViewCell`, our custom class that we made.
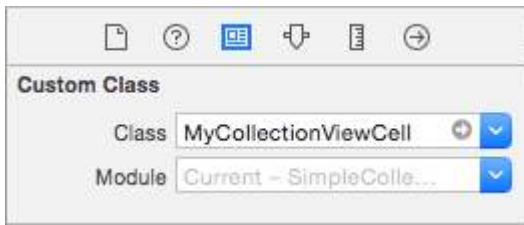
**Hook up the outlets**

- Hook the Label in the collection cell to `myLabel` in the `MyCollectionViewCell` class. (You can [Control-drag](#).)
- Hook the Collection View `delegate` and `dataSource` to the View Controller. (Right click Collection View in the Document Outline. Then click and drag the plus arrow up to the View Controller.)
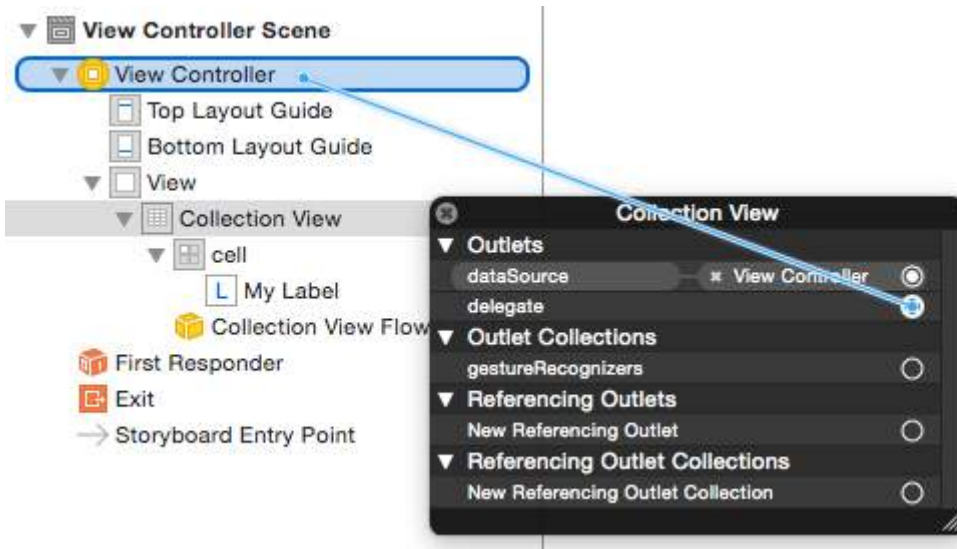


**Finished**

Here is what it looks like after adding constraints to center the Label in the cell and pinning the Collection View to the walls of the parent.
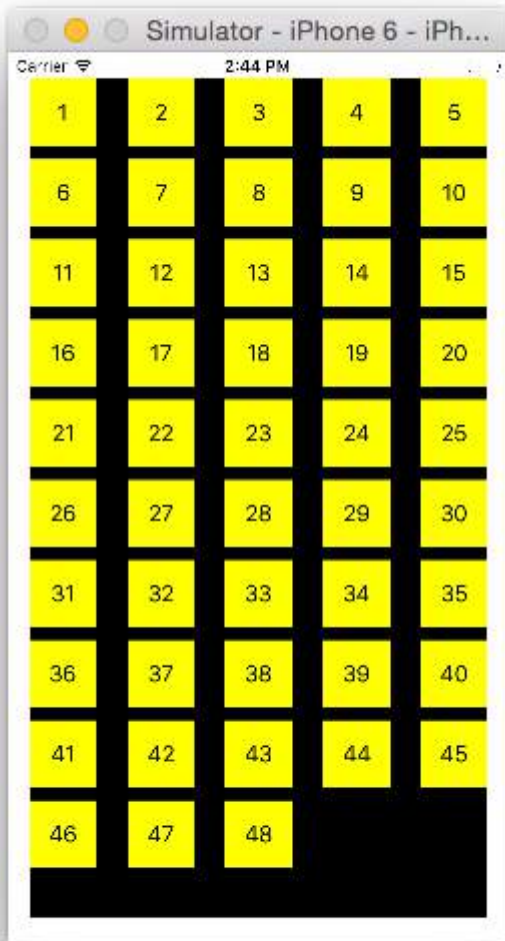
**Making Improvements**

If you want to make improvements on the appearance, see the original post that this example comes from.
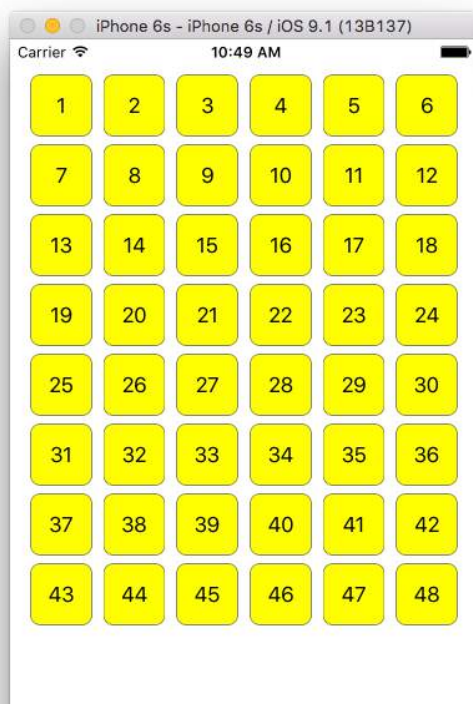
**Further study**

- [A Simple UICollectionView Tutorial](#)
- [UICollectionView Tutorial Part 1: Getting Started](#)
- [UICollectionView Tutorial Part 2: Reusable Views and Cell Selection](#)

# Section 38.4: Manage Multiple Collection view with DataSource and Flowlayout

Here we are managing multiple collection there delegate methods with didselect events.

```swift
extension ProductsVC: UICollectionViewDelegate, UICollectionViewDataSource{

    // MARK: - UICollectionViewDataSource
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section:
Int) -> Int {
        guard collectionView == collectionCategory else {
            return arrOfProducts.count
        }
        return arrOfCategory.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
-> UICollectionViewCell {

        guard collectionView == collectionProduct else {
            let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
"ProductCategoryCell", for: indexPath) as! ProductCategoryCell
            cell.viewBackground.layer.borderWidth = 0.5
            //Do some thing as per use
            return cell
        }

        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: cellIdentifier, for:
indexPath) as! ProductCell
        cell.contentView.layer.borderWidth = 0.5
        cell.contentView.layer.borderColor = UIColor.black.cgColor
        let json = arrOfProducts[indexPath.row]
        //Do something as per use

        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath:
IndexPath) {
        guard collectionView == collectionCategory else {
            let json = arrOfProducts[indexPath.row]
            // Do something for  collectionProduct here
            return
        }
        let json = arrOfCategory[indexPath.row] as [String: AnyObject]
        let id = json["cId"] as? String ?? ""
        // Do something
    }
}

    extension ProductsVC: UICollectionViewDelegateFlowLayout{

    // MARK: - UICollectionViewDelegateFlowLayout
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
```

```swift
UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {

        let collectionWidth = collectionView.bounds.width
        guard collectionView == collectionProduct else {
            var itemWidth = collectionWidth / 4 - 1;

            if(UI_USER_INTERFACE_IDIOM() == .pad) {
                itemWidth = collectionWidth / 4 - 1;
            }
            return CGSize(width: itemWidth, height: 50)
        }

        var itemWidth = collectionWidth / 2 - 1;
        if(UI_USER_INTERFACE_IDIOM() == .pad) {
            itemWidth = collectionWidth / 4 - 1;
        }
        return CGSize(width: itemWidth, height: 250);
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, minimumInteritemSpacingForSectionAt section: Int) -> CGFloat {
        return 1
    }

    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, minimumLineSpacingForSectionAt section: Int) -> CGFloat {
        return 1
    }

}
```

## Section 38.5: UICollectionViewDelegate setup and item selection

Sometimes, if an action should be bind to a collection view's cell selection, you have to implement the `UICollectionViewDelegate` protocol.

Let's say the collection view is inside a `UIViewController` MyViewController.

**Objective-C**

In your *MyViewController.h* declares that it implements the `UICollectionViewDelegate` protocol, as below

```
@interface MyViewController : UIViewController <UICollectionViewDelegate, .../* previous existing delegate, as UICollectionDataSource */>
```

**Swift**

In your *MyViewController.swift* add the following

```
class MyViewController : UICollectionViewDelegate {
```

```
}
```

The method that must be implemented is

**Objective-C**

```objc
-(void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath
*)indexPath
{
}
```

**Swift**

```swift
func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath)
{
}
```

As just an example we can set the background color of selected cell to green.

**Objective-C**

```objc
-(void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath
*)indexPath
{
    UICollectionViewCell* cell = [collectionView cellForItemAtIndexPath:indexPath];
    cell.backgroundColor = [UIColor greenColor];
}
```

**Swift**

```swift
class MyViewController : UICollectionViewDelegate {
    func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath indexPath:
NSIndexPath)
    {
        var cell : UICollectionViewCell = collectionView.cellForItemAtIndexPath(indexPath)!
        cell.backgroundColor = UIColor.greenColor()
    }
}
```

# Section 38.6: Create a Collection View Programmatically

**Swift**

```swift
func createCollectionView() {
    let layout: UICollectionViewFlowLayout = UICollectionViewFlowLayout()
    let collectionView = UICollectionView(frame: CGRect(x: 0, y: 0, width: view.frame.width,
height: view.frame.height), collectionViewLayout: layout)
    collectionView.dataSource = self
    collectionView.delegate = self
    view.addSubview(collectionView)
}
```

**Objective-C**

```objc
- (void)createCollectionView {
    UICollectionViewFlowLayout *layout = [[UICollectionViewFlowLayout alloc] init];
    UICollectionView *collectionView = [[UICollectionView alloc] initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height) collectionViewLayout:layout];
    [collectionView setDataSource:self];
    [collectionView setDelegate:self];
    [self.view addSubview:collectionView];
}
```

## Section 38.7: Swift - UICollectionViewDelegateFlowLayout

```swift
// MARK: - UICollectionViewDelegateFlowLayout
extension ViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAtIndexPath indexPath: NSIndexPath) -> CGSize {
        return CGSize(width: 50, height: 50)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, insetForSectionAtIndex section: Int) -> UIEdgeInsets {
        return UIEdgeInsets(top: 5, left: 5, bottom: 5, right: 5)
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, minimumLineSpacingForSectionAtIndex section: Int) -> CGFloat {
        return 5.0
    }
    func collectionView(collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, minimumInteritemSpacingForSectionAtIndex section: Int) -> CGFloat {
        return 5.0
    }
}
```

## Section 38.8: Performing batch updates

You can animate complex changes to your collection view using the `performBatchUpdates` method. Inside the update block, you can specify several modifications to have them animate all at once.

```
collecitonView.performBatchUpdates({
    // Perform updates
}, nil)
```

Inside the update block, you can perform insertions, deletions, moves, and reloads. Here is how to determine which indexPath to use:

| Type | NSIndexPath |
|---|---|
| Insertion | Index in new array |
| Deletion | Index in old array |
| Move | from: old array, to: new array |
| Reload | either new or old array (it shouldn't matter) |

You should only call reload on cells that have not moved, but their content has changed. It is important to note that a move will not refresh the content of a cell, but only move its location.

To verify that your batch update will be performed correctly, make sure the set of indexPaths for `deletion`, `move-from`, and `reload` are unique, and the set of indexPaths for `insertion`, `move-to`, and `reload` are unique.

Here's an example of a proper batch update:

```
let from = [1, 2, 3, 4, 5]
let to = [1, 3, 6, 4, 5]

collecitonView.performBatchUpdates({
    collectionView.insertItemsAtIndexPaths([NSIndexPath(forItem: 2, inSection: 0)])
    collectionView.deleteItemsAtIndexPaths([NSIndexPath(forItem: 1, inSection: 0)])
    collectionView.moveItemAtIndexPath(NSIndexPath(forItem: 2, inSection: 0),
                                       toIndexPath: NSIndexPath(forItem: 1, inSection:0))
}, nil)
```

# Chapter 39: UISearchController

| Parameter | Details |
| --- | --- |
| `UISearchController.searchBar` | The search bar to install in your interface. *(read-only)* |
| `UISearchController.searchResultsUpdater` | The object responsible for updating the contents of the search results controller. |
| `UISearchController.isActive` | The presented state of the search interface. |
| `UISearchController.obscuresBackgroundDuringPresentation` | A Boolean indicating whether the underlying content is obscured during a search. |
| `UISearchController.dimsBackgroundDuringPresentation` | A Boolean indicating whether the underlying content is dimmed during a search. |
| `UISearchController.hidesNavigationBarDuringPresentation` | A Boolean indicating whether the navigation bar should be hidden when searching. |
| `UIViewController.definesPresentationContext` | A Boolean value that indicates whether this view controller's view is covered when the view controller or one of its descendants presents a view controller. |
| `UIViewController.navigationItem.titleView` | A custom view displayed in the center of the navigation bar when the receiver is the top item in which a search bar can be placed. |
| `UITableViewController.tableView.tableHeaderView` | Returns an accessory view that is displayed above the table in which a search bar can be placed. |

## Section 39.1: Search Bar in Navigation Bar Title

This example uses a search controller to filter the data inside a table view controller. The search bar is placed inside the navigation bar that the table view is embedded into.

Embed a `UITableViewController` into a `UINavigationController` to get the `UINavigationItem` (which contains the navigation bar). Then set our custom ViewController class to inherit from `UITableViewController` and adopt the

`UISearchResultsUpdating` protocol.

```swift
class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "Easiest", image: "green_circle"),
                   (title: "Intermediate", image: "blue_square"),
                   (title: "Advanced", image: "black_diamond"),
                   (title: "Expert Only", image: "double_black_diamond")]

    // An empty tuple that will be updated with search results.
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // Place the search bar in the navigation item's title view.
        self.navigationItem.titleView = searchController.searchBar

        // Don't hide the navigation bar because the search bar is in it.
        searchController.hidesNavigationBarDuringPresentation = false
    }

    func filterContent(for searchText: String) {
        // Update the searchResults array with matches
        // in our entries based on the title value.
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // Return the tuple if the range contains a match.
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating method

    func updateSearchResults(for searchController: UISearchController) {
        // If the search bar contains text, filter our data with the string
        if let searchText = searchController.searchBar.text {
            filterContent(for: searchText)
            // Reload the table view with the search result data.
            tableView.reloadData()
        }
    }

    // MARK: - UITableViewController methods

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // If the search bar is active, use the searchResults data.
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        // If the search bar is active, use the searchResults data.
        let entry = searchController.isActive ?
                    searchResults[indexPath.row] : entries[indexPath.row]
```

```
        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
        cell.textLabel?.text = entry.title
        cell.imageView?.image = UIImage(named: entry.image)
        return cell
    }
}
```

## Section 39.2: Search Bar in Table View Header

This example uses a search controller to filter the cells in a table view controller. The search bar is placed inside the header view of the table view. The table view content is offset with the same height as the search bar so that the search bar is hidden at first. Upon scrolling up past the top edge of the table view, the search bar is revealed. Then when the search bar becomes active, it hides the navigation bar.

Embed a UITableViewController into a UINavigationController to get the UINavigationItem (which contains the navigation bar). Then set our custom ViewController class to inherit from UITableViewController and adopt the

UISearchResultsUpdating protocol.

```swift
class ViewController: UITableViewController, UISearchResultsUpdating {

    let entries = [(title: "Easiest", image: "green_circle"),
                   (title: "Intermediate", image: "blue_square"),
                   (title: "Advanced", image: "black_diamond"),
                   (title: "Expert Only", image: "double_black_diamond")]

    // An empty tuple that will be updated with search results.
    var searchResults : [(title: String, image: String)] = []

    let searchController = UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()

        searchController.searchResultsUpdater = self
        self.definesPresentationContext = true

        // Place the search bar in the table view's header.
        self.tableView.tableHeaderView = searchController.searchBar

        // Set the content offset to the height of the search bar's height
        // to hide it when the view is first presented.
        self.tableView.contentOffset = CGPoint(x: 0, y: searchController.searchBar.frame.height)
    }

    func filterContent(for searchText: String) {
        // Update the searchResults array with matches
        // in our entries based on the title value.
        searchResults = entries.filter({ (title: String, image: String) -> Bool in
            let match = title.range(of: searchText, options: .caseInsensitive)
            // Return the tuple if the range contains a match.
            return match != nil
        })
    }

    // MARK: - UISearchResultsUpdating method

    func updateSearchResults(for searchController: UISearchController) {
        // If the search bar contains text, filter our data with the string
        if let searchText = searchController.searchBar.text {
            filterContent(for: searchText)
            // Reload the table view with the search result data.
            tableView.reloadData()
        }
    }

    // MARK: - UITableViewController methods

    override func numberOfSections(in tableView: UITableView) -> Int { return 1 }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        // If the search bar is active, use the searchResults data.
        return searchController.isActive ? searchResults.count : entries.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        // If the search bar is active, use the searchResults data.
        let entry = searchController.isActive ?
```

```
                searchResults[indexPath.row] : entries[indexPath.row]

        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
        cell.textLabel?.text = entry.title
        cell.imageView?.image = UIImage(named: entry.image)
        return cell
    }
}
```

# Section 39.3: Implementation

First, make your class comply with the UISearchResultsUpdating protocol.

```
class MyTableViewController: UITableViewController, UISearchResultsUpdating {}
```

Add the search controller property:

```
class MyTableViewController: UTableViewController, UISearchResultsUpdating {
    let searchController = UISearchController(searchResultsController: nil)
}
```

Add the search bar:

```
override func viewDidLoad() {
    super.viewDidLoad()

    searchController.searchResultsUpdater = self
    searchController.hidesNavigationBarDuringPresentation = false
    searchController.dimsBackgroundDuringPresentation = false
    searchController.searchBar.sizeToFit()
    self.tableView.tableHeaderView = searchController.searchBar
}
```

And finally, implement the updateSearchResultsForSearchController method that comes from the UISearchResultsUpdating protocol:

```
func updateSearchResultsForSearchController(searchController: UISearchController) {

}
```

# Section 39.4: UISerachController in Objective-C

```
Delegate: UISearchBarDelegate, UISearchControllerDelegate, UISearchBarDelegate

@property (strong, nonatomic)  UISearchController *searchController;

- (void)searchBarConfiguration
{
    self.searchController = [[UISearchController alloc] initWithSearchResultsController:nil];
    self.searchController.searchBar.delegate = self;
    self.searchController.hidesNavigationBarDuringPresentation = NO;

    // Hides search bar initially.  When the user pulls down on the list, the search bar is
revealed.
    [self.tableView setContentOffset:CGPointMake(0,
self.searchController.searchBar.frame.size.height)];

    self.searchController.searchBar.backgroundColor = [UIColor DarkBlue];
```

```objc
    self.searchController.searchBar.tintColor = [UIColor DarkBlue];

    self.tableView.contentOffset = CGPointMake(0,
CGRectGetHeight(_searchController.searchBar.frame));
    self.tableView.tableHeaderView = _searchController.searchBar;
    _searchController.searchBar.delegate = self;
    _searchController.searchBar.showsCancelButton = YES;
    self.tapGestureRecognizer = [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(resetSearchbarAndTableView)];
    [self.view addGestureRecognizer:self.tapGestureRecognizer];

}

- (void)resetSearchbarAndTableView{
// Reload your tableview and resign keyboard.
}


- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar{
// Search cancelled
}
- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar{
// Implement filtration of your data as per your need using NSPredicate or else.
// then reload your data control like Tableview.
}
```

# Chapter 40: UITabBarController

## Section 40.1: Create an instance

A 'tab bar' is commonly found in most iOS apps and is used to present distinct views in each tab.

To create a tab bar controller using the interface builder, drag a tab bar Controller from the Object Library into the canvas.



By default a tab bar controller comes with two views. To add additional views, control drag from the tab bar controller to the new view and select 'view controllers' in the segue-drop down.
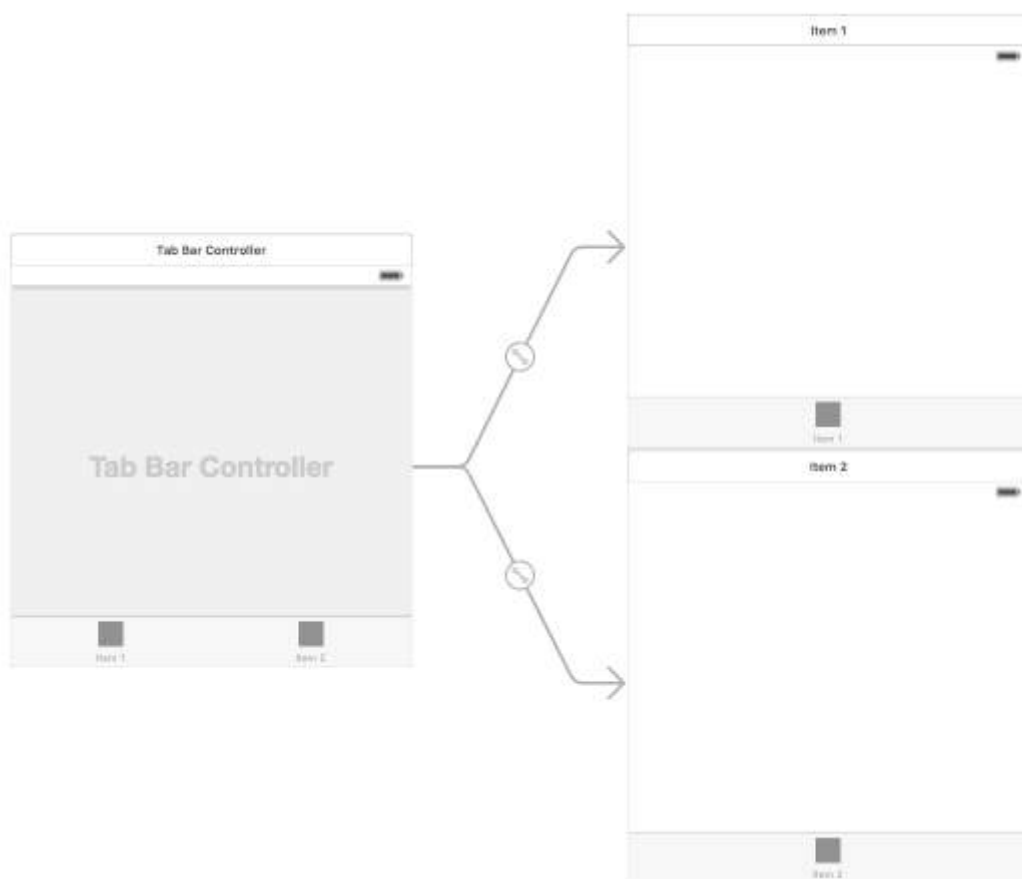


## Section 40.2: Navigation Controller with TabBar

Navigation controller can be embed in each tabs using storyboard it self. It can be like in the screenshot added.

To add a Navigation Controller to a View Controller connecting from Tab Bar Controller, here are the flow

- Select the view controller for which we need to add navigation controller. Here let it be the Search View Controller as the selection display.
- From the **Editor** menu of the Xcode, select **Embed In -> Navigation Controller** option
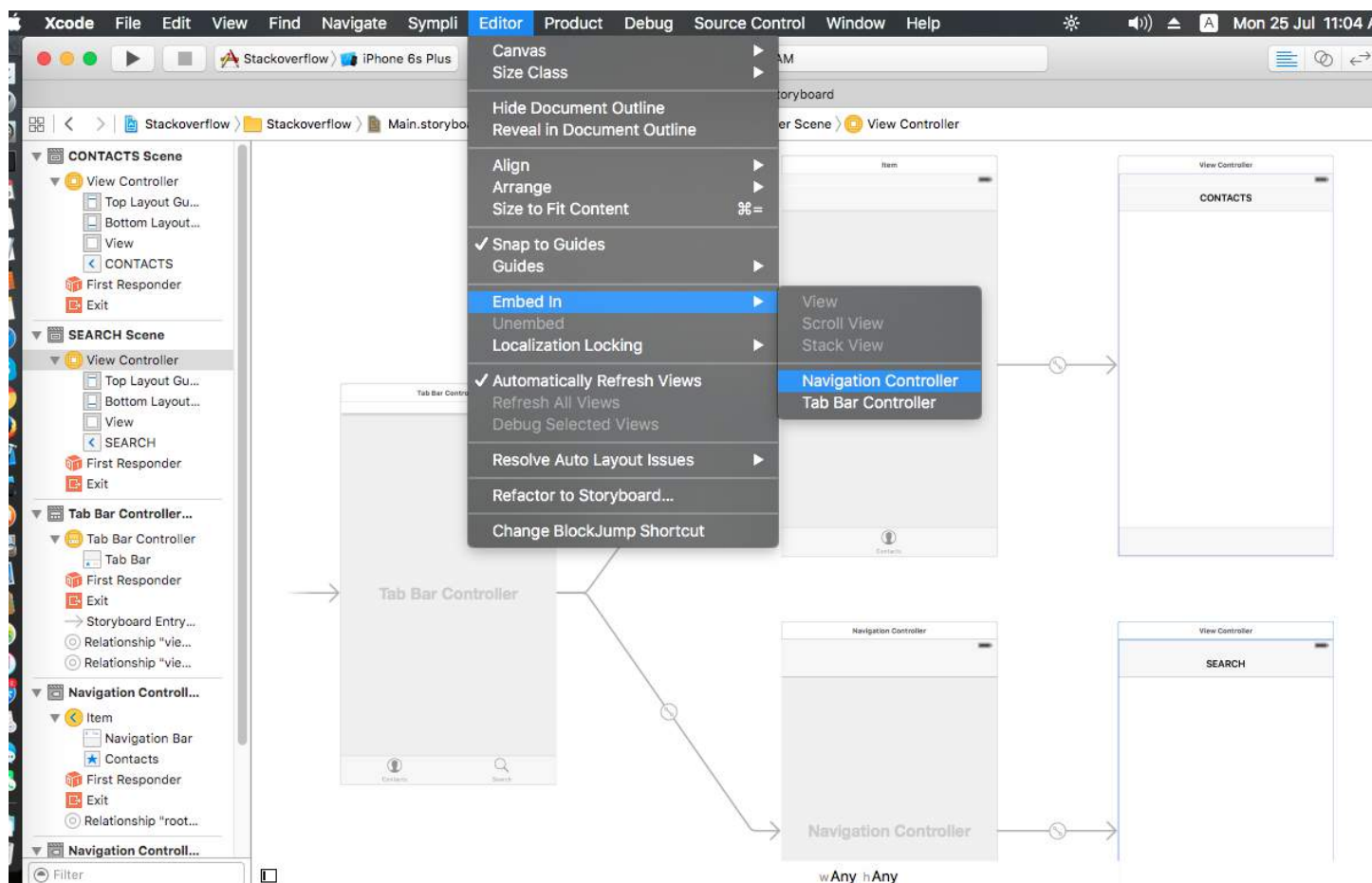
# Section 40.3: Tab Bar color customizing

```objectivec
[[UITabBar appearance] setTintColor:[UIColor whiteColor]];
[[UITabBar appearance] setBarTintColor:[UIColor tabBarBackgroundColor]];
[[UITabBar appearance] setBackgroundColor:[UIColor tabBarInactiveColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor appBlueColor]];
[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarStyle:UIBarStyleBlack];
```

# Section 40.4: Changing Tab Bar Item Title and Icon

**Using the Story Board:**

Select the tab bar item from the corresponding view controller and go to the attributes inspector

If you want a built-in icon and title, set the 'System Item' to the corresponding value.

For a custom icon, add the required images to the assets folder and set the 'System Item' from earlier to 'custom'.

Now, set the icon to be shown when the tab is selected from the 'selected image' drop down and the default tab icon from the 'image' drop down. Add the corresponding title in the 'title' field.
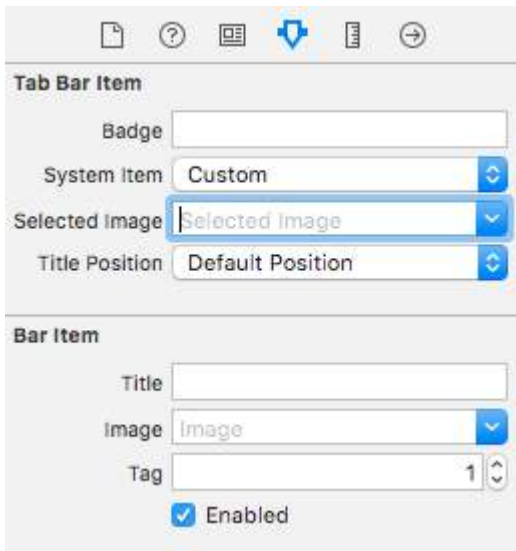
**Programmatically:**

In the `viewDidLoad()` method of the view controller, add the following code:

**Objective-C:**

```objectivec
self.title = @"item";

self.tabBarItem.image = [UIImage imageNamed:@"item"];
self.tabBarItem.selectedImage = [UIImage imageNamed:@"item_selected"];
```

**Swift:**

```swift
self.title = "item"
self.tabBarItem.image = UIImage(named: "item")
self.tabBarItem.selectedImage = UIImage(named: "item_selected")
```

## Section 40.5: Create Tab Bar controller programmatically without Storyboard

```swift
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    var firstTabNavigationController : UINavigationController!
    var secondTabNavigationController : UINavigationController!
    var thirdTabNavigationController : UINavigationController!
    var fourthTabNavigationController : UINavigationController!
    var fifthTabNavigationController : UINavigationController!


    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        Fabric.with([Crashlytics.self])

        window = UIWindow(frame: UIScreen.main.bounds)


        window?.backgroundColor = UIColor.black


        let tabBarController = UITabBarController()
```

```
        firstTabNavigationController = UINavigationController.init(rootViewController:
FirstViewController())
        secondTabNavigationControoller = UINavigationController.init(rootViewController:
SecondViewController())
        thirdTabNavigationController = UINavigationController.init(rootViewController:
ThirdViewController())
        fourthTabNavigationControoller = UINavigationController.init(rootViewController:
FourthViewController())
        fifthTabNavigationController = UINavigationController.init(rootViewController:
FifthViewController())

        tabBarController.viewControllers = [firstTabNavigationController,
secondTabNavigationControoller, thirdTabNavigationController, fourthTabNavigationControoller,
fifthTabNavigationController]


        let item1 = UITabBarItem(title: "Home", image: UIImage(named: "ico-home"), tag: 0)
        let item2 = UITabBarItem(title: "Contest", image: UIImage(named: "ico-contest"), tag: 1)
        let item3 = UITabBarItem(title: "Post a Picture", image: UIImage(named: "ico-photo"), tag:
2)
        let item4 = UITabBarItem(title: "Prizes", image: UIImage(named: "ico-prizes"), tag: 3)
        let item5 = UITabBarItem(title: "Profile", image: UIImage(named: "ico-profile"), tag: 4)

        firstTabNavigationController.tabBarItem = item1
        secondTabNavigationControoller.tabBarItem = item2
        thirdTabNavigationController.tabBarItem = item3
        fourthTabNavigationControoller.tabBarItem = item4
        fifthTabNavigationController.tabBarItem = item5

        UITabBar.appearance().tintColor = UIColor(red: 0/255.0, green: 146/255.0, blue: 248/255.0,
alpha: 1.0)

        self.window?.rootViewController = tabBarController

        window?.makeKeyAndVisible()

        return true
    }
```

# Section 40.6: UITabBarController with custom color selection

UITabBarController building in `Swift 3` Change image color and title according to selection with changing selected tab color.

```
import UIKit

class TabbarController: UITabBarController {

    override func viewDidLoad() {
        super.viewDidLoad()

        self.navigationController?.isNavigationBarHidden = true

        UITabBar.appearance().tintColor = UIColor.purple

        // set red as selected background color
        let numberOfItems = CGFloat(tabBar.items!.count)
        let tabBarItemSize = CGSize(width: tabBar.frame.width / numberOfItems, height:
tabBar.frame.height)
        tabBar.selectionIndicatorImage =
```

```
UIImage.imageWithColor(UIColor.lightText.withAlphaComponent(0.5), size:
tabBarItemSize).resizableImage(withCapInsets: UIEdgeInsets.zero)

        // remove default border
        tabBar.frame.size.width = self.view.frame.width + 4
        tabBar.frame.origin.x = -2

    }

    override func viewWillAppear(_ animated: Bool) {
        // For Images
        let firstViewController:UIViewController = NotificationVC()
        // The following statement is what you need
        let customTabBarItem:UITabBarItem = UITabBarItem(title: nil, image: UIImage(named:
"notification@2x")?.withRenderingMode(UIImageRenderingMode.alwaysOriginal), selectedImage:
UIImage(named: "notification_sel@2x"))
        firstViewController.tabBarItem = customTabBarItem

        for item in self.tabBar.items! {
            let unselectedItem = [NSForegroundColorAttributeName: UIColor.white]
            let selectedItem = [NSForegroundColorAttributeName: UIColor.purple]

            item.setTitleTextAttributes(unselectedItem, for: .normal)
            item.setTitleTextAttributes(selectedItem, for: .selected)
        }
    }

}


extension UIImage {
    class func imageWithColor(_ color: UIColor, size: CGSize) -> UIImage {
        let rect: CGRect = CGRect(origin: CGPoint(x: 0,y :0), size: CGSize(width: size.width,
height: size.height))
        UIGraphicsBeginImageContextWithOptions(size, false, 0)
        color.setFill()
        UIRectFill(rect)
        let image: UIImage = UIGraphicsGetImageFromCurrentImageContext()!
        UIGraphicsEndImageContext()
        return image
    }

}
```

**Choosing image for tab bar and set the tab title here**
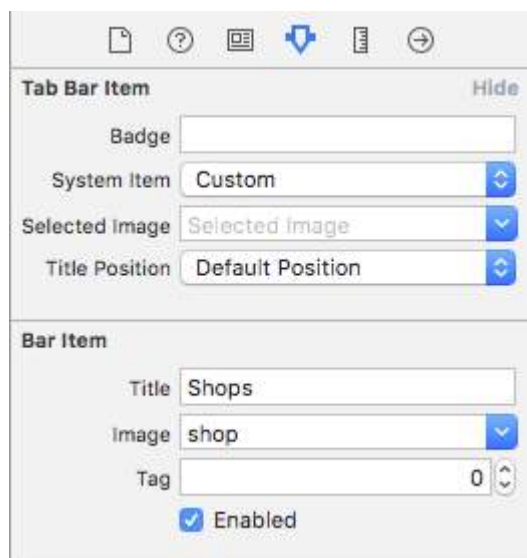
**Selection another tab**

# Chapter 41: UIWebView

## Section 41.1: Create a UIWebView instance

**Swift**

```
let webview = UIWebView(frame: CGRect(x: 0, y: 0, width: 320, height: 480))
```

**Objective-C**

```
UIWebView *webview = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

//Alternative way of defining frame for UIWebView
UIWebView *webview = [[UIWebView alloc] init];
CGRect webviewFrame = webview.frame;
webviewFrame.size.width = 320;
webviewFrame.size.height = 480;
webviewFrame.origin.x = 0;
webviewFrame.origin.y = 0;
webview.frame = webviewFrame;
```

## Section 41.2: Determining content size

In many cases, for instance when using web views in table view cells, it's important to determine the content size of the rendered HTML page. After loading the page, this can be calculated in the `UIWebViewDelegate` delegate method:

```
- (void) webViewDidFinishLoad:(UIWebView *) aWebView {
    CGRect frame = aWebView.frame;
    frame.size.height = 1;
    aWebView.frame = frame;
    CGSize fittingSize = [aWebView sizeThatFits:CGSizeZero];
    frame.size = fittingSize;
    aWebView.frame = frame;

    NSLog(@"size: %f, %f", fittingSize.width, fittingSize.height);
}
```

The code employs an additional trick of shortly setting the height of the web view to 1 prior to measuring the fitting size. Otherwise it would simply report the current frame size. After measuring we immediately set the height to the actual content height.

Source

## Section 41.3: Load HTML string

Web views are useful to load locally generated HTML strings.

```
NSString *html = @"<!DOCTYPE html><html><body>Hello World</body></html>";
[webView loadHTMLString:html baseURL:nil];
```

**Swift**

```
  let htmlString = "<h1>My First Heading</h1><p>My first paragraph.</p>"
  webView.loadHTMLString(htmlString, baseURL: nil)
```

A local base URL may be specified. This is useful to reference images, stylesheets or scripts from the app bundle:

```
NSString *html = @"<!DOCTYPE html><html><head><link href='style.css' rel='stylesheet'
type='text/css'></head><body>Hello World</body></html>";
[self loadHTMLString:html baseURL:[NSURL fileURLWithPath:[[NSBundle mainBundle] resourcePath]]];
```

In this case, `style.css` is loaded locally from the app's resource directory. Of course it's also possible to specify a remote URL.

## Section 41.4: Making a URL request

Load content in webview from the `url`

**Swift**

```
webview.loadRequest(NSURLRequest(URL: NSURL(string: "http://www.google.com")!))
```

**Objective-C**

```
[webview loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:@"http://www.google.com"]]];
```

## Section 41.5: Load JavaScript

We can run custom JavaScript on a `UIWebView` using the method `stringByEvaluatingJavaScriptFromString()`. This method returns the result of running the JavaScript script passed in the script parameter, or nil if the script fails.

**Swift**

**Load script from String**

```
webview.stringByEvaluatingJavaScriptFromString("alert('This is JavaScript!');")
```

**Load script from Local file**

```
//Suppose you have javascript file named "JavaScript.js" in project.
let filePath = NSBundle.mainBundle().pathForResource("JavaScript", ofType: "js")
        do {
            let jsContent = try String.init(contentsOfFile: filePath!, encoding:
NSUTF8StringEncoding)
            webview.stringByEvaluatingJavaScriptFromString(jsContent)
        }
        catch let error as NSError{
            print(error.debugDescription)
        }
```

**Objective-C**

**Load script from String**

```
[webview stringByEvaluatingJavaScriptFromString:@"alert('This is JavaScript!');"];
```

**Load script from Local file**

```
//Suppose you have javascript file named "JavaScript.js" in project.
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"JavaScript" ofType:@"js"];
```

```
NSString *jsContent = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding
error:nil];
[webview stringByEvaluatingJavaScriptFromString:jsContent];
```

**Note** The `stringByEvaluatingJavaScriptFromString:` method waits synchronously for JavaScript evaluation to complete. If you load web content whose JavaScript code you have not vetted, invoking this method could hang your app. Best practice is to adopt the `WKWebView` class and use its `evaluateJavaScript:completionHandler:` method instead. But `WKWebView` is available from iOS 8.0 and later.

# Section 41.6: Stop Loading Web Content

Method `stopLoading()` stops the current loading process of the webview.

**Swift**

```
webview.stopLoading()
```

**Objective-C**

```
[webview stopLoading];
```

# Section 41.7: Reload Current Web Content

**Swift**

```
webview.reload()
```

**Objective-C**

```
[webview reload];
```

# Section 41.8: Load Document files like .pdf, .txt, .doc etc

Instead of web pages, we can also load the document files into iOS WebView like .pdf, .txt, .doc etc.. `loadData` method is used to load `NSData` into webview.

**Swift**

```
//Assuming there is a text file in the project named "home.txt".
let localFilePath = NSBundle.mainBundle().pathForResource("home", ofType:"txt");
let data = NSFileManager.defaultManager().contentsAtPath(localFilePath!);
webview.loadData(data!, MIMEType: "application/txt", textEncodingName:"UTF-8" , baseURL: NSURL())
```

**Objective-C**

```
//Assuming there is a text file in the project named "home.txt".
NSString *localFilePath = [[NSBundle mainBundle] pathForResource:@"home" ofType:@"txt"];
NSData *data = [[NSFileManager defaultManager] contentsAtPath:localFilePath];
[webview loadData:data MIMEType:@"application/txt" textEncodingName:@"UTF-8" baseURL:[NSURL new]];
```

# Section 41.9: Load local HTML file in webView

First, add the HTML File to your Project (If you are asked to choose options for adding the file, select *Copy items if*

*needed*)

**The following line of code loads the content of the HTML file into the webView**

```
webView.loadRequest(NSURLRequest(URL: NSURL(fileURLWithPath:
NSBundle.mainBundle().pathForResource("YOUR HTML FILE", ofType: "html")!}))
```

- If your HTML file is called index.html replace **YOUR HTML FILE** with **index**
- You can use this code either in *viewDidLoad()* or *viewDidAppear()* or any other function

# Section 41.10: Make links That inside UIWebview clickable
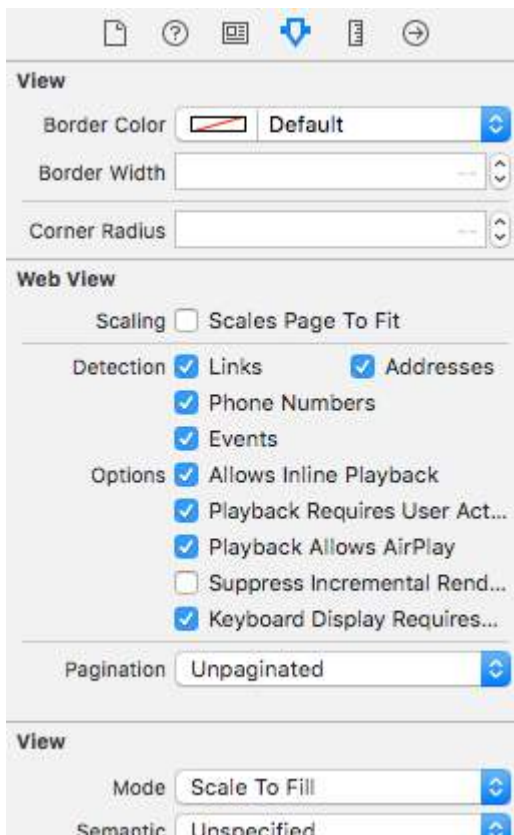


In vc.h

```
@interface vc : UIViewController<UIWebViewDelegate>
```

in vc.m

```
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)request
navigationType:(UIWebViewNavigationType)navigationType{


    if (navigationType == UIWebViewNavigationTypeLinkClicked){
        //open it on browser if you want to open it in same web view remove return NO;
        NSURL *url = request.URL;
        if ([[UIApplication sharedApplication] canOpenURL:url]) {
            [[UIApplication sharedApplication] openURL:url];
        }
        return NO;

    }

    return YES;
```

```
}
```

# Chapter 42: UIActivityViewController

| Parameter Name | Description |
| --- | --- |
| activityItems | Contains array of object to perform the activity. This array must not be nil and must contain at least one object. |
| applicationActivities | An array of UIActivity objects representing the custom services that your application supports. This parameter can be nil. |

## Section 42.1: Initializing the Activity View Controller

**Objective-C**

```
NSString *textToShare = @"StackOverflow Documentation!! Together, we can do for Documentation what
we did for Q&A.";
NSURL *documentationURL = [NSURL URLWithString:@"http://stackoverflow.com/tour/documentation"];

NSArray *objectsToShare = @[textToShare, documentationURL];

UIActivityViewController *activityVC = [[UIActivityViewController alloc]
initWithActivityItems:objectsToShare applicationActivities:nil];

[self presentViewController:activityVC animated:YES completion:nil];
```

**Swift**

```
let textToShare = "StackOverflow Documentation!! Together, we can do for Documentation what we did
for Q&A."
let documentationURL = NSURL(string:"http://stackoverflow.com/tour/documentation")

let objToShare : [AnyObject] = [textToShare, documentationURL!]

let activityVC = UIActivityViewController(activityItems: objToShare, applicationActivities: nil)
self.presentViewController(activityVC, animated: true, completion: nil)
```

# Chapter 43: UIControl - Event Handling with Blocks

## Section 43.1: Introduction

Typically, when using `UIControl` or `UIButton`, we add a `selector` as a callback action for when an event occurs on a button or control, such as the user pressing the button or touching the control.

For example, we would do the following:

```swift
import UIKit

class ViewController: UIViewController {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(self.onButtonPress(_:)), for: .touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func onButtonPress(_ button: UIButton!) {
        print("PRESSED")
    }
}
```

When it comes to `selector`, the compiler only needs to know that it exists.. This can be done through a `protocol` and not be implemented.

For example, the following would crash your application:

```swift
import UIKit

@objc
protocol ButtonEvent {
    @objc optional func onButtonPress(_ button: UIButton)
}

class ViewController: UIViewController, ButtonEvent {
    @IBOutlet weak var button: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))
        button.addTarget(self, action: #selector(ButtonEvent.onButtonPress(_:)), for:
.touchUpInside)
        self.view.addSubview(button)
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
```

```
    }
}
```

This is because your application does NOT implement the `onButtonPress` function.

Now what if you could do all of this alongside the initialization of the button? What if you didn't have to specify callbacks and could instead specify blocks that can be added and removed at any time? Why worry about implementing selectors?

**Solution**

```swift
import Foundation
import UIKit

protocol RemovableTarget {
    func enable();
    func disable();
}

extension UIControl {
    func addEventHandler(event: UIControlEvents, runnable: (control: UIControl) -> Void) ->
RemovableTarget {

        class Target : RemovableTarget {
            private var event: UIControlEvents
            private weak var control: UIControl?
            private var runnable: (control: UIControl) -> Void

            private init(event: UIControlEvents, control: UIControl, runnable: (control: UIControl)
-> Void) {
                self.event = event
                self.control = control
                self.runnable = runnable
            }

            @objc
            private func run(_ control: UIControl) {
                runnable(control: control)
            }

            private func enable() {
                control?.addTarget(self, action: #selector(Target.run(_:)), for: event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), self,
.OBJC_ASSOCIATION_RETAIN)
            }

            private func disable() {
                control?.removeTarget(self, action: #selector(Target.run(_:)), for: self.event)
                objc_setAssociatedObject(self, unsafeAddress(of: self), nil,
.OBJC_ASSOCIATION_ASSIGN)
            }
        }

        let target = Target(event: event, control: self, runnable: runnable)
        target.enable()
        return target
    }
}
```

The above is a simple extension on `UIControl`. It adds an inner private class that has a callback `func run(_`

`control: UIControl`) that is used as the events' action.

Next we use `object association` to add and remove the target because it will not be retained by the `UIControl`.

The event handler function returns a `Protocol` in order to hide the inner workings of the `Target` class but also to allow you to `enable` and `disable` the target at any given time.

**Usage Example:**

```
import Foundation
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()


        //Create a button.
        let button = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 44))

        //Add an event action block/listener -- Handles Button Press.
        let target = button.addEventHandler(event: .touchUpInside) { (control) in
            print("Pressed")
        }


        self.view.addSubview(button)

        //Example of enabling/disabling the listener/event-action-block.
        DispatchQueue.main.after(when: DispatchTime.now() + 5) {
            target.disable() //Disable the listener.

            DispatchQueue.main.after(when: DispatchTime.now() + 5) {
                target.enable() //Enable the listener.
            }
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

# Chapter 44: UISplitViewController

## Section 44.1: Master and Detail View interaction using Delegates in Objective C

`UISplitViewController` must be the rootViewController of your application.

**AppDelegate.m**

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
// Override point for customization after application launch.
self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
self.window.backgroundColor = [UIColor blackColor];
[self.window makeKeyAndVisible];
self.window.clipsToBounds = YES;
SplitViewController *spView = [[SplitViewController alloc]init];
self.window.rootViewController = spView;
[self.window makeKeyAndVisible];
return YES;
}
```

Just create an object for the `UISplitViewController` and set that viewcontroller as the rootviewcontroller for your application.

**SplitViewController.h**

```
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
DetailViewController *detailVC;
MasterViewController *masterVC;
NSMutableArray *array;
}
@end
```

`MasterViewController` is always on the left side of the device you can set the width in `UISplitViewCOntroller` delegate methods and `DetailViewController` is on the Right side of the application

**SplitViewController.m**

```
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
[super viewDidLoad];
masterVC = [[MasterViewController alloc]init];
detailVC = [[DetailViewController alloc]init];
[masterVC setDetailDelegate:(id)detailVC];
NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
```

```
self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
self.viewControllers = vcArray;
self.delegate = (id)self;
self.presentsWithGesture = YES;
}
```

Created master and detail ViewControllers are added to an array which is set to `self.viewControllers` in `UISplitViewController`. `self.preferredDisplayMode` is is the mode set for displaying of master and DetailViewController [Apple Documentation for DisplayMode](). `self.presentsWithGesture` enables swipe gesture for displaying `MasterViewcontroller`

**MasterViewController.h**

```
#import <UIKit/UIKit.h>

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end
```

Create a `DetailViewDelegate` delegate with sendSelectedNavController:(`UIViewController *`)viewController method for sending the `UIViewController` to the `DetailViewcontroller`. Then in `MasterViewController` the mainTableView is the tableview in the leftside. The viewControllerArray contains all the `UIViewControllers` that needs to be displayed in `DetailViewController`

**MasterViewController.m**

```
#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc]init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc]init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc]init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc]init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc]init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray
alloc]initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc]initWithFrame:CGRectMake(0, 50,self.view.frame.size.width,
self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];
```

```
[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}


- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:    (NSInteger)section
{
    return [viewControllerArray count];
}


- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;    //count of section
}


- (UITableViewCell *)tableView:(UITableView *)tableView
     cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView   dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
   cell = [[UITableViewCell alloc]initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
[cell.contentView setBackgroundColor:[UIColor redColor]];
cell.textLabel.text =[NSString stringWithFormat:@"My VC at index %ld",(long)indexPath.row];
return cell;
}


- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end
```

Create some `UIViewControllers` and added it to an array. The Table view is initialized then on
didSelectRowAtIndexPath method I send a `UIViewController` to the `DetailViewController` using detailDelegate
with the corresponding `UIViewController` in array as parameter

**DetailViewController.h**

```
#import <UIKit/UIKit.h>


@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController *tempNav;
}
@end
```

**DetailViewController.m**

```objc
#import "DetailViewController.h"

@implementation DetailViewController
-(void)viewDidLoad
{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view addSubview:tempNav.view];
}
@end
```

sendSelectedNavController is declared here with removing all the views in the `DetailViewController` and adding the passed `UIViewController` from the `MasterViewController`

Adding some screen shots of the application
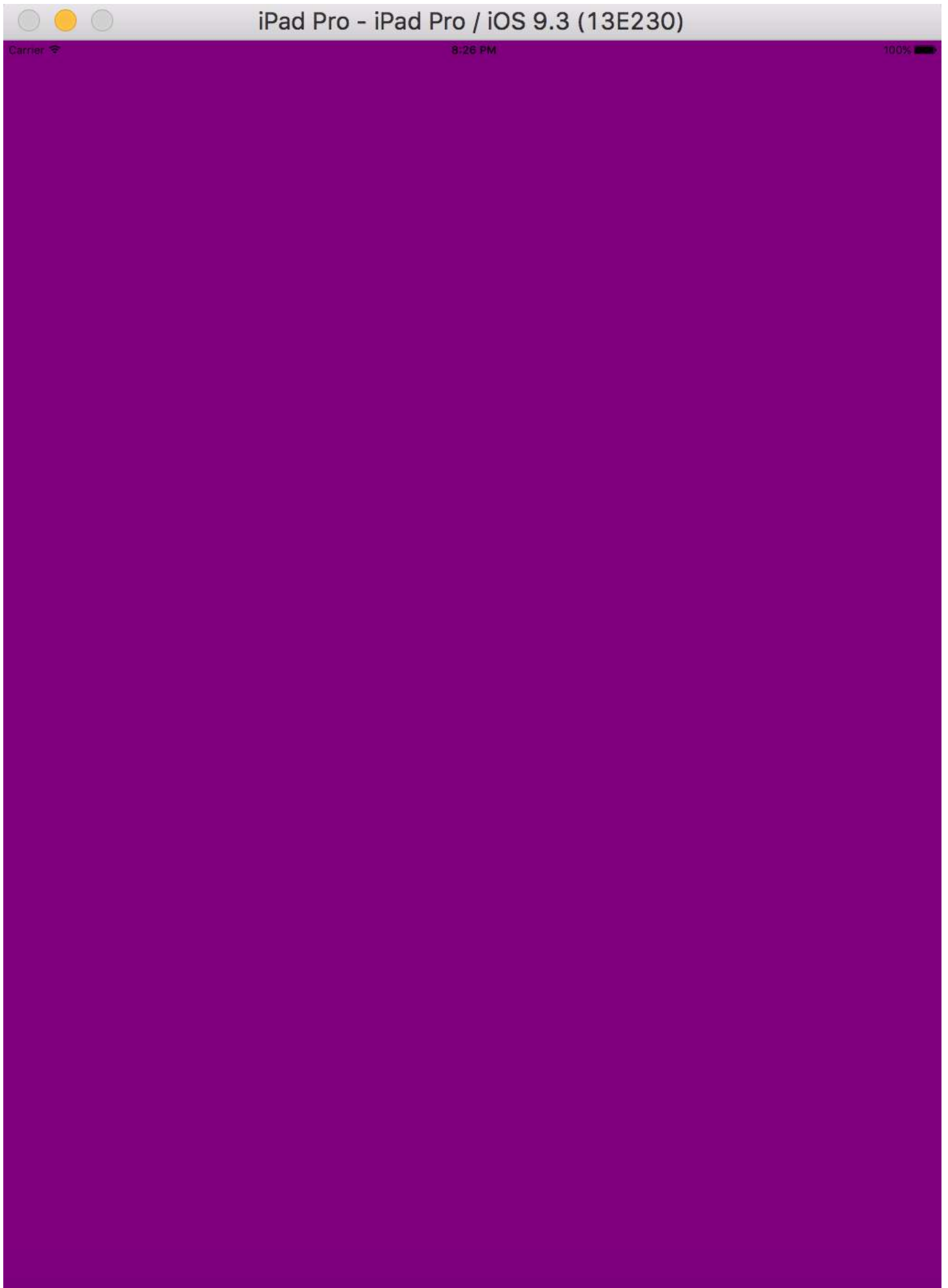
On launching the application we don't get `MasterViewController` since we gave the `preferredDisplayMode` as

automatic on swiping the screen we get the `MasterViewController` as attached in the below image but in Landscape mode we get both the `MasterViewController` and `DetailViewController`

on Landscape Orientiation
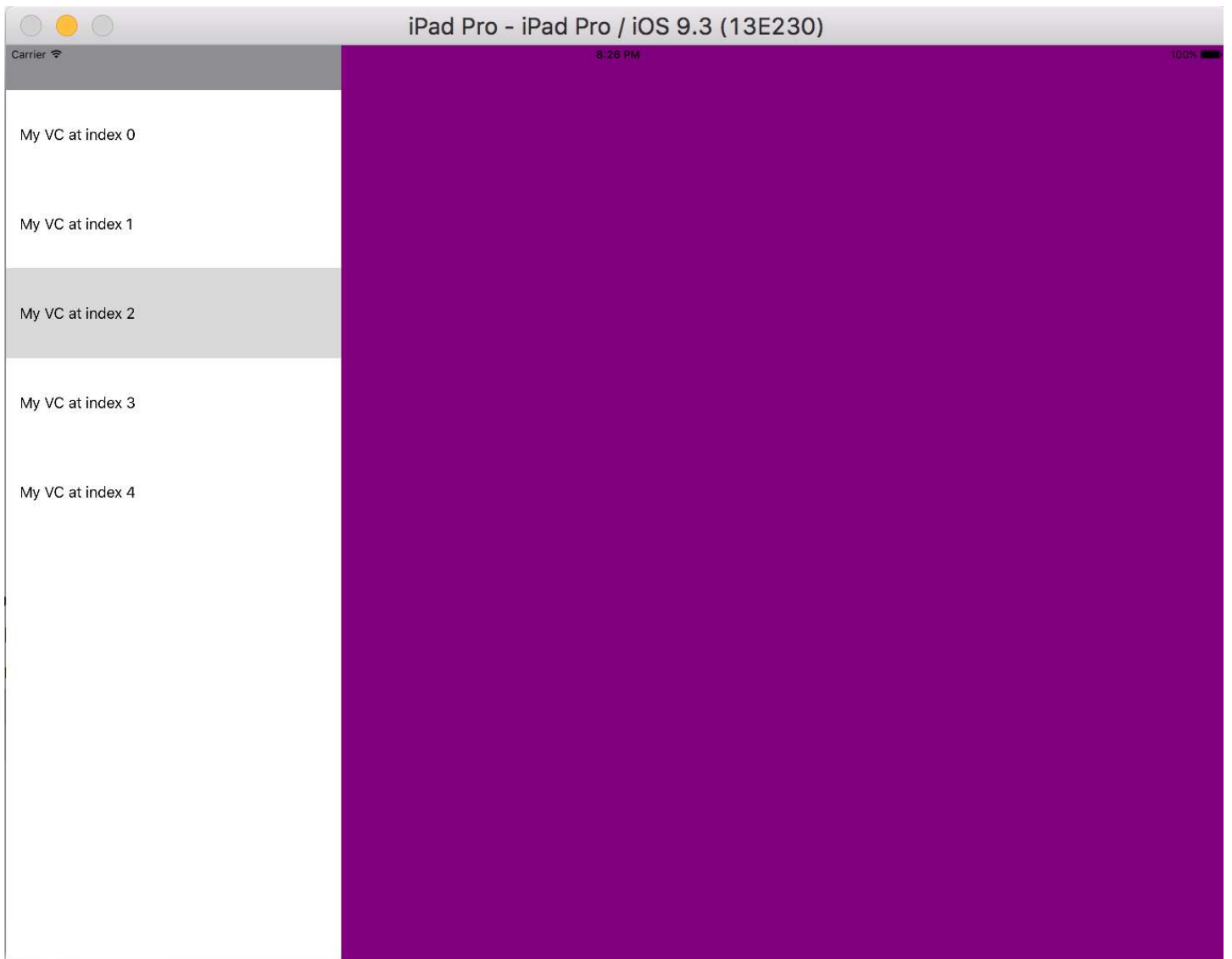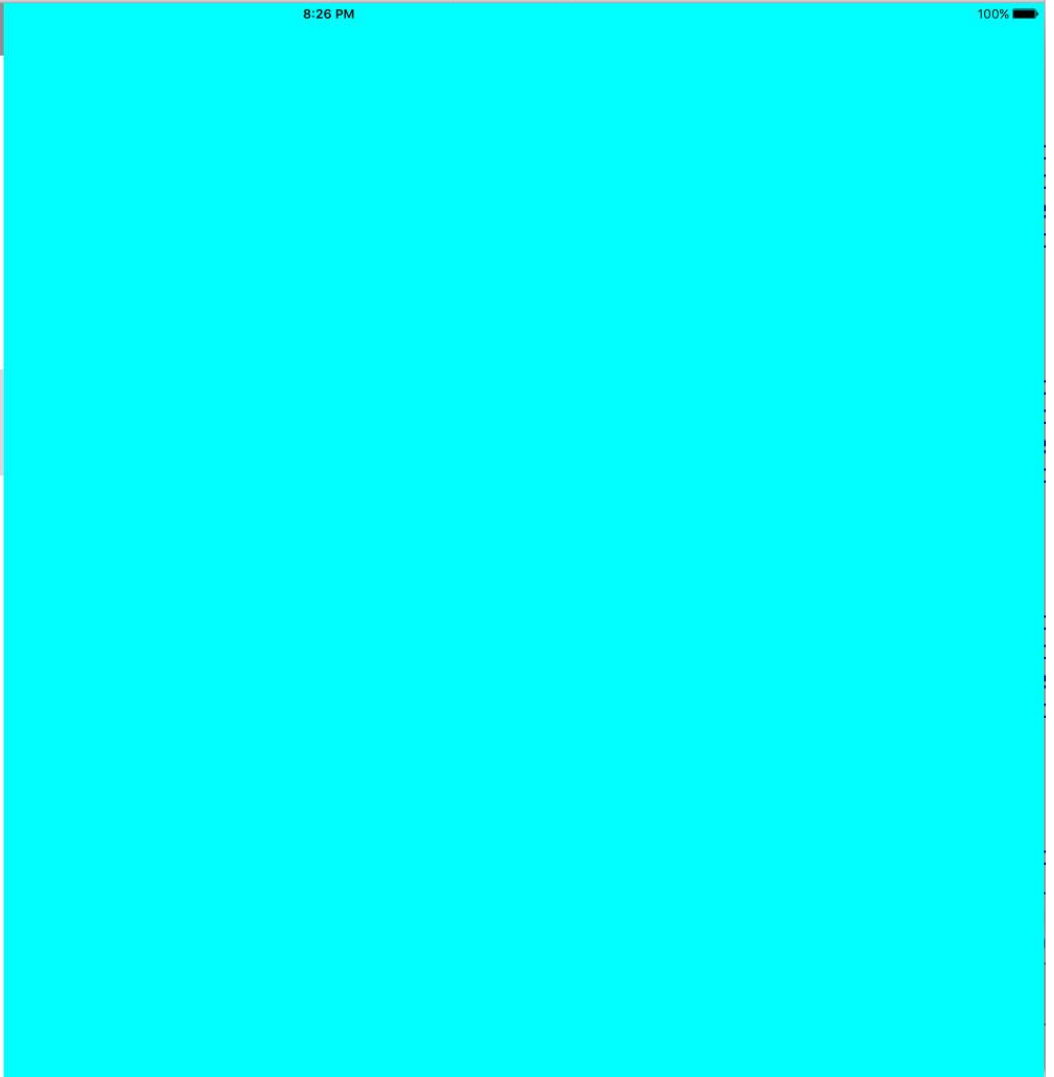
# Chapter 45: UISlider

## Section 45.1: UISlider

**Objective-C**

Declare a slider property in the `ViewController.h` or in the interface of `ViewController.m`

```objc
@property (strong, nonatomic)UISlider *slider;

//Define frame of slider and add to view
CGRect frame = CGRectMake(0.0, 100.0, 320.0, 10.0);
UISlider *slider = [[UISlider alloc] initWithFrame:frame];
[slider addTarget:self action:@selector(sliderAction:)
forControlEvents:UIControlEventValueChanged];
[self.slider setBackgroundColor:[UIColor clearColor]];
self.slider.minimumValue = 0.0;
self.slider.maximumValue = 50.0;
//sending a NO/False would update the value of slider only when the user is no longer touching the
screen. Hence sending only the final value
self.slider.continuous = YES;
self.slider.value = 25.0;
[self.view addSubview slider];
```

Handle the slider change event

```objc
- (IBAction)sliderAction:(id)sender {
    NSLog(@"Slider Value %f", sender.value);
}
```

## Section 45.2: SWIFT Example

```swift
let frame = CGRect(x: 0, y: 100, width: 320, height: 10)
let slider = UISlider(frame: frame)
slider.addTarget(self, action:  #selector(sliderAction), for: .valueChanged)
slider.backgroundColor = .clear
slider.minimumValue = 0.0
slider.maximumValue = 50.0
//sending a NO/False would update the value of slider only when the user is no longer touching the
screen. Hence sending only the final value
slider.isContinuous = true
slider.value = 25.0
view.addSubview(slider)
```

Handling the slider change event

```swift
func sliderAction(sender:UISlider!)
{
    print("value--\(sender.value)")
}
```

## Section 45.3: Adding a custom thumb image

To add a custom image for the thumb of the slider, simply call the setThumbImage method with your custom
image:

Swift 3.1:

```swift
let slider = UISlider()
let thumbImage = UIImage
slider.setThumbImage(thumbImage, for: .normal)
```

# Chapter 46: UIStoryboard

A UIStoryboard object encapsulates the view controller graph stored in an Interface Builder storyboard resource file. This view controller graph represents the view controllers for all or part of your application's user interface.

## Section 46.1: Getting an instance of UIStoryboard programmatically

**SWIFT:**

Getting an instance of **UIStoryboard** programmatically can be done as follows:

```swift
let storyboard = UIStoryboard(name: "Main", bundle: nil)
```

where:

- **name** => the name of the storyboard without the extension
- **bundle** => the bundle containing the storyboard file and its related resources. If you specify nil, this method looks in the main bundle of the current application.

For example, you can use the instance created above to access a certain **UIViewController** instantiated within that storyboard:

```swift
let viewController = storyboard.instantiateViewController(withIdentifier: "yourIdentifier")
```

**OBJECTIVE-C:**

Getting an instance of **UIStoryboard** in Objective-C can be done as follows:

```objc
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"MainStoryboard" bundle:nil];
```

Example of accessing **UIViewController** instantiated within that storyboard:

```objc
MyViewController *myViewController = [storyboard
instantiateViewControllerWithIdentifier:@"MyViewControllerIdentifier"];
```

## Section 46.2: Open another storyboard

```swift
let storyboard = UIStoryboard(name: "StoryboardName", bundle: nil)
let vc = storyboard.instantiateViewController(withIdentifier: "ViewControllerID") as
YourViewController
self.present(vc, animated: true, completion: nil)
```

# Chapter 47: UIPageViewController

UIPageViewController provides users the ability to easily transition between several views by using a swipe gesture. In order to create a UIPageViewController, you must implement the UIPageViewControllerDataSource methods. These include methods to return both the UIPageViewController before and after the current UIPageViewController along with the presentationCount and presentationIndex methods.

## Section 47.1: Create a horizontal paging UIPageViewController programatically

1. Init array of view controllers which will be managed by UIPageViewController. Add a base view controller class which has property `identifier` which will be used to identify view controllers when working with UIPageViewController data source methods. Let the view controllers to inherit from that base class.

```
UIViewController *firstVC = [[UIViewController alloc] init];
firstVC.identifier = 0
UIViewController *secondVC = [[UIViewController alloc] init];
secondVC.identifier = 1
NSArray *viewControllers = [[NSArray alloc] initWithObjects: firstVC, secondVC, nil];
```

2. Create UIPageViewController instance.

```
UIPageViewController *pageViewController = [[UIPageViewController alloc]
initWithTransitionStyle:UIPageViewControllerTransitionStyleScroll

 navigationOrientation:UIPageViewControllerNavigationOrientationHorizontal

 options:nil];
```

3. Data source is current class which must implement `UIPageViewControllerDataSource` protocol.

```
pageViewController.dataSource = self;
```

4. `setViewControllers` will add only first view controller, next will be added to the stack using data source methods

```
if (viewControllers.count) {
    [pageViewController setViewControllers:@[[viewControllers objectAtIndex:0]]
                            direction:UIPageViewControllerNavigationDirectionForward
                              animated:NO
                            completion:nil];
}
```

5. Add UIPageViewController as a child view controller so it will receive from it's parent view controller appearance and `rotation` events.

```
[self addChildViewController:pageViewController];
pageViewController.view.frame = self.view.frame;
[self.view addSubview:pageViewController.view];
[pageViewController didMoveToParentViewController:self];
```

6. Implementing UIPageViewControllerDataSource methods

```
- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
      viewControllerBeforeViewController:(UIViewController *)viewController
{
    index = [(Your View Controler Base Class *)viewController identifier];
```

```objc
        index--;
        return [self childViewControllerAtIndex:index];
}

- (UIViewController *)pageViewController:(UIPageViewController *)pageViewController
        viewControllerAfterViewController:(UIViewController *)viewController
{
        index = [(Your View Controler Base Class *)viewController identifier];
        index++;
        return [self childViewControllerAtIndex:index];
}

- (NSInteger)presentationCountForPageViewController:(UIPageViewController *)pageViewController
{
        return [viewControllers count];
}

- (NSInteger)presentationIndexForPageViewController:(UIPageViewController *)pageViewController
{
        return index;
}
```

7. Utility method which returns a view controller using an index, if index is out of bounds it returns nil.

```objc
- (UIViewController *)childViewControllerAtIndex:(NSInteger)index
{
        if (index <= ([viewControllers count] - 1)) {
            return [viewControllers objectAtIndex:index];
        } else {
            return nil;
        }
}
```

## Section 47.2: A simple way to create horizontal page view controllers ( infinite pages )

1. Let's create a new project, I'm choosing Single View Application for better demonstration
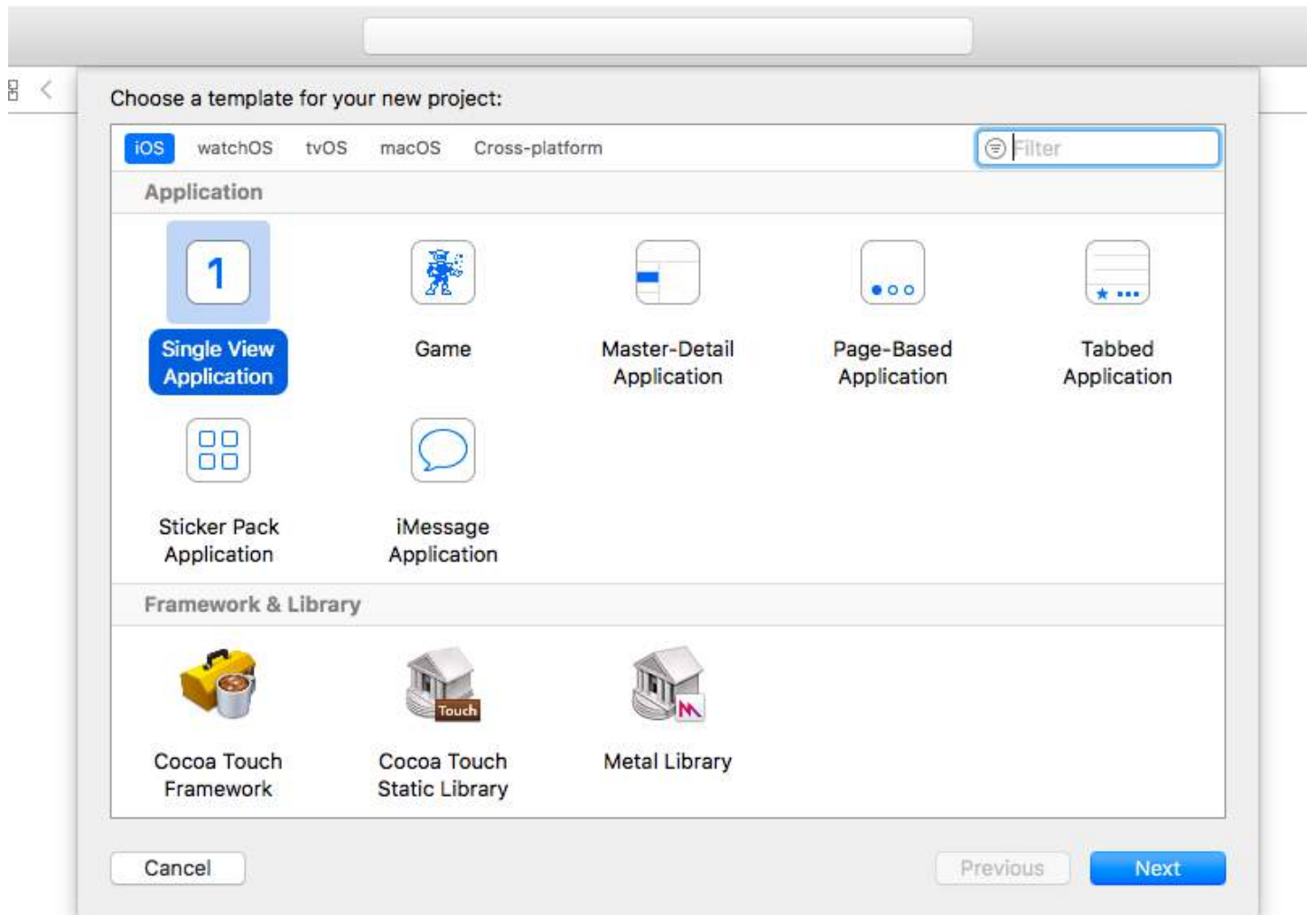
2. Drag a page view controller to the storyboard, there are 2 things you should change after that:
   1. Set the page view controller as initial view controller
   2. Change the transition style to scroll
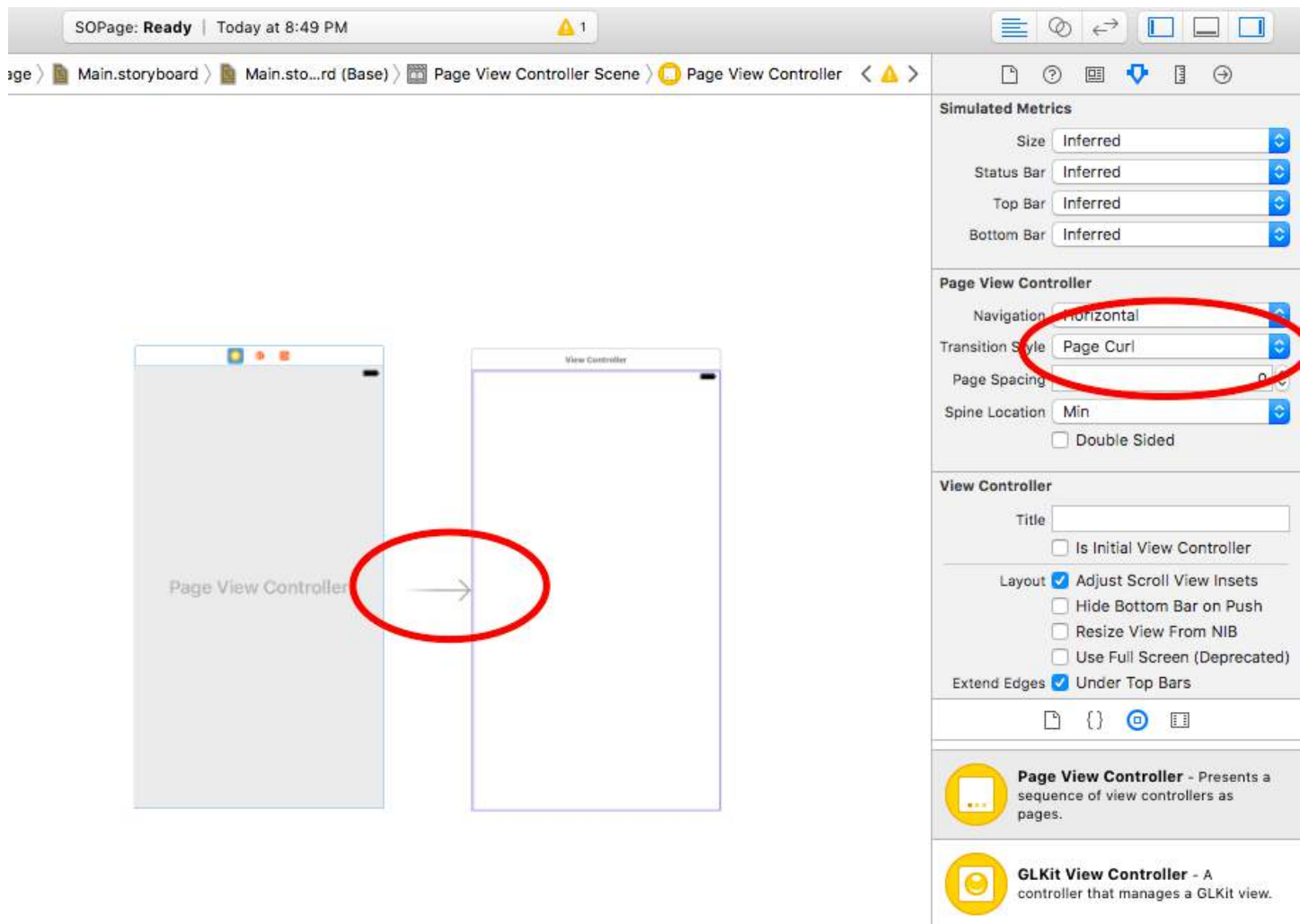
3. And you need to create a UIPageViewController class, then set it as custom class of the page view controller on the storyboard

4. Paste this code into your UIPageViewController class, you should get a colorful infinite paged app :)

```swift
class PageViewController: UIPageViewController, UIPageViewControllerDataSource {

    override func viewDidLoad() {
        self.dataSource = self
        let controller = createViewController()
        self.setViewControllers([controller], direction: .forward, animated: false,
completion: nil)
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore
viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter
viewController: UIViewController) -> UIViewController? {
        let controller = createViewController()
        return controller
    }

    func createViewController() -> UIViewController {
        var randomColor: UIColor {
            return UIColor(hue: CGFloat(arc4random_uniform(360))/360, saturation: 0.5,
```

```
brightness: 0.8, alpha: 1)
        }
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let controller = storyboard.instantiateViewController(withIdentifier: "View
Controller")
        controller.view.backgroundColor = randomColor
        return controller
    }
}
```

This is what the final project looks like, you get a view controller with different color with every scroll:
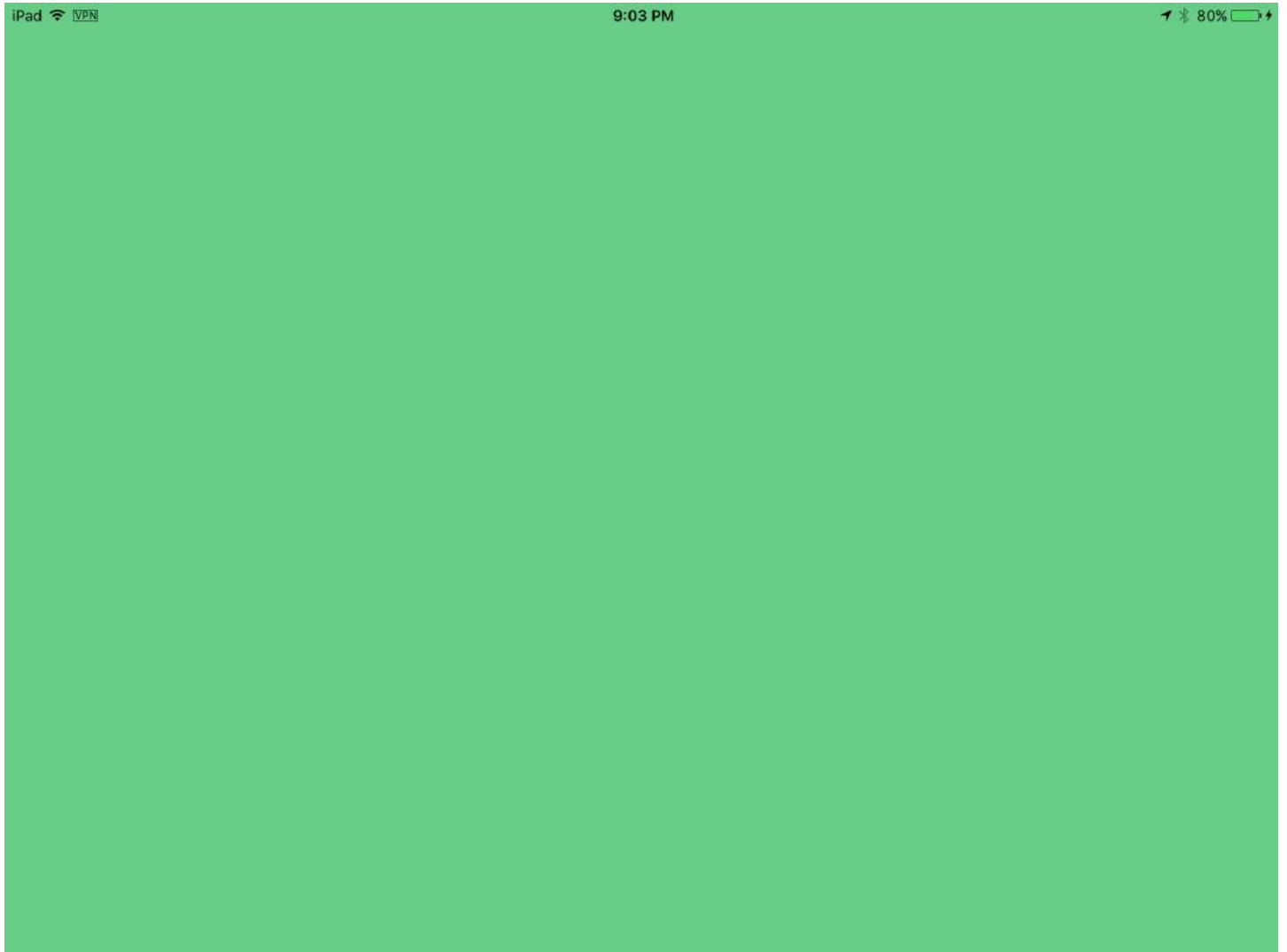
# Chapter 48: UISplitViewController

## Section 48.1: Interacting Between Master and Detail View using Delegates in Objective C

`UISplitViewController` needs to the root view controller of your app's window

**AppDelegate.m**

```objc
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
self.window.backgroundColor = [UIColor blackColor];
[self.window makeKeyAndVisible];
self.window.clipsToBounds = YES;
SplitViewController *spView = [[SplitViewController alloc]init];
self.window.rootViewController = spView;
[self.window makeKeyAndVisible];
return YES;
}
```

Just create an object for your `UISplitVIewController` and set it as the rootViewController for your application.

**SplitViewController.h**

```objc
#import <UIKit/UIKit.h>
#import "MasterViewController.h"
#import "DetailViewController.h"
@interface ViewController : UISplitViewController
{
DetailViewController *detailVC;
MasterViewController *masterVC;
NSMutableArray *array;
}
@end
```

`MasterViewController` is a `UIViewController` that is set on the left side of the device you can set the width in `UISplitViewController` using `maximumPrimaryColumnWidth` and `DetailViewController` is on the Right side

**SplitViewController.m**

```objc
#import "ViewController.h"
#define ANIMATION_LENGTH 0.3
@interface ViewController ()
@end

@implementation ViewController
- (void)viewDidLoad
{
[super viewDidLoad];
masterVC = [[MasterViewController alloc]init];
detailVC = [[DetailViewController alloc]init];
[masterVC setDetailDelegate:(id)detailVC];
NSArray *vcArray = [NSArray arrayWithObjects:masterVC, detailVC, nil];
self.preferredDisplayMode = UISplitViewControllerDisplayModeAutomatic;
self.viewControllers = vcArray;
self.delegate = (id)self;
```

```
self.presentsWithGesture = YES;
}
```

The master and detail `UIViewController`'s are added to an `NSArray` which is set to `self.viewControllers`.
`self.preferredDisplayMode` is is the mode set for displaying of `MasterViewController` and `DetailViewController`
. `self.presentsWithGesture` enables swipe gesture for displaying `MasterViewController`

**MasterViewController.h**

```
#import <UIKit/UIKit.h>

@protocol DetailViewDelegate <NSObject>
@required
- (void)sendSelectedNavController:(UIViewController *)viewController;
@end

@interface MasterViewController : UIViewController
{
    UITableView *mainTableView;
    NSMutableArray *viewControllerArray;
}
@property (nonatomic, retain) id<DetailViewDelegate> detailDelegate;
@end
```

Create a `DetailViewDelegate` Delegate with `sendSelectedNavController` method for sending the
`UIViewControllers` to the `DetailViewController`. Then in `MasterViewController` an `UITableView`is created . The
`ViewControllerArray` contains all the `UIViewControllers` that needs to be displayed in `DetailViewController`

**MasterViewController.m**

```
#import "MasterViewController.h"

@implementation MasterViewController
@synthesize detailDelegate;

-(void)viewDidLoad
{
[super viewDidLoad];

UIViewController *dashBoardVC = [[UIViewController alloc]init];
[dashBoardVC.view setBackgroundColor:[UIColor redColor]];
UIViewController *inventVC = [[UIViewController alloc]init];
[inventVC.view setBackgroundColor:[UIColor whiteColor]];
UIViewController *alarmVC = [[UIViewController alloc]init];
[alarmVC.view setBackgroundColor: [UIColor purpleColor]];
UIViewController *scanDeviceVC = [[UIViewController alloc]init];
[scanDeviceVC.view setBackgroundColor:[UIColor cyanColor]];
UIViewController *serverDetailVC = [[UIViewController alloc]init];
[serverDetailVC.view setBackgroundColor: [UIColor whiteColor]];
viewControllerArray = [[NSMutableArray
alloc]initWithObjects:dashBoardVC,inventVC,alarmVC,scanDeviceVC,serverDetailVC,nil];
mainTableView = [[UITableView alloc]initWithFrame:CGRectMake(0, 50,self.view.frame.size.width,
self.view.frame.size.height-50) style:UITableViewStylePlain];
[mainTableView setDelegate:(id)self];
[mainTableView setDataSource:(id)self];
[mainTableView setSeparatorStyle:UITableViewCellSeparatorStyleNone];
[mainTableView setScrollsToTop:NO];
[self.view addSubview:mainTableView];
}
```

```objc
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    return 100;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:    (NSInteger)section
{
    return [viewControllerArray count];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;    //count of section
}


- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *cellId = [NSString
stringWithFormat:@"Cell%li%ld",(long)indexPath.section,(long)indexPath.row];
UITableViewCell *cell =[tableView   dequeueReusableCellWithIdentifier:cellId];

if (cell == nil)
{
   cell = [[UITableViewCell alloc]initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:cellId];
}
[cell.contentView setBackgroundColor:[UIColor redColor]];
cell.textLabel.text =[NSString stringWithFormat:@"My VC at index %ld",(long)indexPath.row];
return cell;
}


- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    [detailDelegate sendSelectedNavController:[viewControllerArray objectAtIndex:indexPath.row]];
}

@end
```

Created some `UIViewController` and added it to an `NSMutableArray`. The `UITableView` is initialized then on didselectrowatindexpath method I send a `UIViewController` to the `DetailViewController` using detailDelegate delegate with the corresponding `UIViewController` in the `NSMutableArray` as a parameter

**DetailViewController.h**

```objc
#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController<UICollectionViewDelegate>
{
    UIViewController  *tempNav;
}
@end
```

**DetailViewController.m**

```objc
#import "DetailViewController.h"

@implementation DetailViewController
-(void)viewDidLoad
```

```
{
    [super viewDidLoad];
    [self.view setBackgroundColor:[UIColor whiteColor]];
}
-(void)sendSelectedNavController:(UIViewController *)navController
{
    NSArray *viewsToRemove = [self.view subviews];
    for (UIView *v in viewsToRemove) {
        [v removeFromSuperview];
    }
    tempNav = navController;
    [self.view addSubview:tempNav.view];
}
@end
```

The `sendSelectedNavController` is declared here with removing all the `UIView`'s in the `DetailViewController` and adding the passed `UIViewController` from the `MasterViewController`.

# Chapter 49: UIFont

UIFont is a class that is used for getting and setting font-related information. It inherits from NSObject and conforms to Hashable, Equatable, CVarArg and NSCopying.

## Section 49.1: Declaring and initializing UIFont

You can declare a UIFont as follows:

```
var font: UIFont!
```

UIFont has more init() methods:

- UIFont.init(descriptor: UIFontDescriptor, size: CGFloat)
- UIFont.init(name: String, size: CGFloat)

Therefore, you can initialize a UIFont like this:

```
let font = UIFont(name: "Helvetica Neue", size: 15)
```

The default font is System, size 17.

## Section 49.2: Changing the font of a label

To change a label's text font, you need to access its font property:

```
label.font = UIFont(name:"Helvetica Neue", size: 15)
```

The code above will change the font of the label to Helvetica Neue, size 15. Beware that you must spell the font name correctly, otherwise it will throw this error, because the initialized value above is an Optional, and thus can be nil:

> Unexpectedly found nil while unwrapping an Optional value

# Chapter 50: UIDevice

| Property | Description |
|---|---|
| name | The name identifying the device. |
| systemName: String | The name of the operating system running on the device represented by the receiver. |
| model: String | The model of the device. |
| systemVersion: String | The current version of the operating system.. |

## Section 50.1: Get iOS device model name

**Swift 2**

```swift
import UIKit

extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8 where value != 0 else { return identifier }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1":                                 return "iPod Touch 5"
        case "iPod7,1":                                 return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3":     return "iPhone 4"
        case "iPhone4,1":                               return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2":                  return "iPhone 5"
        case "iPhone5,3", "iPhone5,4":                  return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2":                  return "iPhone 5s"
        case "iPhone7,2":                               return "iPhone 6"
        case "iPhone7,1":                               return "iPhone 6 Plus"
        case "iPhone8,1":                               return "iPhone 6s"
        case "iPhone8,2":                               return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3":                  return "iPhone 7"
        case "iPhone9,2", "iPhone9,4":                  return "iPhone 7 Plus"
        case "iPhone8,4":                               return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3":           return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6":           return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3":           return "iPad Air"
        case "iPad5,3", "iPad5,4":                      return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7":           return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6":           return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9":           return "iPad Mini 3"
        case "iPad5,1", "iPad5,2":                      return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8":return "iPad Pro"
        case "AppleTV5,3":                              return "Apple TV"
        case "i386", "x86_64":                          return "Simulator"
        default:                                        return identifier
        }
    }
}

if UIDevice.currentDevice().modelName == "iPhone 6 Plus" {
    // is an iPhone 6 Plus
```

```
    }

Swift 3

import UIKit

public extension UIDevice {

    var modelName: String {
        var systemInfo = utsname()
        uname(&systemInfo)
        let machineMirror = Mirror(reflecting: systemInfo.machine)
        let identifier = machineMirror.children.reduce("") { identifier, element in
            guard let value = element.value as? Int8 , value != 0 else { return identifier      }
            return identifier + String(UnicodeScalar(UInt8(value)))
        }

        switch identifier {
        case "iPod5,1":                                    return "iPod Touch 5"
        case "iPod7,1":                                    return "iPod Touch 6"
        case "iPhone3,1", "iPhone3,2", "iPhone3,3":        return "iPhone 4"
        case "iPhone4,1":                                  return "iPhone 4s"
        case "iPhone5,1", "iPhone5,2":                     return "iPhone 5"
        case "iPhone5,3", "iPhone5,4":                     return "iPhone 5c"
        case "iPhone6,1", "iPhone6,2":                     return "iPhone 5s"
        case "iPhone7,2":                                  return "iPhone 6"
        case "iPhone7,1":                                  return "iPhone 6 Plus"
        case "iPhone8,1":                                  return "iPhone 6s"
        case "iPhone8,2":                                  return "iPhone 6s Plus"
        case "iPhone9,1", "iPhone9,3":                     return "iPhone 7"
        case "iPhone9,2", "iPhone9,4":                     return "iPhone 7 Plus"
        case "iPhone8,4":                                  return "iPhone SE"
        case "iPad2,1", "iPad2,2", "iPad2,3", "iPad2,4":return "iPad 2"
        case "iPad3,1", "iPad3,2", "iPad3,3":              return "iPad 3"
        case "iPad3,4", "iPad3,5", "iPad3,6":              return "iPad 4"
        case "iPad4,1", "iPad4,2", "iPad4,3":              return "iPad Air"
        case "iPad5,3", "iPad5,4":                         return "iPad Air 2"
        case "iPad2,5", "iPad2,6", "iPad2,7":              return "iPad Mini"
        case "iPad4,4", "iPad4,5", "iPad4,6":              return "iPad Mini 2"
        case "iPad4,7", "iPad4,8", "iPad4,9":              return "iPad Mini 3"
        case "iPad5,1", "iPad5,2":                         return "iPad Mini 4"
        case "iPad6,3", "iPad6,4", "iPad6,7", "iPad6,8":return "iPad Pro"
        case "AppleTV5,3":                                 return "Apple TV"
        case "i386", "x86_64":                             return "Simulator"
        default:                                           return identifier
        }
    }
}

if UIDevice.current.modelName == "iPhone 7" {
    // is an iPhone 7
}
```

# Section 50.2: Identifying the Device and Operating

```
UIDevice *deviceInfo = [UIDevice currentDevice];
NSLog(@"Device Name %@", deviceInfo.name);
//Ex: myIphone6s
NSLog(@"System Name %@", deviceInfo.systemName);
//Device Name iPhone OS
```

```
NSLog(@"System Version %@", deviceInfo.systemVersion);
//System Version 9.3
NSLog(@"Model %@", deviceInfo.model);
//Model iPhone
NSLog(@"Localized Model %@", deviceInfo.localizedModel);
//Localized Model iPhone
int device=deviceInfo.userInterfaceIdiom;
//UIUserInterfaceIdiomPhone=0
//UIUserInterfaceIdiomPad=1
//UIUserInterfaceIdiomTV=2
//UIUserInterfaceIdiomCarPlay=3
//UIUserInterfaceIdiomUnspecified=-1
NSLog(@"identifierForVendor %@", deviceInfo.identifierForVendor);
//identifierForVendor <__NSConcreteUUID 0x7a10ae20> 556395DC-0EB4-4FD5-BC7E-B16F612ECC6D
```

# Section 50.3: Getting the Device Orientation

```
UIDevice *deviceInfo = [UIDevice currentDevice];
int d = deviceInfo.orientation;
```

deviceInfo.orientation returns an UIDeviceOrientation value which is shown as below:

```
UIDeviceOrientationUnknown 0
UIDeviceOrientationPortrait 1
UIDeviceOrientationPortraitUpsideDown 2
UIDeviceOrientationLandscapeLeft 3
UIDeviceOrientationLandscapeRight 4
UIDeviceOrientationFaceUp 5
UIDeviceOrientationFaceDown 6
```

Listening for device orientation changes in a View Controller:

```
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];
    [[NSNotificationCenter defaultCenter] addObserver:self
                                    selector:@selector(deviceOrientationDidChange)
                                        name:UIDeviceOrientationDidChangeNotification
                                      object:nil];
}

-(void)deviceOrientationDidChange
{
    UIDeviceOrientation orientation = [[UIDevice currentDevice] orientation];
    if (orientation == UIDeviceOrientationPortrait || orientation ==
UIDeviceOrientationPortraitUpsideDown) {
        [self changedToPortrait];
    } else if (orientation == UIDeviceOrientationLandscapeLeft || orientation ==
UIDeviceOrientationLandscapeRight) {
        [self changedToLandscape];
    }
}

-(void)changedToPortrait
{
    // Function Body
}

-(void)changedToLandscape
{
```

```
    // Function Body
}
```

To disable checking for any orientation change:

```
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    [[UIDevice currentDevice] endGeneratingDeviceOrientationNotifications];
}
```

# Section 50.4: Getting the Device Battery State

```
//Get permission for Battery Monitoring
[[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];
UIDevice *myDevice = [UIDevice currentDevice];

[myDevice setBatteryMonitoringEnabled:YES];
double batLeft = (float)[myDevice batteryLevel] * 100;
NSLog(@"%.f",batLeft);

int d = myDevice.batteryState;
//Returns an Integer Value
//UIDeviceBatteryStateUnknown 0
//UIDeviceBatteryStateUnplugged 1
//UIDeviceBatteryStateCharging 2
//UIDeviceBatteryStateFull 3

//Using notifications for Battery Monitoring
-(void)startMonitoringForBatteryChanges
{
// Enable monitoring of battery status
[[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];
// Request to be notified when battery charge or state changes
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)
name:UIDeviceBatteryLevelDidChangeNotification object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(checkBatteryStatus)
name:UIDeviceBatteryStateDidChangeNotification object:nil];
}
-(void) checkBatteryStatus
{
NSLog (@"Battery Level is %.f",[[UIDevice currentDevice] batteryLevel]*100);
 int d=[[UIDevice currentDevice] batteryState];
    if (d==0)
    {
        NSLog(@"Unknown");
    }
    else if (d==1)
    {
        NSLog(@"Unplugged");
    }
    else if (d==2)
    {
        NSLog(@"Charging");
    }
    else if (d==3)
    {
        NSLog(@"Battery Full");
    }
}
```

# Section 50.5: Using the Proximity Sensor

```objc
//Enabling the proximity Sensor
- (void)viewWillAppear:(BOOL)animated {

    [super viewWillAppear:animated];
    [[UIDevice currentDevice] setProximityMonitoringEnabled:YES];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(sensorStateMonitor:)
name:@"UIDeviceProximityStateDidChangeNotification" object:nil];
}

- (void)sensorStateMonitor:(NSNotificationCenter *)notification
{
    if ([[UIDevice currentDevice] proximityState] == YES)
    {
        NSLog(@"Device is close to user.");
    }

    else
    {
        NSLog(@"Device is not closer to user.");
    }
}
```

# Section 50.6: Getting Battery Status and Battery Level

```swift
override func viewDidLoad() {
    super.viewDidLoad()
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryStateDidChange:")),
name: NSNotification.Name.UIDeviceBatteryStateDidChange, object: nil)
    NotificationCenter.default.addObserver(self, selector: Selector(("batteryLevelDidChange:")),
name: NSNotification.Name.UIDeviceBatteryLevelDidChange, object: nil)

    // Stuff...
}

func batteryStateDidChange(notification: NSNotification){
    // The stage did change: plugged, unplugged, full charge...
}

func batteryLevelDidChange(notification: NSNotification){

    let batteryLevel = UIDevice.current.batteryLevel
    if batteryLevel < 0.0 {
        print(" -1.0 means battery state is UIDeviceBatteryStateUnknown")
        return
    }

    print("Battery Level : \(batteryLevel * 100)%")
    // The battery's level did change (98%, 99%, ...)
}
```

# Chapter 51: Make selective UIView corners rounded

## Section 51.1: Objective C code to make selected corner of a UiView rounded

First **import** `#import <QuartzCore/QuartzCore.h>` into your ViewController class. Here is how I set my view in code

```objc
UIView *view1=[[UIView alloc]init];
view1.backgroundColor=[UIColor colorWithRed:255/255.0 green:193/255.0 blue:72/255.0 alpha:1.0];
CGRect view1Frame = view1.frame;
view1Frame.size.width = SCREEN_WIDTH*0.97;
view1Frame.size.height = SCREEN_HEIGHT*0.2158;
view1Frame.origin.x = 0;
view1Frame.origin.y = 0.1422*SCREEN_HEIGHT-10;
view1.frame = view1Frame;
[self setMaskTo:view1 byRoundingCorners:UIRectCornerBottomRight|UIRectCornerTopRight];
[self.view addSubview:view1];
```

Here is the function which does the heavy lifting and rounds off the selected edges which is the Bottom Right and the Top Right edge in our case

```objc
- (void)setMaskTo:(UIView*)view byRoundingCorners:(UIRectCorner)corners
{
    UIBezierPath *rounded = [UIBezierPath bezierPathWithRoundedRect:view.bounds
                                        byRoundingCorners:corners
                                              cornerRadii:CGSizeMake(20.0, 20.0)];
    CAShapeLayer *shape = [[CAShapeLayer alloc] init];
    [shape setPath:rounded.CGPath];
    view.layer.mask = shape;
}
```

# Chapter 52: Custom UIViews from XIB files

## Section 52.1: Wiring elements

Create a XIB file

Xcode Menu Bar > File > New > File.
Select iOS, User Interface and then "View":
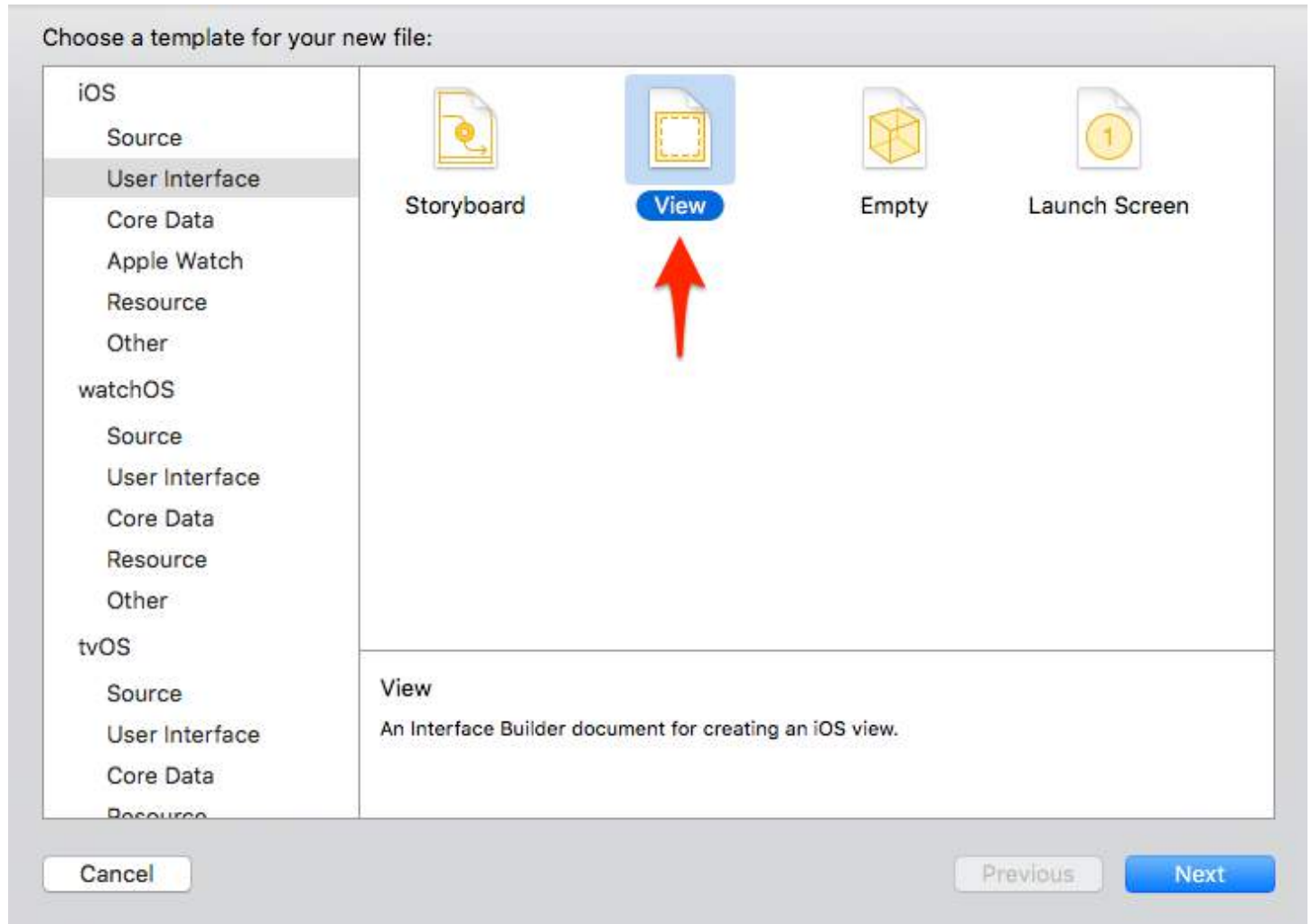


Give your XIB a name (yes, we are doing a Pokemon example ????).
**Remember to check your target** and hit "Create".
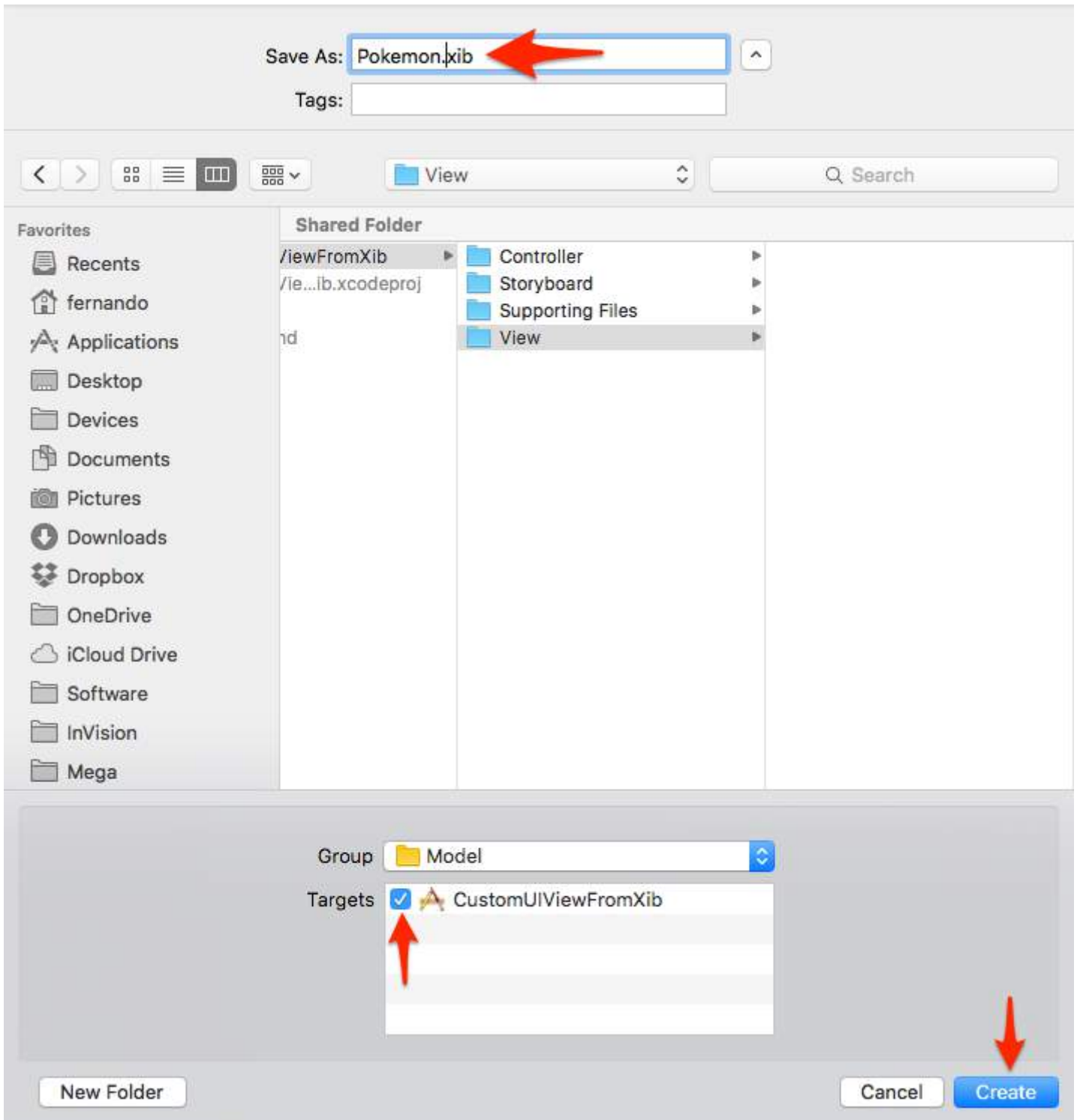
Design your view

To make things easier, set:

- Size: Freeform
- Status Bar: None
- Top Bar: None
- Bottom Bar: None
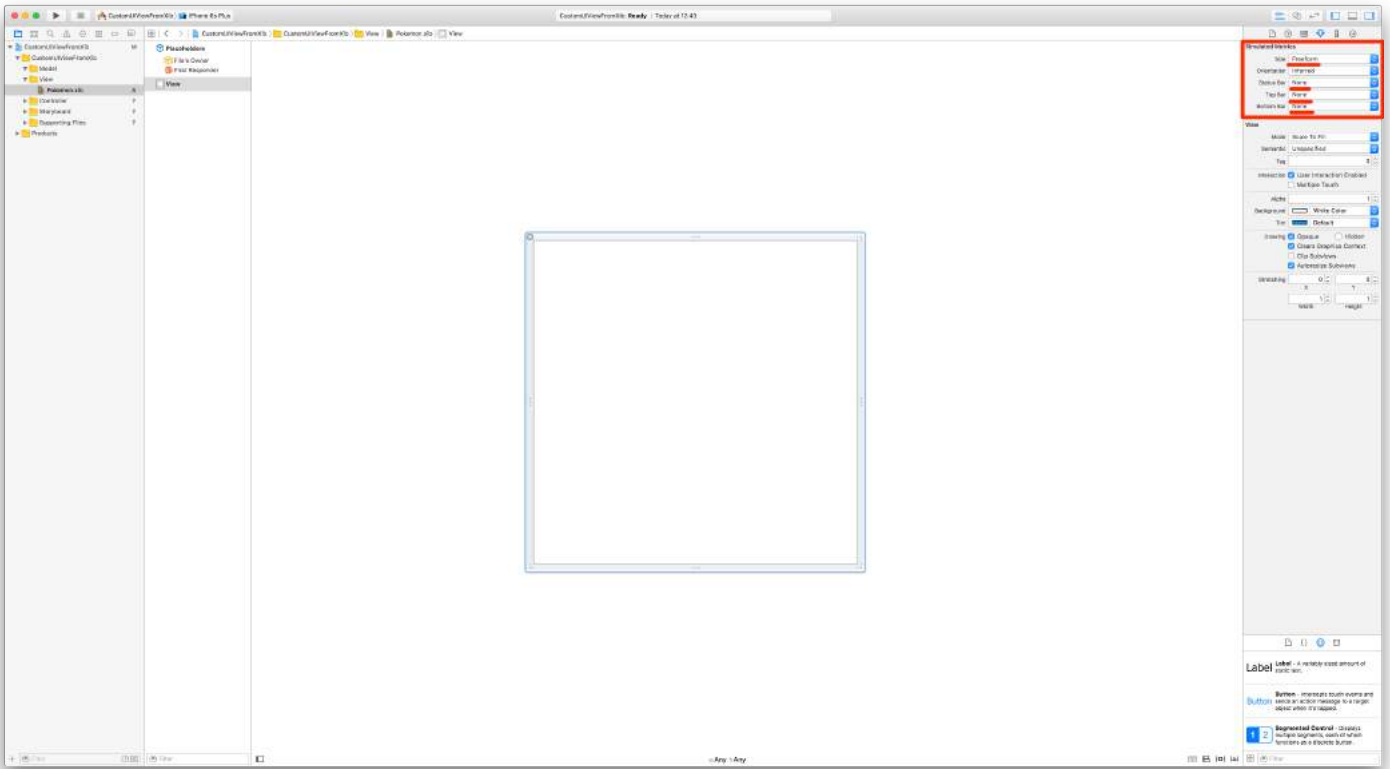
Click on the Size Inspector and resize the view.

For this example we'll be using width 321 and height 256.
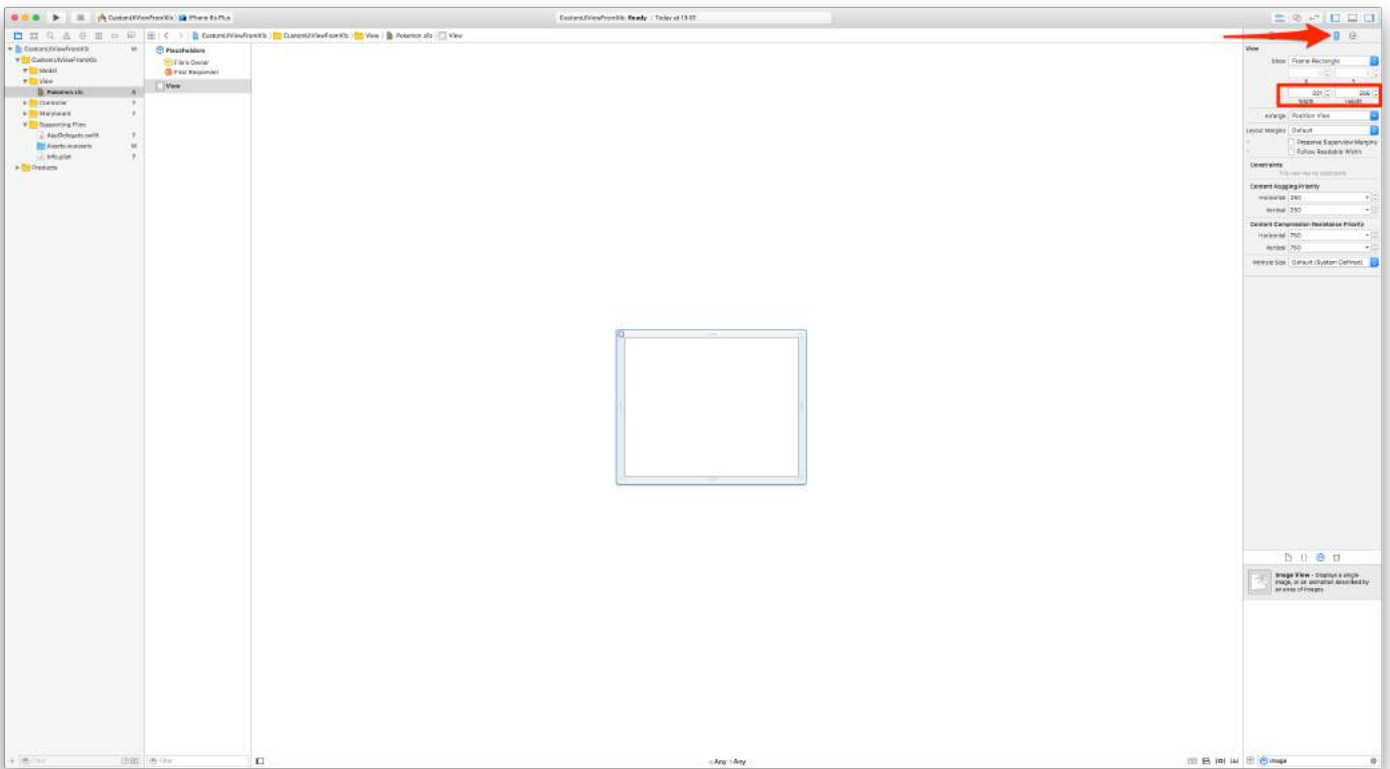


Drop some elements into your XIB file like shown below.

Here we'll be adding an **Image View** (256x256) and a **Switch**.
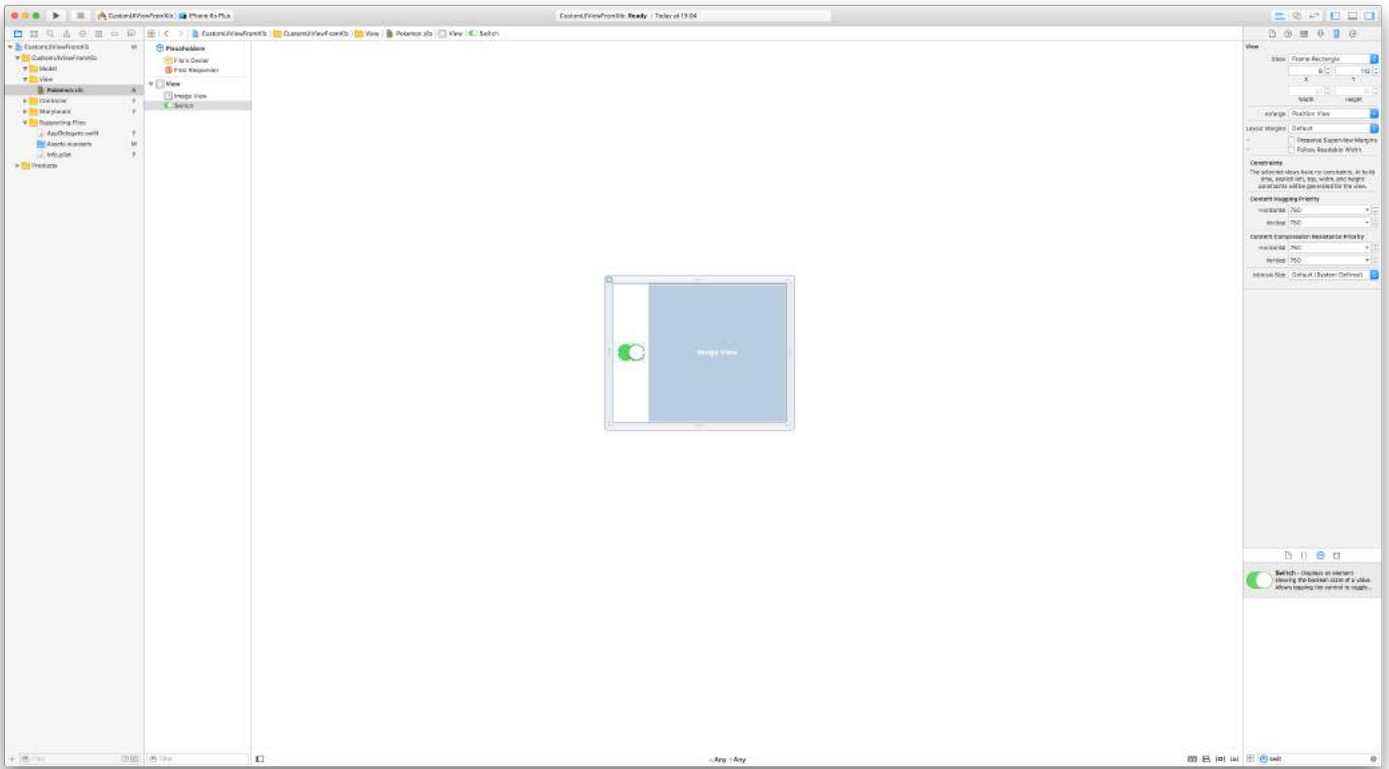
Add Auto-Layout constraints by clicking on "Resolve Auto Layout Issues" (bottom-right) and selecting "Add Missing Constraints" under "All Views".
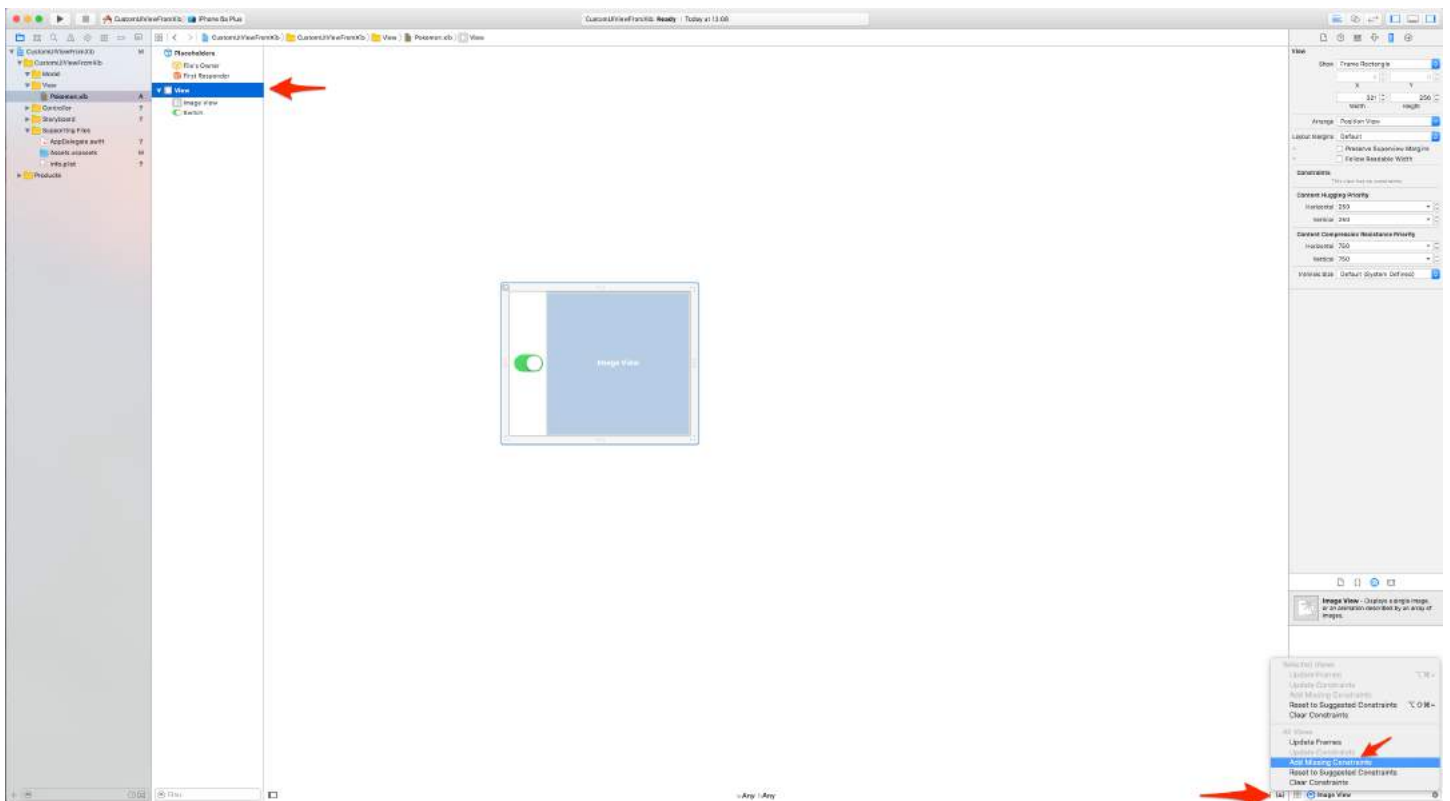


Preview the changes you made by clicking on "Show the Assistant Editor" (top-right), then "Preview".
You can add iPhone screens by clicking on the "Plus" button.
The preview should look like this:
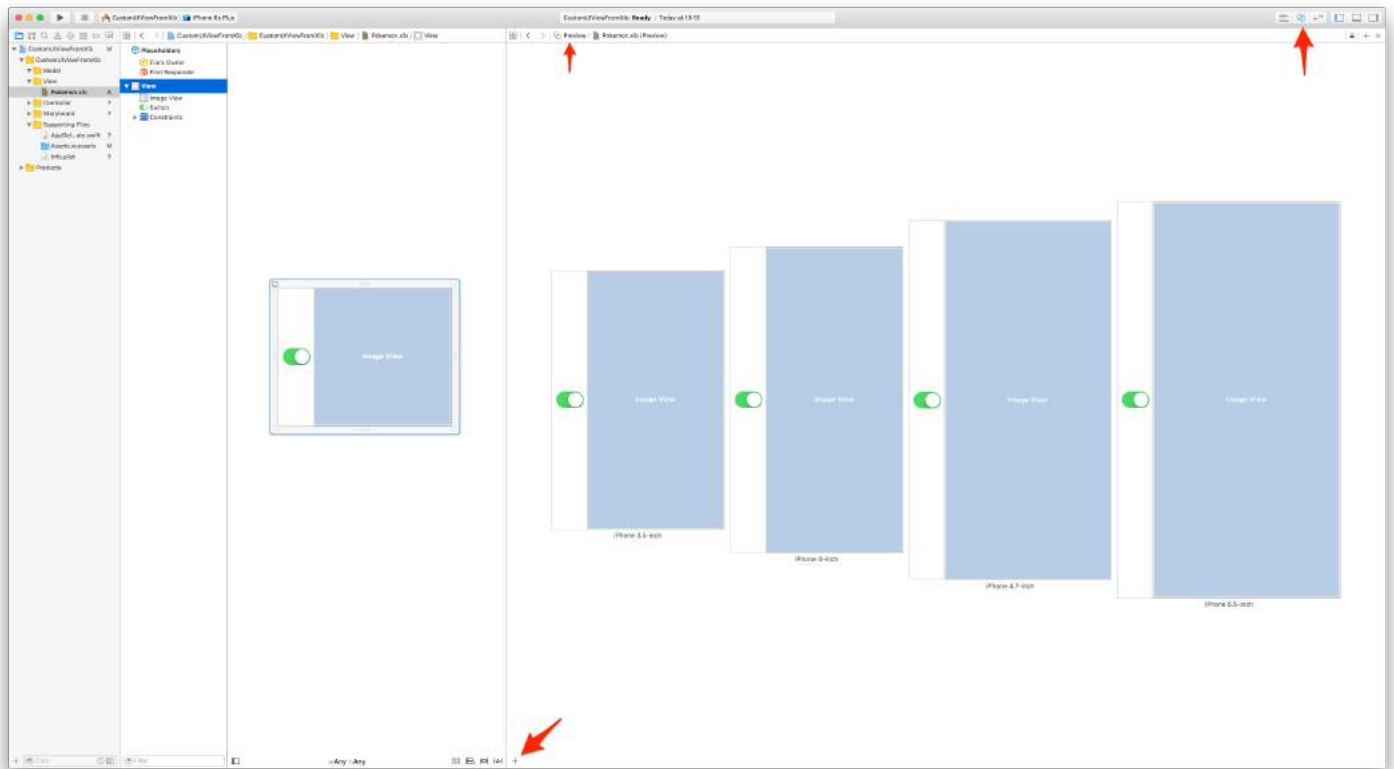
Subclass UIView

Create the class that is going to manage the XIB file.
Xcode Menu Bar > File > New > File.
Select iOS / Source / Cocoa Touch Class. Hit "Next".
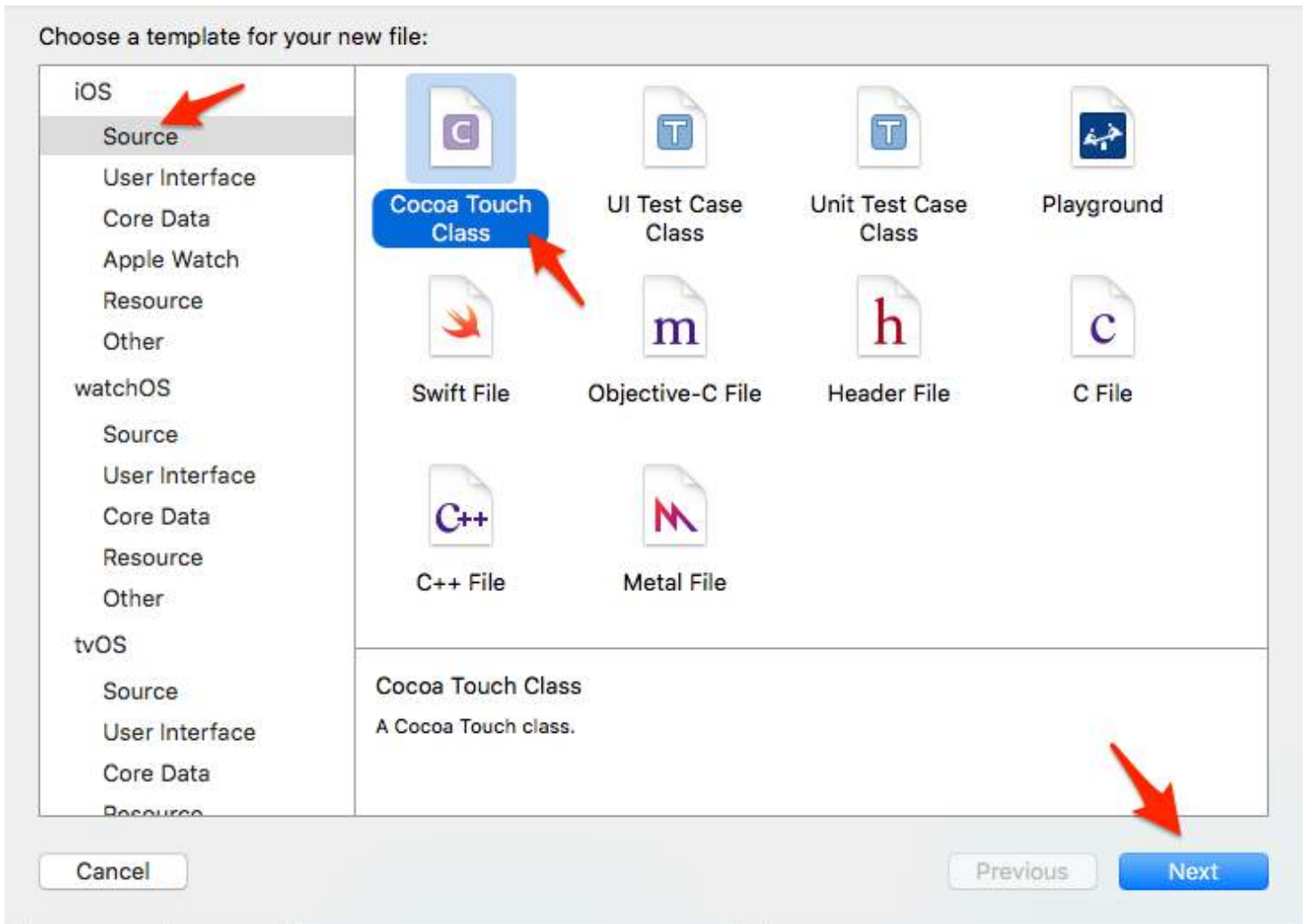
Give the class a name, which must be the same name as the XIB file (Pokemon).
Select UIView as the subclass type, then hit "Next".
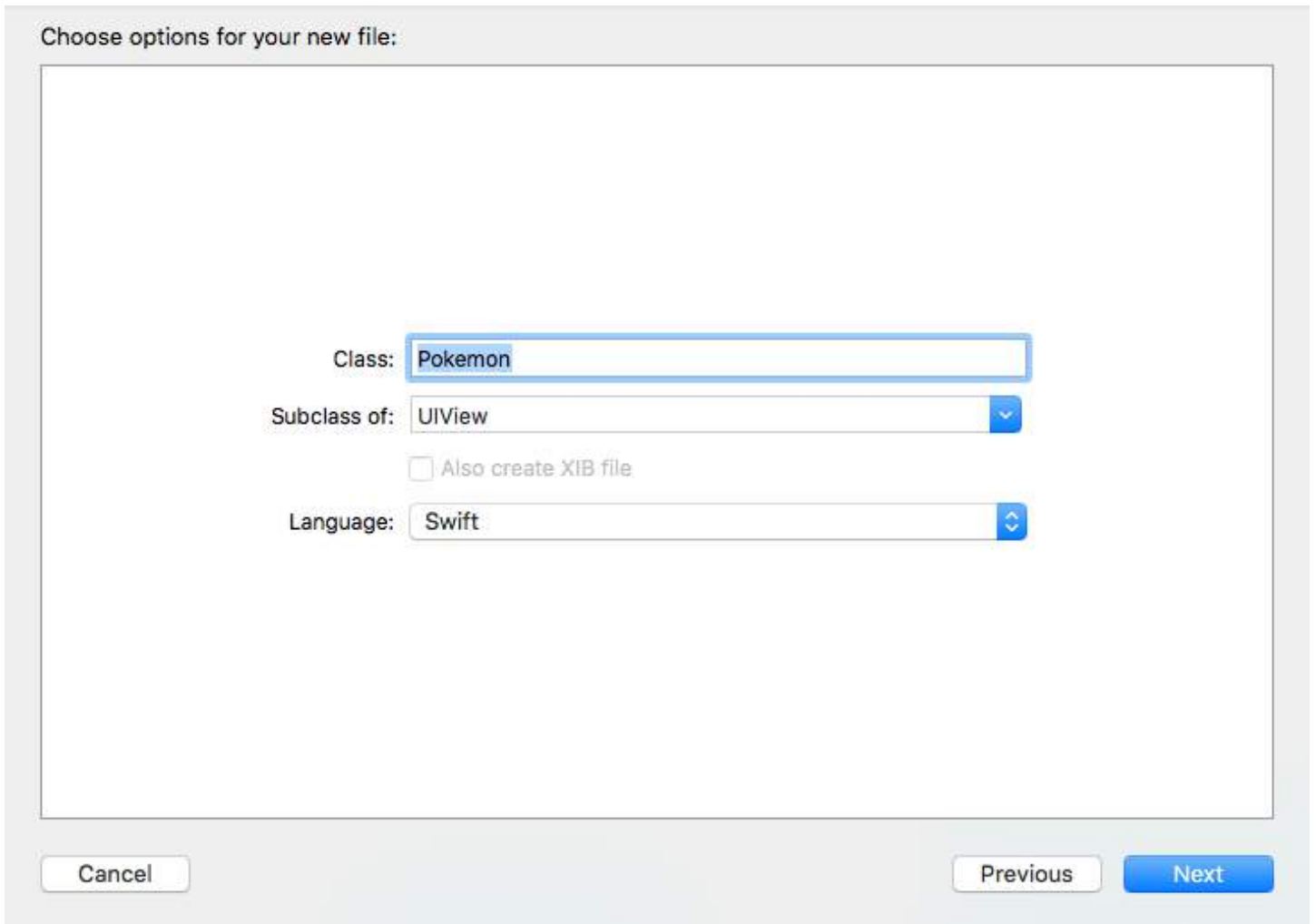
On the next window, select your target and hit "Create".
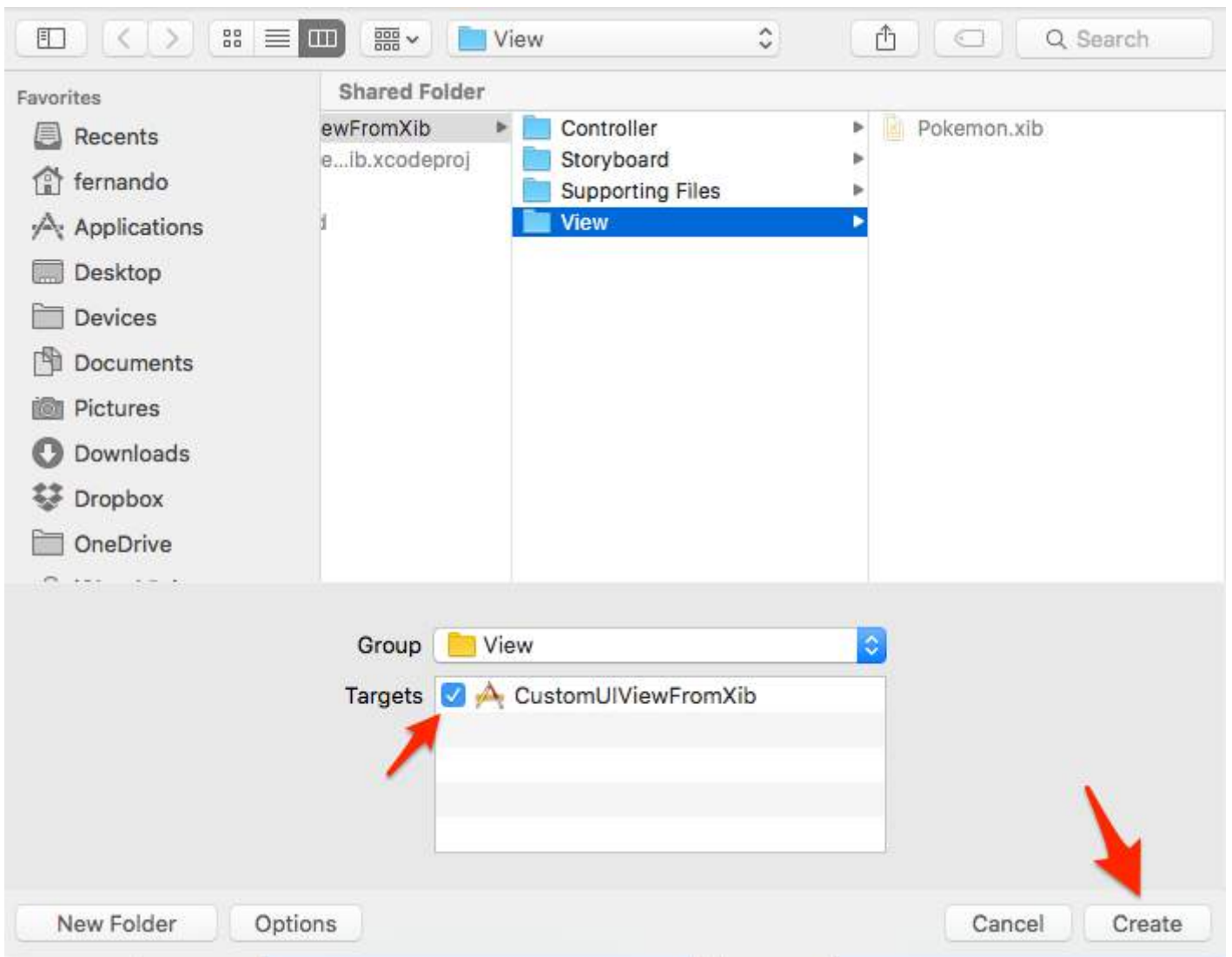
> Connect Pokemon.xib to Pokemon.swift via "File's Owner" attribute

Click on the Pokemon.xib file in Xcode.
Click on the "File's Owner" outlet.
On the "Identity inspector" (top-right), set the Class to our recently created Pokemon.swift file.
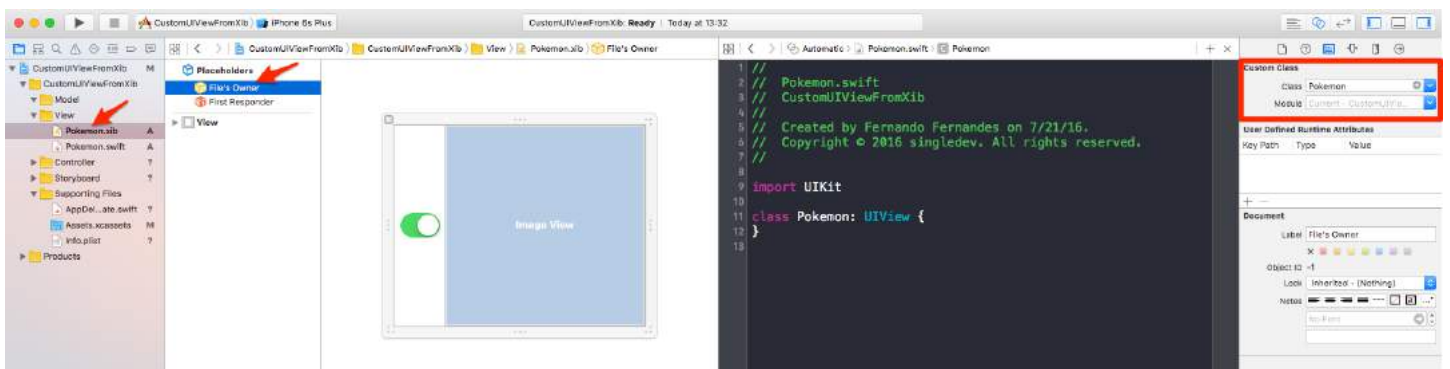


> POKEMONS!!!

Yes! Drag and drop some Pokemons into your project to finish up our "infrastructure".
Here we are adding two PGN files, 256x256, transparent.
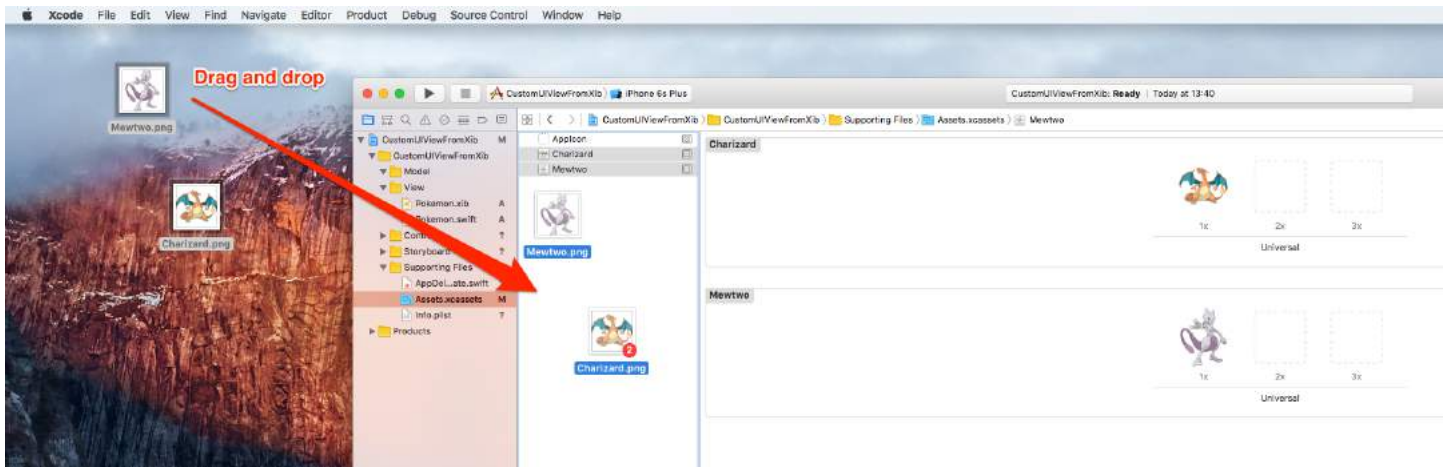
> Show me code already.

All right, all right.
Time to add some code to our Pokemon.swift class.

It's actually pretty simple:

1. Implement required initializers
2. Load the XIB file
3. Configure the view that will display the XIB file
4. Show the above view

Add the following code to the Pokemon.swift class:

```swift
import UIKit

class Pokemon: UIView {

    // MARK: - Initializers

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setupView()
    }

    // MARK: - Private Helper Methods

    // Performs the initial setup.
    private func setupView() {
        let view = viewFromNibForClass()
        view.frame = bounds

        // Auto-layout stuff.
        view.autoresizingMask = [
            UIViewAutoresizing.flexibleWidth,
            UIViewAutoresizing.flexibleHeight
        ]

        // Show the view.
```

```
        addSubview(view)
    }

    // Loads a XIB file into a view and returns this view.
    private func viewFromNibForClass() -> UIView {

        let bundle = Bundle(for: type(of: self))
        let nib = UINib(nibName: String(describing: type(of: self)), bundle: bundle)
        let view = nib.instantiate(withOwner: self, options: nil).first as! UIView

        /* Usage for swift < 3.x
        let bundle = NSBundle(forClass: self.dynamicType)
        let nib = UINib(nibName: String(self.dynamicType), bundle: bundle)
        let view = nib.instantiateWithOwner(self, options: nil)[0] as! UIView
        */

        return view
    }
}
```

| @IBDesignable and @IBInspectable

By adding `@IBDesignable` to your class, you make possible for it to live-render in Interface Builder.
By adding `@IBInspectable` to the properties of your class, you can see your custom views changing in Interface
Builder as soon as you modify those properties.

Let's make the `Image View` of our custom view "Inspectable".

First, hook up the `Image View` from the Pokemon.xib file to the Pokemon.swift class.
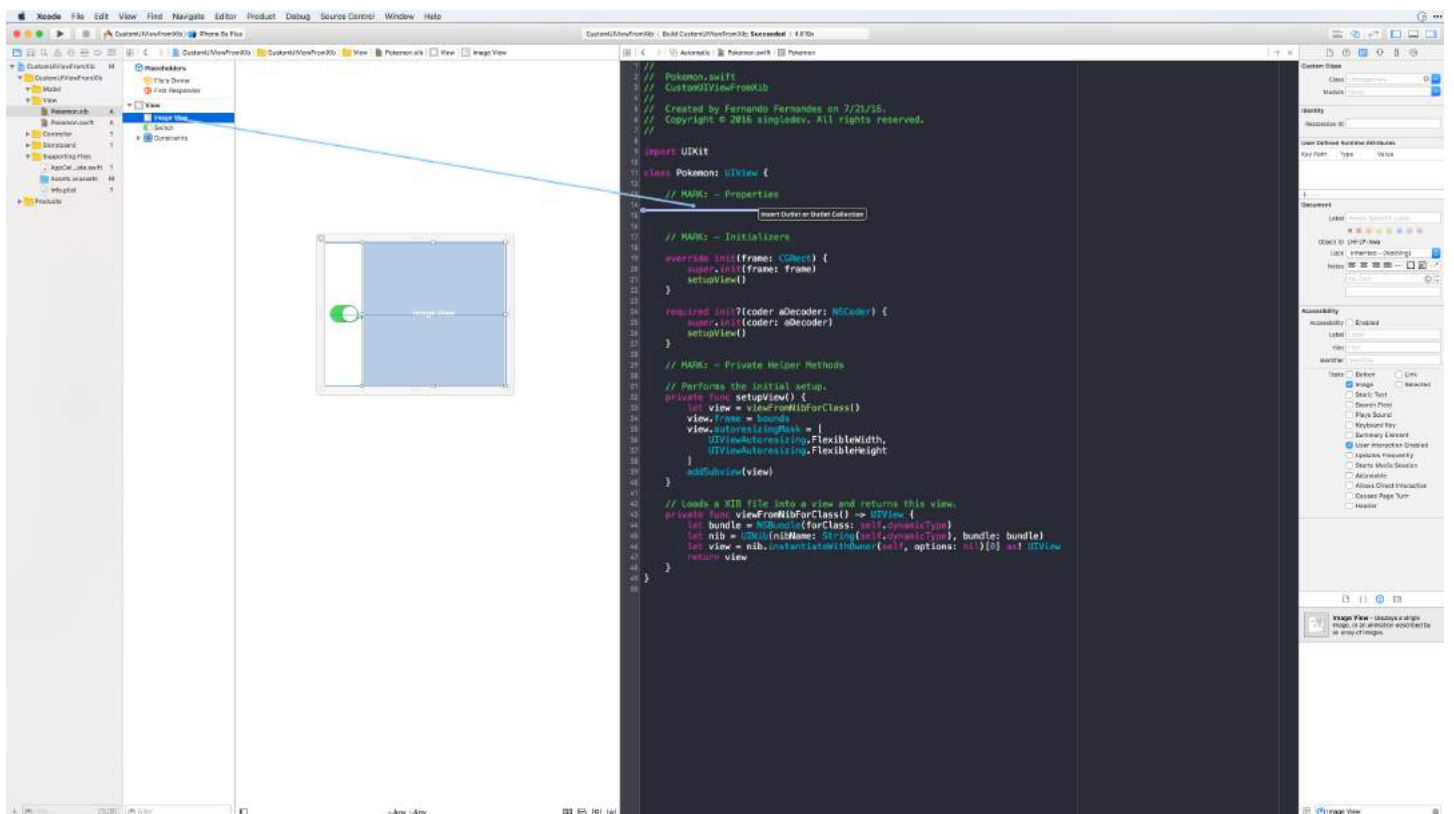


Call the outlet `imageView` and then add the following code (notice the `@IBDesignable` before the class name):

```
@IBDesignable class Pokemon: UIView {
```

```
    // MARK: - Properties

    @IBOutlet weak var imageView: UIImageView!

    @IBInspectable var image: UIImage? {
        get {
            return imageView.image
        }
        set(image) {
            imageView.image = image
        }
    }

    // MARK: - Initializers
    ...
```

Using your Custom Views

Got to your Main storyboard file, drag a UIView into it.
Resize the view to, say 200x200. Centralize.
Go to the Identity inspector (top-right) and set the Class to Pokemon.
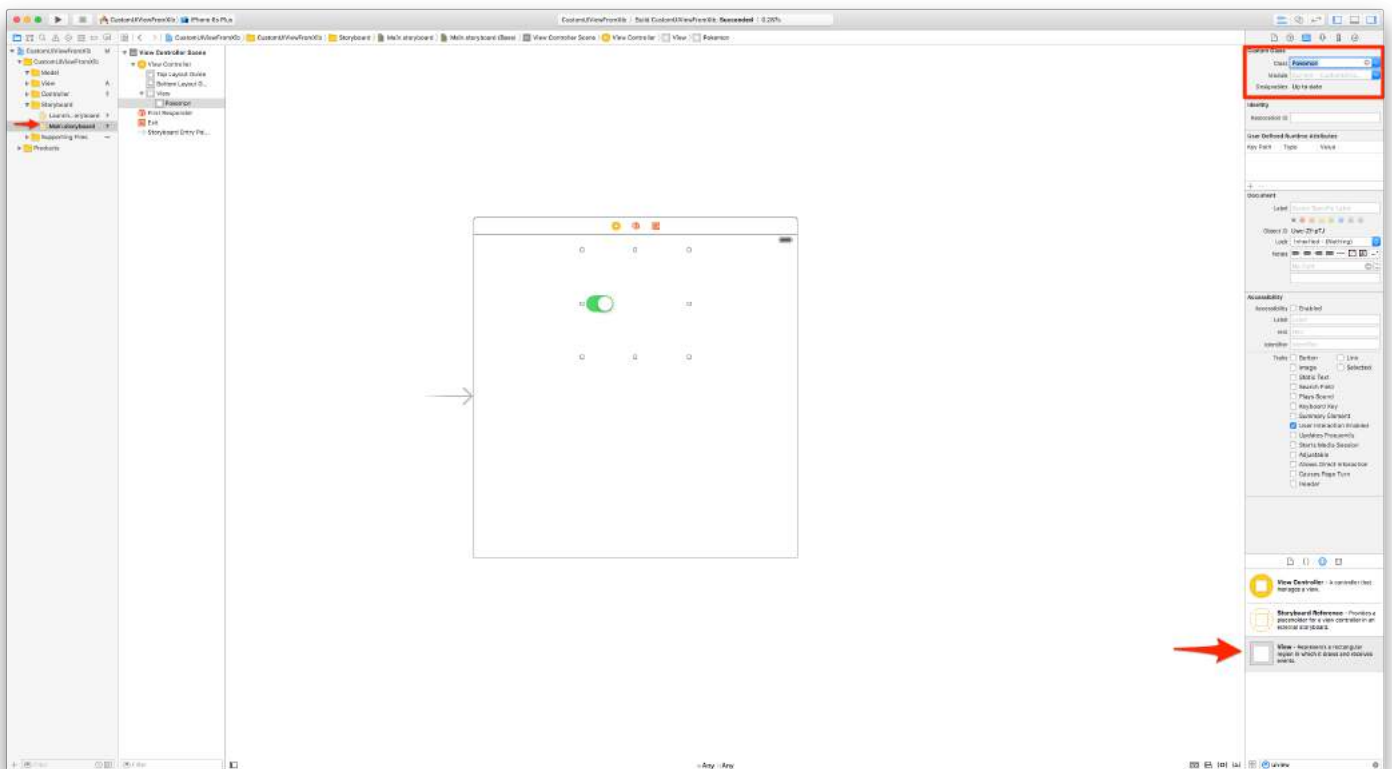


To select a Pokemon, go to the Attribute Inspector (top-right) and select one of the Pokemon images you previously added using the awesome `@IBInspectable` image property.
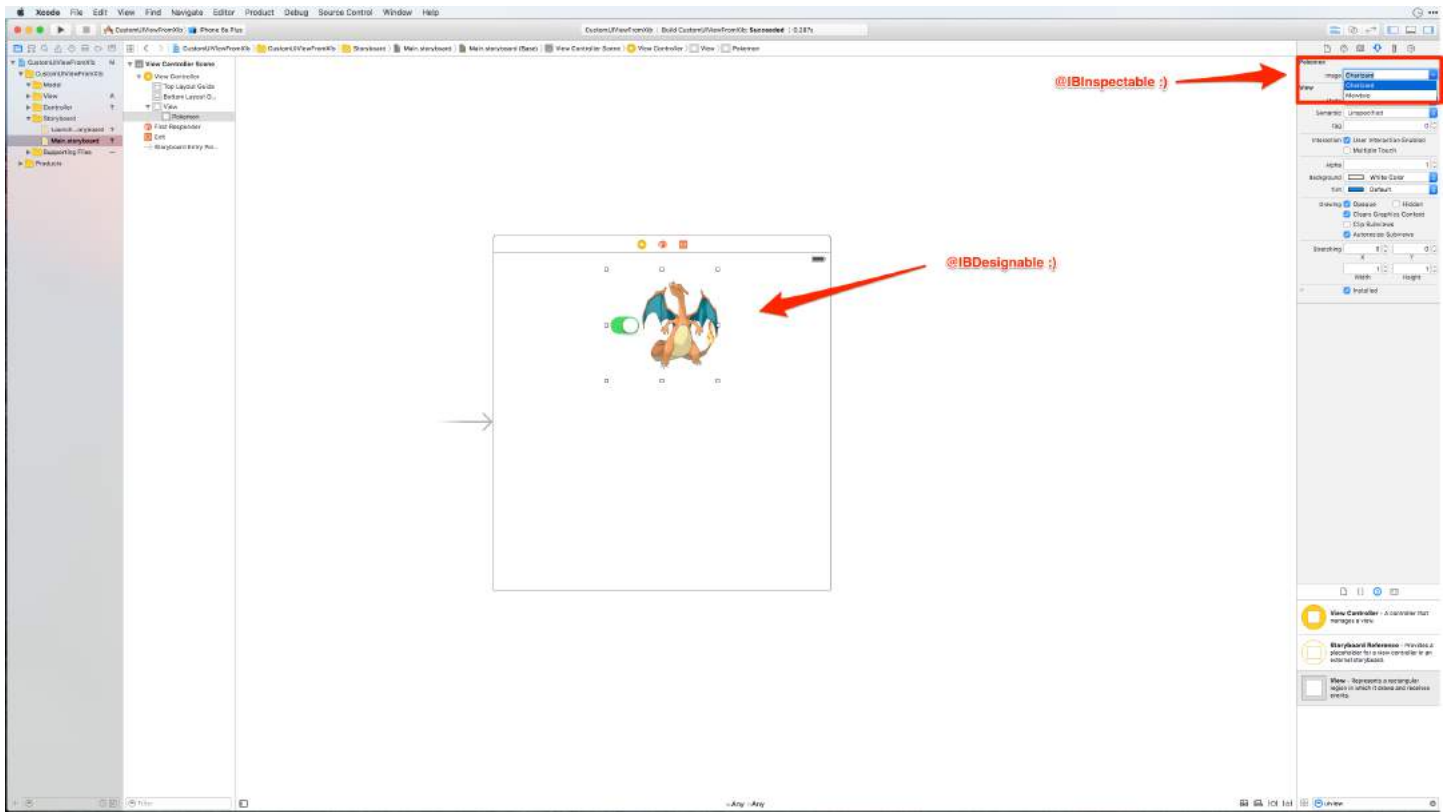
Now duplicate your custom Pokemon view.

Give it a different size, say 150x150.

Choose another Pokemon image, observe:
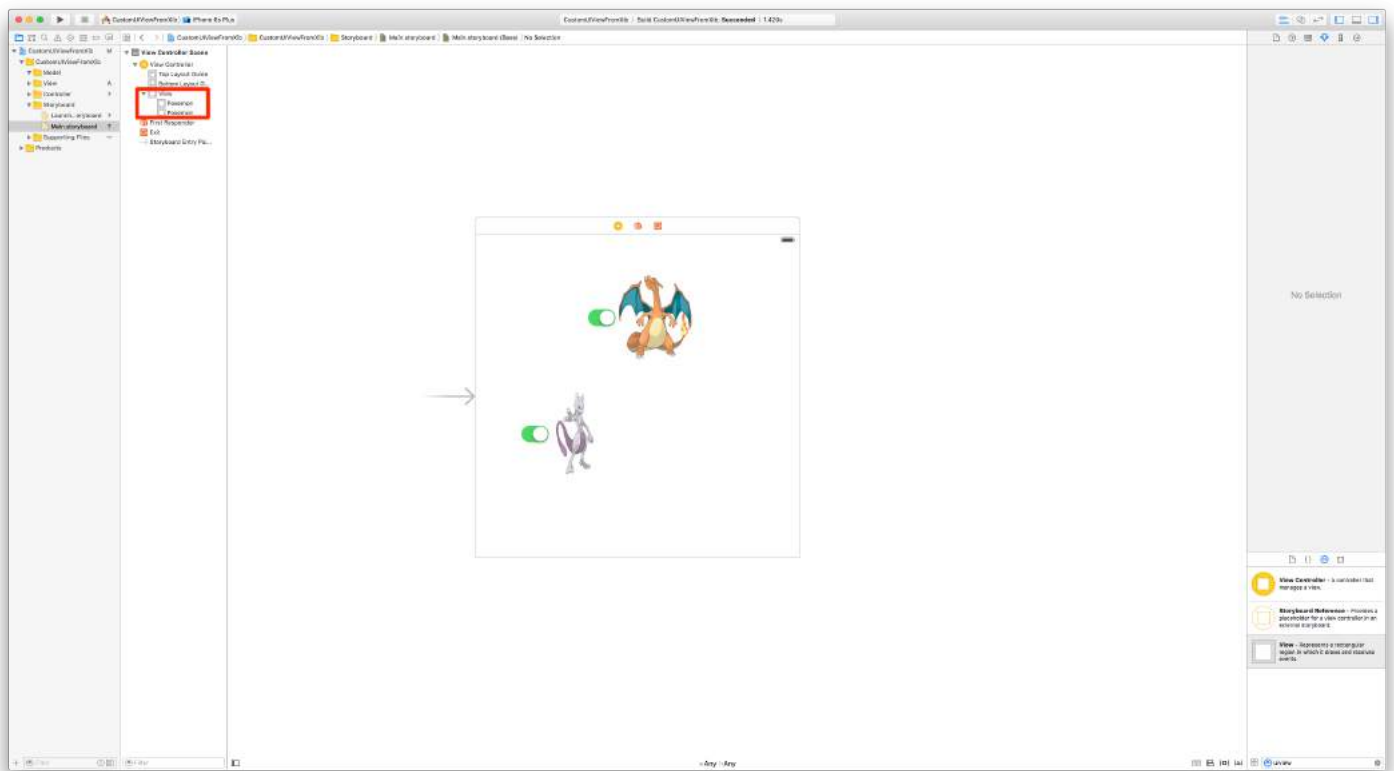


Now we are going to add more logic to that self-containing custom UI element.

The button will allow Pokemons to be enabled/disabled.

Create an `IBAction` from the Switch button to the Pokemon.swift class.

Call the action something like `switchTapped`.

Add the following code to it:

```swift
// MARK: - Actions

@IBAction func switchTapped(sender: UISwitch) {
    imageView.alpha = sender.on ? 1.0 : 0.2
}

// MARK: - Initializers
...
```

Final result:

You are done!

Now you can create complex custom views and reuse them anywhere you want.

This will increase productivity while isolating code into self-contained UI elements.

[The final project can be cloned in Github.](#)

(**Updated to Swift 3.1**)

# Section 52.2: How to make custom reusable UIView using XIB

Following example shows steps involved in initializing a view from XIB.

This is not a complex operation but exact steps need to be followed in order to do it right way first time, avoiding exceptions.

[How does loadNibNamed Works](#)

Main steps are:

1. Create XIB
2. Create class .h and .m
3. Define outlets in .h
4. Connect outlets between .h and XIB

See attached screenshot:
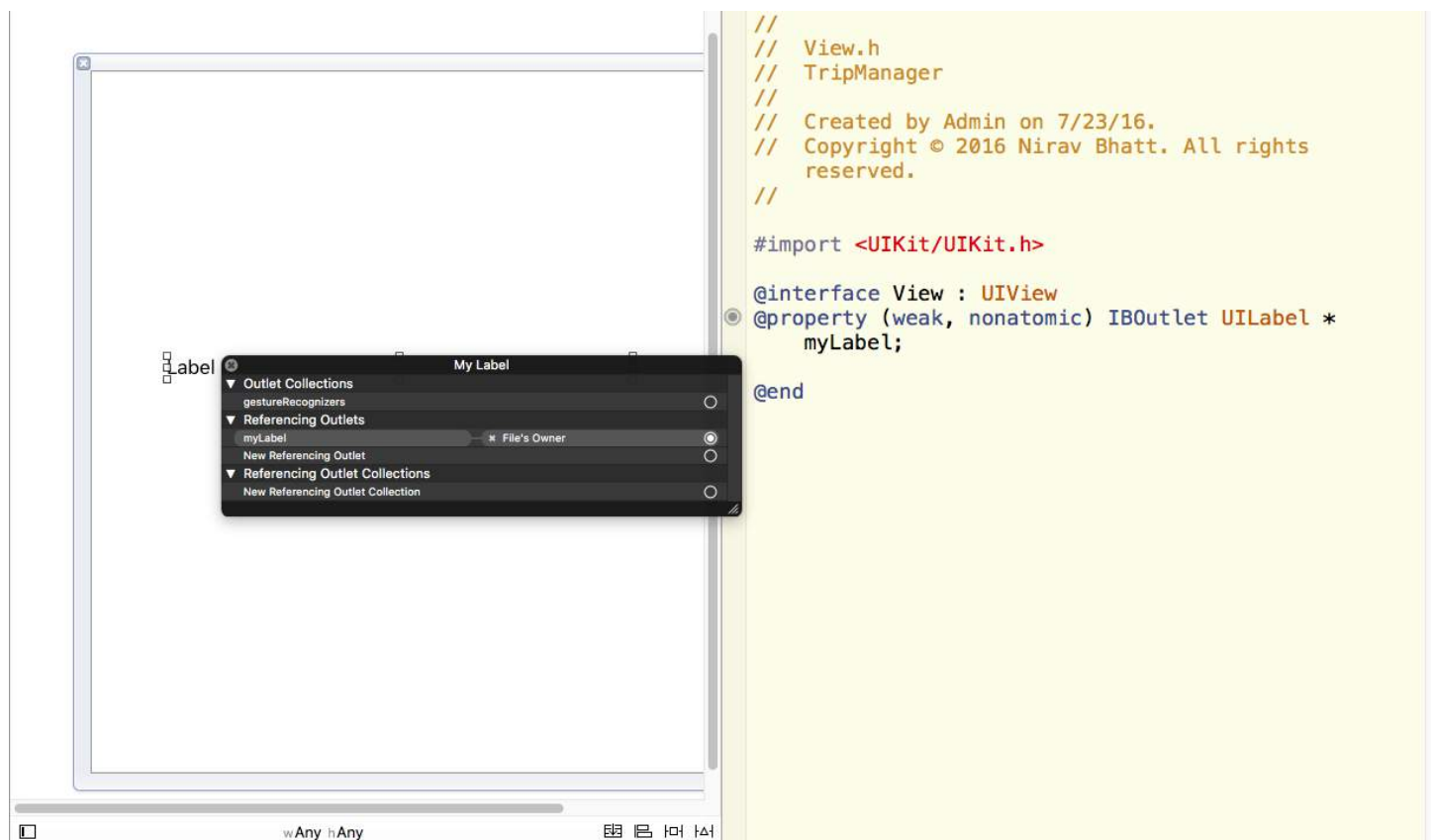


5. Invoke loadNibNamed inside initWithCoder function of .m file. This is needed to ensure you can directly place UIView object into storyboard / Parent UIView XIB file and define it as your custom view. No other initialization code is needed once you load the storyboard / parent XIB. Your custom view can be added to other views just like other built-in Objective C view objects given in XCode.

# Chapter 53: UIBezierPath

## Section 53.1: Designing and drawing a Bezier Path

This example shows the process from designing the shape you want to drawing it on a view. A specific shap is used but the concepts you learn can be applied to any shape.

**How to draw a [Bézier path](#) in a custom view**

These are the main steps:

1. Design the outline of the shape you want.
2. Divide the outline path into segments of lines, arcs, and curves.
3. Build that path programmatically.
4. Draw the path either in `drawRect` or using a `CAShapeLayer`.

**Design shape outline**

You could do anything, but as an example I have chosen the shape below. It could be a popup key on a keyboard.
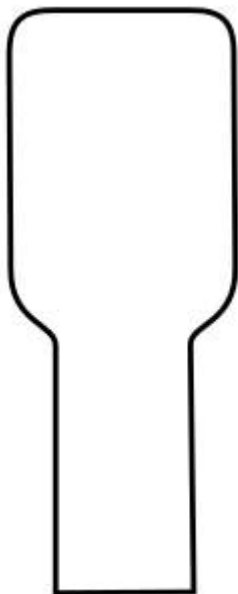


**Divide the path into segments**

Look back at your shape design and break it down into simpler elements of lines (for straight lines), arcs (for circles and round corners), and curves (for anything else).

Here is what our example design would look like:
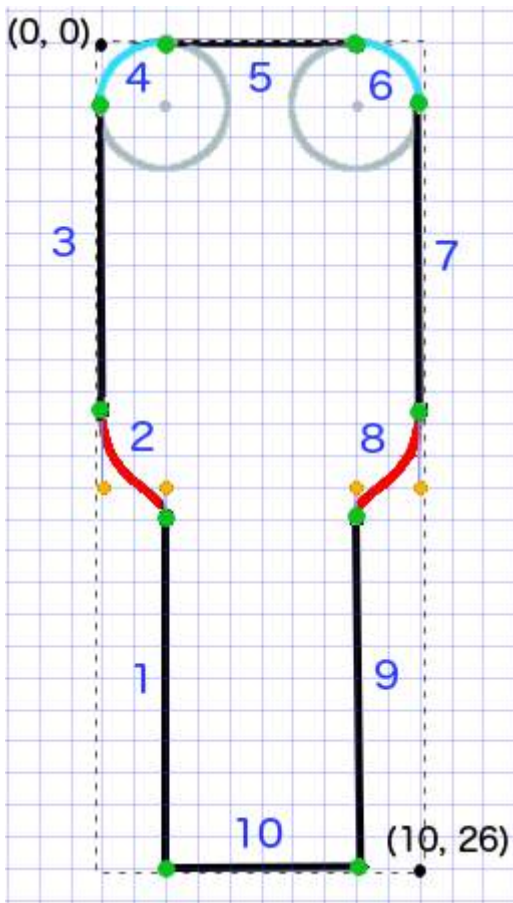
- Black are line segments
- Light blue are arc segments
- Red are curves
- Orange dots are the control points for the curves
- Green dots are the points between path segments
- Dotted lines show the bounding rectangle
- Dark blue numbers are the segments in the order that they will be added programmatically

**Build the path programmatically**

We'll arbitrarily start in the bottom left corner and work clockwise. I'll use the grid in the image to get the x and y values for the points. I'll hardcode everything here, but of course you wouldn't do that in a real project.

The basic process is:

1. Create a new `UIBezierPath`
2. Choose a starting point on the path with `moveToPoint`
3. Add segments to the path

- line: `addLineToPoint`
- arc: `addArcWithCenter`
- curve: `addCurveToPoint`

4. Close the path with `closePath`

Here is the code to make the path in the image above.

```
func createBezierPath() -> UIBezierPath {

    // create a new path
    let path = UIBezierPath()
```

```swift
// starting point for the path (bottom left)
path.moveToPoint(CGPoint(x: 2, y: 26))

// *********************
// ***** Left side *****
// *********************

// segment 1: line
path.addLineToPoint(CGPoint(x: 2, y: 15))

// segment 2: curve
path.addCurveToPoint(CGPoint(x: 0, y: 12), // ending point
    controlPoint1: CGPoint(x: 2, y: 14),
    controlPoint2: CGPoint(x: 0, y: 14))

// segment 3: line
path.addLineToPoint(CGPoint(x: 0, y: 2))

// *********************
// ****** Top side *****
// *********************

// segment 4: arc
path.addArcWithCenter(CGPoint(x: 2, y: 2), // center point of circle
    radius: 2, // this will make it meet our path line
    startAngle: CGFloat(M_PI), // ? radians = 180 degrees = straight left
    endAngle: CGFloat(3*M_PI_2), // 3?/2 radians = 270 degrees = straight up
    clockwise: true) // startAngle to endAngle goes in a clockwise direction

// segment 5: line
path.addLineToPoint(CGPoint(x: 8, y: 0))

// segment 6: arc
path.addArcWithCenter(CGPoint(x: 8, y: 2),
    radius: 2,
    startAngle: CGFloat(3*M_PI_2), // straight up
    endAngle: CGFloat(0), // 0 radians = straight right
    clockwise: true)

// *********************
// ***** Right side ****
// *********************

// segment 7: line
path.addLineToPoint(CGPoint(x: 10, y: 12))

// segment 8: curve
path.addCurveToPoint(CGPoint(x: 8, y: 15), // ending point
    controlPoint1: CGPoint(x: 10, y: 14),
    controlPoint2: CGPoint(x: 8, y: 14))

// segment 9: line
path.addLineToPoint(CGPoint(x: 8, y: 26))

// *********************
// **** Bottom side ****
// *********************

// segment 10: line
path.closePath() // draws the final line to close the path
```

```
        return path
}
```

Note: Some of the above code can be reduced by adding a line and an arc in a single command (since the arc has an implied starting point). See [here](#) for more details.

**Draw the path**

We can draw the path either in a layer or in `drawRect`.

**Method 1: Draw path in a layer**

Our custom class looks like this. We add our Bezier path to a new `CAShapeLayer` when the view is initialized.

```swift
import UIKit
class MyCustomView: UIView {

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setup()
    }

    func setup() {

        // Create a CAShapeLayer
        let shapeLayer = CAShapeLayer()

        // The Bezier path that we made needs to be converted to
        // a CGPath before it can be used on a layer.
        shapeLayer.path = createBezierPath().CGPath

        // apply other properties related to the path
        shapeLayer.strokeColor = UIColor.blueColor().CGColor
        shapeLayer.fillColor = UIColor.whiteColor().CGColor
        shapeLayer.lineWidth = 1.0
        shapeLayer.position = CGPoint(x: 10, y: 10)

        // add the new layer to our custom view
        self.layer.addSublayer(shapeLayer)
    }

    func createBezierPath() -> UIBezierPath {

        // see previous code for creating the Bezier path
    }
}
```

And creating our view in the View Controller like this

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    // create a new UIView and add it to the view controller
    let myView = MyCustomView()
    myView.frame = CGRect(x: 100, y: 100, width: 50, height: 50)
```

```
        myView.backgroundColor = UIColor.yellowColor()
        view.addSubview(myView)

}
```

We get...
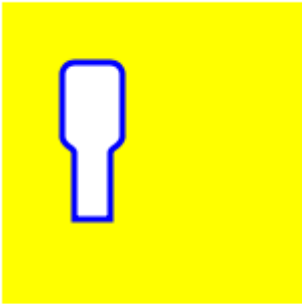


Hmm, that's a little small because I hardcoded all the numbers in. I can scale the path size up, though, like this:

```
let path = createBezierPath()
let scale = CGAffineTransformMakeScale(2, 2)
path.applyTransform(scale)
shapeLayer.path = path.CGPath
```
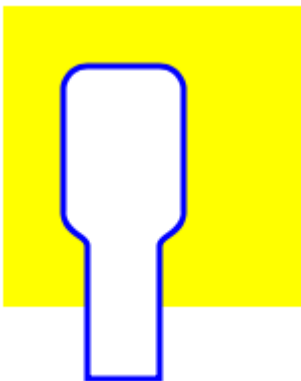


**Method 2: Draw path in `drawRect`**

Using `drawRect` is slower than drawing to the layer, so this is not the recommended method if you don't need it.

Here is the revised code for our custom view:

```
import UIKit
class MyCustomView: UIView {

    override func drawRect(rect: CGRect) {

        // create path (see previous code)
        let path = createBezierPath()

        // fill
        let fillColor = UIColor.whiteColor()
        fillColor.setFill()

        // stroke
        path.lineWidth = 1.0
        let strokeColor = UIColor.blueColor()
```

```
        strokeColor.setStroke()

        // Move the path to a new location
        path.applyTransform(CGAffineTransformMakeTranslation(10, 10))

        // fill and stroke the path (always do these last)
        path.fill()
        path.stroke()

    }

    func createBezierPath() -> UIBezierPath {

        // see previous code for creating the Bezier path
    }
}
```

which gives us the same result...
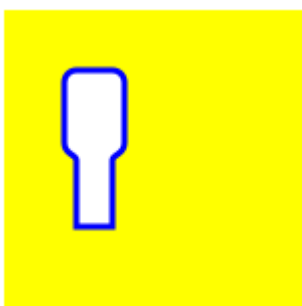


**Further study**

Excellent articles for understanding Bezier paths.

- Thinking like a Bézier path (Everything I've ever read from this author is good and the inspiration for my example above came from here.)
- Coding Math: Episode 19 - Bezier Curves (entertaining and good visual illustrations)
- Bezier Curves (how they are used in graphics applications)
- Bezier Curves (good description of how the mathematical formulas are derived)

**Notes**

- This example originally comes from this Stack Overflow answer.
- In your actual projects you probably shouldn't use hard coded numbers, but rather get the sizes from your view's bounds.

# Section 53.2: How to apply corner radius to rectangles drawn by UIBezierPath

*Corner radius for all 4 edges:*

```
  UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:
CGRectMake(x,y,width,height) cornerRadius: 11];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*Corner radius for top-left edge:*



```
  UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:
CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerTopLeft cornerRadii: CGSizeMake(11,
11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*Corner radius for top-right edge:*



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)
byRoundingCorners: UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*corner radius for bottom-left edge:*



```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)
byRoundingCorners: UIRectCornerBottomLeft cornerRadii: CGSizeMake(11, 11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*corner radius for bottom-right edge:*

```objc
 UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect:
CGRectMake(x,y,width,height) byRoundingCorners: UIRectCornerBottomRight cornerRadii: CGSizeMake(11,
11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*corner radius for bottom edges:*



```objc
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)
byRoundingCorners: UIRectCornerBottomLeft | UIRectCornerBottomRight cornerRadii: CGSizeMake(11,
11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*corner radius for top edges:*



```objc
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(x,y,width,height)
byRoundingCorners: UIRectCornerTopLeft | UIRectCornerTopRight cornerRadii: CGSizeMake(11, 11)];
[rectanglePath closePath];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

# Section 53.3: How to apply shadows to UIBezierPath

Consider a simple rectangle that is drawn by the bezier path.

```objectivec
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

*Basic Outer-fill shadow:*



```objectivec
CGContextRef context = UIGraphicsGetCurrentContext();

NSShadow* shadow = [[NSShadow alloc] init];
[shadow setShadowColor: UIColor.blackColor];
[shadow setShadowOffset: CGSizeMake(7.1, 5.1)];
[shadow setShadowBlurRadius: 5];

UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];
CGContextSaveGState(context);
CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,
[shadow.shadowColor CGColor]);
[UIColor.grayColor setFill];
[rectanglePath fill];
CGContextRestoreGState(context);
```

*Basic Inner fill shadow:*



```objectivec
CGContextRef context = UIGraphicsGetCurrentContext();

NSShadow* shadow = [[NSShadow alloc] init];
[shadow setShadowColor: UIColor.blackColor];
[shadow setShadowOffset: CGSizeMake(9.1, -7.1)];
[shadow setShadowBlurRadius: 6];
```

```
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(x,y,width,height)];
[UIColor.grayColor setFill];
[rectanglePath fill];

CGContextSaveGState(context);
UIRectClip(rectanglePath.bounds);
CGContextSetShadowWithColor(context, CGSizeZero, 0, NULL);

CGContextSetAlpha(context, CGColorGetAlpha([shadow.shadowColor CGColor]));
CGContextBeginTransparencyLayer(context, NULL);
{
    UIColor* opaqueShadow = [shadow.shadowColor colorWithAlphaComponent: 1];
    CGContextSetShadowWithColor(context, shadow.shadowOffset, shadow.shadowBlurRadius,
[opaqueShadow CGColor]);
    CGContextSetBlendMode(context, kCGBlendModeSourceOut);
    CGContextBeginTransparencyLayer(context, NULL);

    [opaqueShadow setFill];
    [rectanglePath fill];

    CGContextEndTransparencyLayer(context);
}
CGContextEndTransparencyLayer(context);
CGContextRestoreGState(context);
```

# Section 53.4: How to create a simple shapes using UIBezierPath

*For a simple circle:*



```
UIBezierPath* ovalPath = [UIBezierPath bezierPathWithOvalInRect: CGRectMake(0,0,50,50)];
[UIColor.grayColor setFill];
[ovalPath fill];
```

Swift:

```
let ovalPath = UIBezierPath(ovalInRect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
ovalPath.fill()
```

*For a simple Rectangle:*

```objc
UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect: CGRectMake(0,0,50,50)];
[UIColor.grayColor setFill];
[rectanglePath fill];
```

Swift:

```swift
let rectanglePath = UIBezierPath(rect: CGRect(x: 0, y: 0, width: 50, height: 50))
UIColor.grayColor().setFill()
rectanglePath.fill()
```

*For a simple Line:*



```objc
UIBezierPath* bezierPath = [UIBezierPath bezierPath];
[bezierPath moveToPoint: CGPointMake(x1,y1)];
[bezierPath addLineToPoint: CGPointMake(x2,y2)];
[UIColor.blackColor setStroke];
bezierPath.lineWidth = 1;
[bezierPath stroke];
```

Swift:

```swift
let bezierPath = UIBezierPath()
bezierPath.moveToPoint(CGPoint(x: x1, y: y1))
bezierPath.addLineToPoint(CGPoint(x: x2, y: y2))
UIColor.blackColor().setStroke()
bezierPath.lineWidth = 1
bezierPath.stroke()
```

*For a half circle:*



```objc
 CGRect ovalRect = CGRectMake(x,y,width,height);
UIBezierPath* ovalPath = [UIBezierPath bezierPath];
[ovalPath addArcWithCenter: CGPointMake(0, 0) radius: CGRectGetWidth(ovalRect) / 2 startAngle: 180
* M_PI/180 endAngle: 0 * M_PI/180 clockwise: YES];
[ovalPath addLineToPoint: CGPointMake(0, 0)];
[ovalPath closePath];

CGAffineTransform ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect));
ovalTransform = CGAffineTransformScale(ovalTransform, 1, CGRectGetHeight(ovalRect) /
CGRectGetWidth(ovalRect));
```

```
[ovalPath applyTransform: ovalTransform];

[UIColor.grayColor setFill];
[ovalPath fill];
```

Swift:

```
let ovalRect = CGRect(x: 0, y: 0, width: 50, height: 50)
let ovalPath = UIBezierPath()
ovalPath.addArcWithCenter(CGPoint.zero, radius: ovalRect.width / 2, startAngle: 180 *
CGFloat(M_PI)/180, endAngle: 0 * CGFloat(M_PI)/180, clockwise: true)
ovalPath.addLineToPoint(CGPoint.zero)
ovalPath.closePath()

var ovalTransform = CGAffineTransformMakeTranslation(CGRectGetMidX(ovalRect),
CGRectGetMidY(ovalRect))
ovalTransform = CGAffineTransformScale(ovalTransform, 1, ovalRect.height / ovalRect.width)
ovalPath.applyTransform(ovalTransform)

UIColor.grayColor().setFill()
ovalPath.fill()
```

*For a simple triangle:*



```
UIBezierPath* polygonPath = [UIBezierPath bezierPath];
[polygonPath moveToPoint: CGPointMake(x1, y1)];
[polygonPath addLineToPoint: CGPointMake(x2, y2)];
[polygonPath addLineToPoint: CGPointMake(x3, y2)];
[polygonPath closePath];
[UIColor.grayColor setFill];
[polygonPath fill];
```

Swift:

```
let polygonPath = UIBezierPath()
polygonPath.moveToPoint(CGPoint(x: x1, y: y1))
polygonPath.addLineToPoint(CGPoint(x: x2, y: y2))
polygonPath.addLineToPoint(CGPoint(x: x3, y: y3))
polygonPath.closePath()
UIColor.grayColor().setFill()
polygonPath.fill()
```

# Section 53.5: UIBezierPath + AutoLayout

For bezier path to get resized based on the view frame, override the drawRect of view that you are drawing the
bezier path :

```
- (void)drawRect:(CGRect)frame
{
    UIBezierPath* rectanglePath = [UIBezierPath bezierPathWithRect:
CGRectMake(CGRectGetMinX(frame), CGRectGetMinY(frame), CGRectGetWidth(frame),
CGRectGetHeight(frame))];
```

```
    [UIColor.grayColor setFill];
    [rectanglePath fill];
}
```

# Section 53.6: pie view & column view with UIBezierPath

- pie view
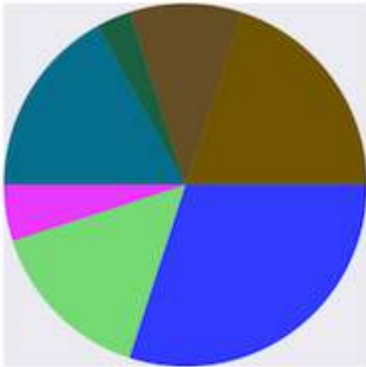


```objc
- (void)drawRect:(CGRect)rect {

    NSArray *data = @[@30, @15, @5, @17, @3, @10, @20];

    // 1. context
    CGContextRef cxtRef = UIGraphicsGetCurrentContext();

    CGPoint center = CGPointMake(150, 150);
    CGFloat radius = 150;
    __block CGFloat startAngle = 0;
    [data enumerateObjectsUsingBlock:^(NSNumber * _Nonnull obj, NSUInteger idx, BOOL * _Nonnull
stop) {

        // 2. create path
        CGFloat endAngle = obj.floatValue / 100 * M_PI * 2 + startAngle;
        UIBezierPath *circlePath = [UIBezierPath bezierPathWithArcCenter:center radius:radius
startAngle:startAngle endAngle:endAngle clockwise:YES];
        [circlePath addLineToPoint:center];

        // 3. add path
        CGContextAddPath(cxtRef, circlePath.CGPath);

        // set color
        [[UIColor colorWithRed:((float)arc4random_uniform(256) / 255.0)
green:((float)arc4random_uniform(256) / 255.0) blue:((float)arc4random_uniform(256) / 255.0)
alpha:1.0] setFill];

        // 4. render
        CGContextDrawPath(cxtRef, kCGPathFill);

        // reset angle
        startAngle = endAngle;
    }];
}
```

```swift
override func draw(_ rect: CGRect) {
    // define data to create pie chart
    let data: [Int] = [30, 15, 5, 17, 3, 10, 20]

    // 1. find center of draw rect
```

```swift
        let center: CGPoint = CGPoint(x: rect.midX, y: rect.midY)

        // 2. calculate radius of pie
        let radius = min(rect.width, rect.height) / 2.0

        var startAngle: CGFloat = 0.0
        for value in data {

            // 3. calculate end angle for slice
            let endAngle = CGFloat(value) / 100.0 * CGFloat.pi * 2.0 + startAngle

            // 4. create UIBezierPath for slide
            let circlePath = UIBezierPath(arcCenter: center, radius: radius, startAngle: startAngle,
endAngle: endAngle, clockwise: true)

            // 5. add line to center to close path
            circlePath.addLine(to: center)

            // 6. set fill color for current slice
            UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

            // 7. fill slice path
            circlePath.fill()

            // 8. set end angle as start angle for next slice
            startAngle = endAngle
        }
    }
```

- column view
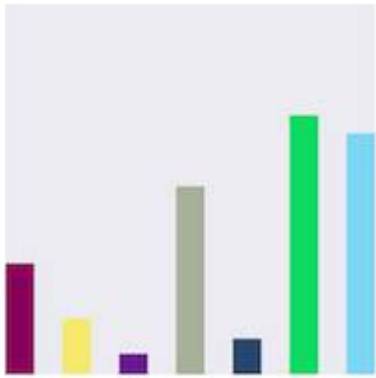


```objectivec
- (void)drawRect:(CGRect)rect {

    NSArray *data = @[@300, @150.65, @55.3, @507.7, @95.8, @700, @650.65];

    // 1.
    CGContextRef cxtRef = UIGraphicsGetCurrentContext();

    NSInteger columnCount = 7;
    CGFloat width = self.bounds.size.width / (columnCount + columnCount - 1);
    for (NSInteger i = 0; i < columnCount; i++) {

        // 2.
        CGFloat height = [data[i] floatValue] / 1000 * self.bounds.size.height;  // floatValue
        CGFloat x = 0 + width * (2 * i);
        CGFloat y = self.bounds.size.height - height;
        UIBezierPath *rectPath = [UIBezierPath bezierPathWithRect:CGRectMake(x, y, width, height)];
```

```
        CGContextAddPath(cxtRef, rectPath.CGPath);

        // 3.
        [[UIColor colorWithRed:((float)arc4random_uniform(256) / 255.0)
green:((float)arc4random_uniform(256) / 255.0) blue:((float)arc4random_uniform(256) / 255.0)
alpha:1.0] setFill];
        CGContextDrawPath(cxtRef, kCGPathFill);
    }
}

override func draw(_ rect: CGRect) {
    // define data for chart
    let data: [CGFloat] = [300, 150.65, 55.3, 507.7, 95.8, 700, 650.65]

    // 1. calculate number of columns
    let columnCount = data.count

    // 2. calculate column width
    let columnWidth = rect.width / CGFloat(columnCount + columnCount - 1)

    for (columnIndex, value) in data.enumerated() {
      // 3. calculate column height
      let columnHeight = value / 1000.0 * rect.height

      // 4. calculate column origin
      let columnOrigin = CGPoint(x: (columnWidth * 2.0 * CGFloat(columnIndex)), y: (rect.height -
columnHeight))

      // 5. create path for column
      let columnPath = UIBezierPath(rect: CGRect(origin: columnOrigin, size: CGSize(width:
columnWidth, height: columnHeight)))

      // 6. set fill color for current column
      UIColor(red: (CGFloat(arc4random_uniform(256)) / 255.0), green:
(CGFloat(arc4random_uniform(256)) / 255.0), blue: (CGFloat(arc4random_uniform(256)) / 255.0),
alpha: 1.0).setFill()

      // 7. fill column path
      columnPath.fill()
    }
  }
```

# Chapter 54: UIPickerView

## Section 54.1: Basic example

**Swift**

```swift
class PickerViewExampleViewController : UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource {
    @IBOutlet weak var btnFolder: UIButton!
    let pickerView = UIPickerView()
    let pickerViewRows = ["First row,", "Secound row,","Third row,","Fourth row"]

    override func viewDidLoad() {
        super.viewDidLoad()
        self.btnFolder.addTarget(self, action: #selector(CreateListVC.btnFolderPress),
forControlEvents: UIControlEvents.TouchUpInside)
    }

    @objc private func btnFolderPress() {
        self.pickerView.delegate = self
        self.pickerView.dataSource = self
        self.view.addSubview(self.pickerView)
    }

    //MARK: UIPickerViewDelegate

    func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) ->
String? {
        return self.pickerViewRows[row]
    }

    //MARK: UIPickerViewDataSource

    func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return self.pickerViewRows.count
    }

}
```

**Objective-C**

```objc
@property (nonatomic,strong) UIPickerView *countryPicker;
@property (nonatomic,strong) NSArray *countryNames;

- (void)viewDidLoad {
    [super viewDidLoad];
    _countryNames = @[@"Australia (AUD)", @"China (CNY)",
                      @"France (EUR)", @"Great Britain (GBP)", @"Japan (JPY)",@"INDIA
(IN)",@"AUSTRALIA (AUS)",@"NEW YORK (NW)"];

    [self pickcountry];
 }

-(void)pickcountry {
    _countryPicker = [[UIPickerView alloc]init];

    _countryPicker.delegate = self;
    _countryPicker.dataSource = self;
```

```
    [[UIPickerView appearance] setBackgroundColor:[UIColor colorWithRed:21/255.0 green:17/255.0
blue:50/255.0 alpha:1.0]];
}

#pragma mark-  pickerView Delegates And datasource

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component {
    return _countryNames.count;
}

- (NSString *)pickerView:(UIPickerView *)pickerView
            titleForRow:(NSInteger)row
          forComponent:(NSInteger)component {
    return _countryNames[row];
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger)component {
      NSString *pickedCountryName = _countryNames[row];
}
```

## Section 54.2: Changing pickerView Background Color and text color

**Objective-C**

```
//Displays the country pickerView with black background and white text
[self. countryPicker setValue:[UIColor whiteColor] forKey:@"textColor"];
[self. countryPicker setValue:[UIColor blackColor] forKey:@"backgroundColor"];
```

**Swift**

```
let color1 = UIColor(colorLiteralRed: 1, green: 1, blue: 1, alpha: 1)
let color2 = UIColor(colorLiteralRed: 0, green: 0, blue: 0, alpha: 1)
pickerView2.setValue(color1, forKey: "textColor")
pickerView2.setValue(color2, forKey: "backgroundColor")
```

# Chapter 55: UIFeedbackGenerator

`UIFeedbackGenerator` and its subclasses offers a public interface to the Taptic Engine® found on iOS devices starting with iPhone 7. Haptics, branded Taptics, provide tactile feedback for on-screen events. While many system controls provide haptics out-of-the-box, developers can use `UIFeedbackGenerator` subclasses to add haptics to custom controls and other events. `UIFeedbackGenerator` is an abstract class that should not be used directly, rather developers use one of its subclasses.

## Section 55.1: Trigger Impact Haptic

Example shows how to trigger an impact haptic using `UIImpactFeedbackGenerator` after a button press.

**Swift**

```swift
class ViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Impact", for: .normal)
        button.setTitleColor(UIColor.gray, for: .normal)
        return button
    }()

    // Choose between heavy, medium, and light for style
    let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .heavy)

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
.touchUpInside)

        // Primes feedback generator for upcoming events and reduces latency
        impactFeedbackGenerator.prepare()
    }

    func didPressButton(sender: UIButton)
    {
        // Triggers haptic
        impactFeedbackGenerator.impactOccurred()
    }
}
```

**Objective-C**

```objc
@interface ViewController ()
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;
@property (nonatomic, strong) UIButton *button;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressButton:)
```

```objc
forControlEvents:UIControlEventTouchUpInside];

    // Choose between heavy, medium, and light for style
    self.impactFeedbackGenerator = [[UIImpactFeedbackGenerator alloc]
initWithStyle:UIImpactFeedbackStyleHeavy];

    // Primes feedback generator for upcoming events and reduces latency
    [self.impactFeedbackGenerator prepare];
}

- (void)didPressButton:(UIButton *)sender
{
    // Triggers haptic
    [self.impactFeedbackGenerator impactOccurred];
}

#pragma mark - Lazy Init
- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc]init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Impact" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor grayColor] forState:UIControlStateNormal];
    }
    return _button;
}

@end
```

# Chapter 56: UIAppearance

## Section 56.1: Set appearance of all instances of the class

To customize appearance of all instances of a class, access appearance proxy of the desired class. For example:

**Set UIButton tint color**

*Swift:*

```
UIButton.appearance().tintColor = UIColor.greenColor()
```

*Objective-C:*

```
[UIButton appearance].tintColor = [UIColor greenColor];
```

**Set UIButton background color**

*Swift:*

```
UIButton.appearance().backgroundColor = UIColor.blueColor()
```

*Objective-C:*

```
[UIButton appearance].backgroundColor = [UIColor blueColor];
```

**Set UILabel text color**

*Swift:*

```
UILabel.appearance().textColor = UIColor.redColor()
```

*Objective-C:*

```
[UILabel appearance].textColor = [UIColor redColor];
```

**Set UILabel background color**

*Swift:*

```
UILabel.appearance().backgroundColor = UIColor.greenColor()
```

*Objective-C:*

```
[UILabel appearance].backgroundColor = [UIColor greenColor];
```

**Set UINavigationBar tint color**

*Swift:*

```
UINavigationBar.appearance().tintColor = UIColor.cyanColor()
```

*Objective-C:*

```
[UINavigationBar appearance].tintColor = [UIColor cyanColor];
```

**Set UINavigationBar background color**

*Swift:*

```
UINavigationBar.appearance().backgroundColor = UIColor.redColor()
```

*Objective-C:*

```
[UINavigationBar appearance].backgroundColor = [UIColor redColor];
```

# Section 56.2: Appearance for class when contained in container class

Use appearanceWhenContainedInInstancesOfClasses: to customize the appearance for instance of a class when contained within an instance of container class. For example customization of UILabel's textColor and backgroundColor within ViewController class will look like this:

**Set UILabel text color**

*Swift:*

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).textColor =
UIColor.whiteColor()
```

*Objective-C:*

```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].textColor =
[UIColor whiteColor];
```

**Set UILabel background color**

*Swift:*

```
UILabel.appearanceWhenContainedInInstancesOfClasses([ViewController.self]).backgroundColor =
UIColor.blueColor()
```

*Objective-C:*

```
[UILabel appearanceWhenContainedInInstancesOfClasses:@[[ViewController class]]].backgroundColor =
[UIColor blueColor];
```

# Chapter 57: UIKit Dynamics with UICollectionView

UIKit Dynamics is a physics engine integrated into UIKit. UIKit Dynamics offers a set of API that offers interoperability with a `UICollectionView` and `UICollectionViewLayout`

## Section 57.1: Creating a Custom Dragging Behavior with UIDynamicAnimator

This example shows how to create a custom dragging behavior by Subclassing `UIDynamicBehavior` and subclassing `UICollectionViewFlowLayout`. In the example, we have `UICollectionView` that allows for the selection of multiple items. Then with a long press gesture those items can be dragged in an elastic, "springy" animation driven by a `UIDynamicAnimator`.
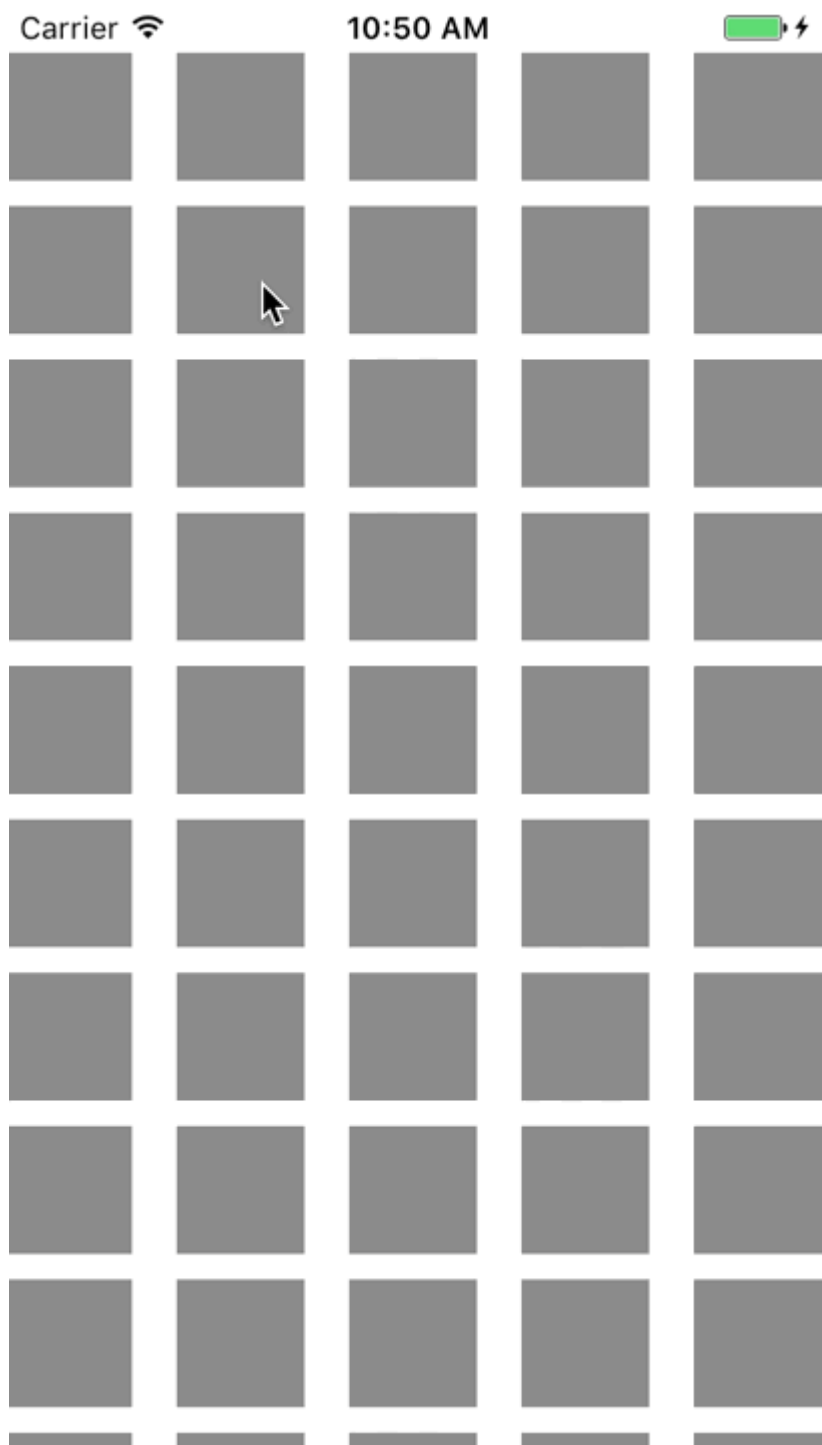
The dragging behavior is produced by combining a low-level behavior that adds a `UIAttachmentBehavior` to the for corners of a `UIDynamicItem` and a high-level behavior that manages the low-level behavior for a number of `UIDynamicItems`.

We can begin by creating this low-level behavior, we'll call `RectangleAttachmentBehavior`

**Swift**

```swift
final class RectangleAttachmentBehavior: UIDynamicBehavior
{
    init(item: UIDynamicItem, point: CGPoint)
    {
        // Higher frequency more "ridged" formation
        let frequency: CGFloat = 8.0

        // Lower damping longer animation takes to come to rest
        let damping: CGFloat = 0.6

        super.init()

        // Attachment points are four corners of item
        let points = self.attachmentPoints(for: point)

        let attachmentBehaviors: [UIAttachmentBehavior] = points.map
        {
            let attachmentBehavior = UIAttachmentBehavior(item: item, attachedToAnchor: $0)
            attachmentBehavior.frequency = frequency
            attachmentBehavior.damping = damping
            return attachmentBehavior
        }

        attachmentBehaviors.forEach
        {
            addChildBehavior($0)
        }
    }

    func updateAttachmentLocation(with point: CGPoint)
    {
        // Update anchor points to new attachment points
        let points = self.attachmentPoints(for: point)
        let attachments = self.childBehaviors.flatMap { $0 as? UIAttachmentBehavior }
        let pairs = zip(points, attachments)
        pairs.forEach { $0.1.anchorPoint = $0.0 }
    }

    func attachmentPoints(for point: CGPoint) -> [CGPoint]
    {
        // Width and height should be close to the width and height of the item
        let width: CGFloat = 40.0
        let height: CGFloat = 40.0

        let topLeft = CGPoint(x: point.x - width * 0.5, y: point.y - height * 0.5)
        let topRight = CGPoint(x: point.x + width * 0.5, y: point.y - height * 0.5)
        let bottomLeft = CGPoint(x: point.x - width * 0.5, y: point.y + height * 0.5)
        let bottomRight = CGPoint(x: point.x + width * 0.5, y: point.y + height * 0.5)
        let points = [topLeft, topRight, bottomLeft, bottomRight]
        return points
    }
}
```

**Objective-C**

```objc
@implementation RectangleAttachmentBehavior

- (instancetype)initWithItem:(id<UIDynamicItem>)item point:(CGPoint)point
{
    CGFloat frequency = 8.0f;
    CGFloat damping = 0.6f;
    self = [super init];
    if (self)
    {
        NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
        for (NSValue *value in pointValues)
        {
            UIAttachmentBehavior *attachment = [[UIAttachmentBehavior alloc]initWithItem:item
attachedToAnchor:[value CGPointValue]];
            attachment.frequency = frequency;
            attachment.damping = damping;
            [self addChildBehavior:attachment];
        }
    }
    return self;
}

- (void)updateAttachmentLocationWithPoint:(CGPoint)point
{
    NSArray <NSValue *> *pointValues = [self attachmentPointValuesForPoint:point];
    for (NSInteger i = 0; i < pointValues.count; i++)
    {
        NSValue *pointValue = pointValues[i];
        UIAttachmentBehavior *attachment = self.childBehaviors[i];
        attachment.anchorPoint = [pointValue CGPointValue];
    }
}

- (NSArray <NSValue *> *)attachmentPointValuesForPoint:(CGPoint)point
{
    CGFloat width = 40.0f;
    CGFloat height = 40.0f;

    CGPoint topLeft = CGPointMake(point.x - width * 0.5, point.y - height * 0.5);
    CGPoint topRight = CGPointMake(point.x + width * 0.5, point.y - height * 0.5);
    CGPoint bottomLeft = CGPointMake(point.x - width * 0.5, point.y + height * 0.5);
    CGPoint bottomRight = CGPointMake(point.x + width * 0.5, point.y + height * 0.5);

    NSArray <NSValue *> *pointValues = @[[NSValue valueWithCGPoint:topLeft], [NSValue
valueWithCGPoint:topRight], [NSValue valueWithCGPoint:bottomLeft], [NSValue
valueWithCGPoint:bottomRight]];
    return pointValues;
}

@end
```

Next we can create the high-level behavior that will combine a number of `RectangleAttachmentBehavior`.

**Swift**

```swift
final class DragBehavior: UIDynamicBehavior
{
    init(items: [UIDynamicItem], point: CGPoint)
    {
        super.init()
        items.forEach
        {
```

```
            let rectAttachment = RectangleAttachmentBehavior(item: $0, point: point)
            self.addChildBehavior(rectAttachment)
        }
    }

    func updateDragLocation(with point: CGPoint)
    {
        // Tell low-level behaviors location has changed
        self.childBehaviors.flatMap { $0 as? RectangleAttachmentBehavior }.forEach {
$0.updateAttachmentLocation(with: point) }
    }
}
```

**Objective-C**

```
@implementation DragBehavior

- (instancetype)initWithItems:(NSArray <id<UIDynamicItem>> *)items point: (CGPoint)point
{
    self = [super init];
    if (self)
    {
        for (id<UIDynamicItem> item in items)
        {
            RectangleAttachmentBehavior *rectAttachment = [[RectangleAttachmentBehavior
alloc]initWithItem:item point:point];
            [self addChildBehavior:rectAttachment];
        }
    }
    return self;
}

- (void)updateDragLocationWithPoint:(CGPoint)point
{
    for (RectangleAttachmentBehavior *rectAttachment in self.childBehaviors)
    {
        [rectAttachment updateAttachmentLocationWithPoint:point];
    }
}

@end
```

Now with our behaviors in place, the next step is to add them to our collection view when. Because normally we want a standard grid layout we can subclass UICollectionViewFlowLayout and only change attributes when dragging. We do this mainly through overriding layoutAttributesForElementsInRect and using the UIDynamicAnimator's convenience method itemsInRect.

**Swift**

```
final class DraggableLayout: UICollectionViewFlowLayout
{
    // Array that holds dragged index paths
    var indexPathsForDraggingElements: [IndexPath]?

    // The dynamic animator that will animate drag behavior
    var animator: UIDynamicAnimator?

    // Custom high-level behavior that dictates drag animation
    var dragBehavior: DragBehavior?

    // Where dragging starts so can return there once dragging ends
    var startDragPoint = CGPoint.zero
```

```swift
    // Bool to keep track if dragging has ended
    var isFinishedDragging = false


    // Method to inform layout that dragging has started
    func startDragging(indexPaths selectedIndexPaths: [IndexPath], from point: CGPoint)
    {
        indexPathsForDraggingElements = selectedIndexPaths
        animator = UIDynamicAnimator(collectionViewLayout: self)
        animator?.delegate = self

        // Get all of the draggable attributes but change zIndex so above other cells
        let draggableAttributes: [UICollectionViewLayoutAttributes] = selectedIndexPaths.flatMap {
            let attribute = super.layoutAttributesForItem(at: $0)
            attribute?.zIndex = 1
            return attribute
        }

        startDragPoint = point

        // Add them to high-level behavior
        dragBehavior = DragBehavior(items: draggableAttributes, point: point)

        // Add high-level behavior to animator
        animator?.addBehavior(dragBehavior!)
    }

    func updateDragLocation(_ point: CGPoint)
    {
        // Tell high-level behavior that point has updated
        dragBehavior?.updateDragLocation(with: point)
    }

    func endDragging()
    {
        isFinishedDragging = true

        // Return high-level behavior to starting point
        dragBehavior?.updateDragLocation(with: startDragPoint)
    }

    func clearDraggedIndexPaths()
    {
        // Reset state for next drag event
        animator = nil
        indexPathsForDraggingElements = nil
        isFinishedDragging = false
    }

    override func layoutAttributesForElements(in rect: CGRect) ->
[UICollectionViewLayoutAttributes]?
    {
        let existingAttributes: [UICollectionViewLayoutAttributes] =
super.layoutAttributesForElements(in: rect) ?? []
        var allAttributes = [UICollectionViewLayoutAttributes]()

        // Get normal flow layout attributes for non-drag items
        for attributes in existingAttributes
        {
            if (indexPathsForDraggingElements?.contains(attributes.indexPath) ?? false) == false
            {
                allAttributes.append(attributes)
```

```
            }
        }

        // Add dragged item attributes by asking animator for them
        if let animator = self.animator
        {
            let animatorAttributes: [UICollectionViewLayoutAttributes] = animator.items(in:
rect).flatMap { $0 as? UICollectionViewLayoutAttributes }
            allAttributes.append(contentsOf: animatorAttributes)
        }
        return allAttributes
    }
}
extension DraggableLayout: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        // Animator has paused and done dragging; reset state
        guard isFinishedDragging else { return }
        clearDraggedIndexPaths()
    }
}
```

**Objective-C**

```
@interface DraggableLayout () <UIDynamicAnimatorDelegate>
@property (nonatomic, strong) NSArray <NSIndexPath *> *indexPathsForDraggingElements;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) CGPoint startDragPoint;
@property (nonatomic, assign) BOOL finishedDragging;
@property (nonatomic, strong) DragBehavior *dragBehavior;
@end

@implementation DraggableLayout

- (void)startDraggingWithIndexPaths:(NSArray <NSIndexPath *> *)selectedIndexPaths
fromPoint:(CGPoint)point
{
    self.indexPathsForDraggingElements = selectedIndexPaths;
    self.animator = [[UIDynamicAnimator alloc]initWithCollectionViewLayout:self];
    self.animator.delegate = self;
    NSMutableArray *draggableAttributes = [[NSMutableArray
alloc]initWithCapacity:selectedIndexPaths.count];
    for (NSIndexPath *indexPath in selectedIndexPaths)
    {
        UICollectionViewLayoutAttributes *attributes = [super
layoutAttributesForItemAtIndexPath:indexPath];
        attributes.zIndex = 1;
        [draggableAttributes addObject:attributes];
    }
    self.startDragPoint = point;
    self.dragBehavior = [[DragBehavior alloc]initWithItems:draggableAttributes point:point];
    [self.animator addBehavior:self.dragBehavior];
}

- (void)updateDragLoactionWithPoint:(CGPoint)point
{
    [self.dragBehavior updateDragLocationWithPoint:point];
}

- (void)endDragging
{
    self.finishedDragging = YES;
```

```
        [self.dragBehavior updateDragLocationWithPoint:self.startDragPoint];
}

- (void)clearDraggedIndexPath
{
    self.animator = nil;
    self.indexPathsForDraggingElements = nil;
    self.finishedDragging = NO;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    if (self.finishedDragging)
    {
        [self clearDraggedIndexPath];
    }
}

- (NSArray<UICollectionViewLayoutAttributes *> *)layoutAttributesForElementsInRect:(CGRect)rect
{
    NSArray *existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [[NSMutableArray
alloc]initWithCapacity:existingAttributes.count];
    for (UICollectionViewLayoutAttributes *attributes in existingAttributes)
    {
        if (![self.indexPathsForDraggingElements containsObject:attributes.indexPath])
        {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[self.animator itemsInRect:rect]];
    return allAttributes;
}

@end
```

Finally, we'll create a view controller that will create our UICollectionView and handle our long press gesture.

**Swift**

```
final class ViewController: UIViewController
{
    // Collection view that displays cells
    lazy var collectionView: UICollectionView =
    {
        let collectionView = UICollectionView(frame: .zero, collectionViewLayout:
DraggableLayout())
        collectionView.backgroundColor = .white
        collectionView.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(collectionView)
        collectionView.topAnchor.constraint(equalTo: self.topLayoutGuide.bottomAnchor).isActive =
true
        collectionView.leadingAnchor.constraint(equalTo: self.view.leadingAnchor).isActive = true
        collectionView.trailingAnchor.constraint(equalTo: self.view.trailingAnchor).isActive = true
        collectionView.bottomAnchor.constraint(equalTo: self.bottomLayoutGuide.topAnchor).isActive
= true

        return collectionView
    }()

    // Gesture that drives dragging
    lazy var longPress: UILongPressGestureRecognizer =
```

```
    {
        let longPress = UILongPressGestureRecognizer(target: self, action:
#selector(self.handleLongPress(sender:)))
        return longPress
    }()

    // Array that holds selected index paths
    var selectedIndexPaths = [IndexPath]()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        collectionView.delegate = self
        collectionView.dataSource = self
        collectionView.register(UICollectionViewCell.self, forCellWithReuseIdentifier: "Cell")
        collectionView.addGestureRecognizer(longPress)
    }

    func handleLongPress(sender: UILongPressGestureRecognizer)
    {
        guard let draggableLayout = collectionView.collectionViewLayout as? DraggableLayout else {
return }
        let location = sender.location(in: collectionView)
        switch sender.state
        {
        case .began:
            draggableLayout.startDragging(indexPaths: selectedIndexPaths, from: location)
        case .changed:
            draggableLayout.updateDragLocation(location)
        case .ended, .failed, .cancelled:
            draggableLayout.endDragging()
        case .possible:
            break
        }
    }
}
extension ViewController: UICollectionViewDelegate, UICollectionViewDataSource
{
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) ->
Int
    {
        return 1000
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) ->
UICollectionViewCell
    {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "Cell", for: indexPath)
        cell.backgroundColor = .gray
        if selectedIndexPaths.contains(indexPath) == true
        {
            cell.backgroundColor = .red
        }
        return cell
    }

    func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath)
    {
        // Bool that determines if cell is being selected or unselected
        let isSelected = !selectedIndexPaths.contains(indexPath)
        let cell = collectionView.cellForItem(at: indexPath)
        cell?.backgroundColor = isSelected ? .red : .gray
```

```
        if isSelected
        {
            selectedIndexPaths.append(indexPath)
        }
        else
        {
            selectedIndexPaths.remove(at: selectedIndexPaths.index(of: indexPath)!)
        }
    }
}
```

**Objective-C**

```
@interface ViewController () <UICollectionViewDelegate, UICollectionViewDataSource>
@property (nonatomic, strong) UICollectionView *collectionView;
@property (nonatomic, strong) UILongPressGestureRecognizer *longPress;
@property (nonatomic, strong) NSMutableArray <NSIndexPath *> *selectedIndexPaths;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.collectionView.delegate = self;
    self.collectionView.dataSource = self;
    [self.collectionView registerClass:[UICollectionViewCell class]
forCellWithReuseIdentifier:@"Cell"];
    [self.collectionView addGestureRecognizer:self.longPress];
    self.selectedIndexPaths = [[NSMutableArray alloc]init];
}

- (UICollectionView *)collectionView
{
    if (!_collectionView)
    {
        _collectionView = [[UICollectionView alloc]initWithFrame:CGRectZero
collectionViewLayout:[[DraggableLayout alloc]init]];
        _collectionView.backgroundColor = [UIColor whiteColor];
        _collectionView.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_collectionView];
        [_collectionView.topAnchor constraintEqualToAnchor:self.topLayoutGuide.bottomAnchor].active
= YES;
        [_collectionView.leadingAnchor constraintEqualToAnchor:self.view.leadingAnchor].active =
YES;
        [_collectionView.trailingAnchor constraintEqualToAnchor:self.view.trailingAnchor].active =
YES;
        [_collectionView.bottomAnchor
constraintEqualToAnchor:self.bottomLayoutGuide.topAnchor].active = YES;
    }
    return _collectionView;
}

- (UILongPressGestureRecognizer *)longPress
{
    if (!_longPress)
    {
        _longPress = [[UILongPressGestureRecognizer alloc]initWithTarget:self
action:@selector(handleLongPress:)];
    }
    return _longPress;
}
```

```
- (void)handleLongPress:(UILongPressGestureRecognizer *)sender
{
    DraggableLayout *draggableLayout = (DraggableLayout *)self.collectionView.collectionViewLayout;
    CGPoint location = [sender locationInView:self.collectionView];
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        [draggableLayout startDraggingWithIndexPaths:self.selectedIndexPaths fromPoint:location];
    }
    else if(sender.state == UIGestureRecognizerStateChanged)
    {
        [draggableLayout updateDragLoactionWithPoint:location];
    }
    else if(sender.state == UIGestureRecognizerStateEnded ||  sender.state ==
UIGestureRecognizerStateCancelled || sender.state == UIGestureRecognizerStateFailed)
    {
        [draggableLayout endDragging];
    }
}

- (NSInteger)collectionView:(UICollectionView *)collectionView
numberOfItemsInSection:(NSInteger)section
{
    return 1000;
}

- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath
{
    UICollectionViewCell *cell = [collectionView dequeueReusableCellWithReuseIdentifier:@"Cell"
forIndexPath:indexPath];
    cell.backgroundColor = [UIColor grayColor];
    if ([self.selectedIndexPaths containsObject:indexPath])
    {
        cell.backgroundColor = [UIColor redColor];
    }
    return cell;
}

- (void)collectionView:(UICollectionView *)collectionView didSelectItemAtIndexPath:(NSIndexPath
*)indexPath
{
    BOOL isSelected = ![self.selectedIndexPaths containsObject:indexPath];
    UICollectionViewCell *cell = [collectionView cellForItemAtIndexPath:indexPath];
    if (isSelected)
    {
        cell.backgroundColor = [UIColor redColor];
        [self.selectedIndexPaths addObject:indexPath];
    }
    else
    {
        cell.backgroundColor = [UIColor grayColor];
        [self.selectedIndexPaths removeObject:indexPath];
    }
}

@end
```

For more information [2013 WWDC Session "Advanced Techniques with UIKit Dynamics"](#)

# Chapter 58: UIPheonix - easy, flexible, dynamic & highly scalable UI framework

Inspired by game development UIPheonix is a super easy, flexible, dynamic and highly scalable UI framework + concept for building reusable component/control-driven apps for macOS, iOS and tvOS. The same API apply for cross platform development! Think of it as using Lego blocks, you can use similar ones and move them around easy as pie.

https://github.com/MKGitHub/UIPheonix

## Section 58.1: Example UI Components

## iOS - Collection View



## iOS - Table View



## macOS - Collection View



## macOS - Table View



## tvOS - Collection View
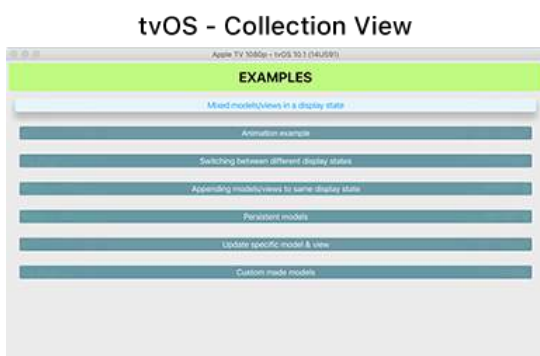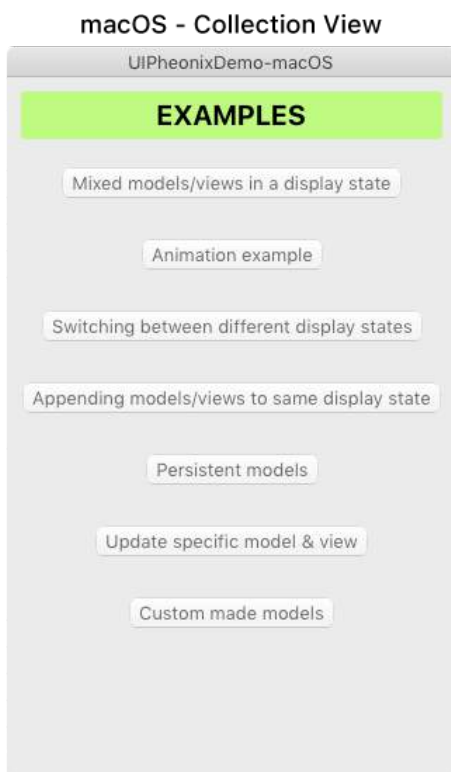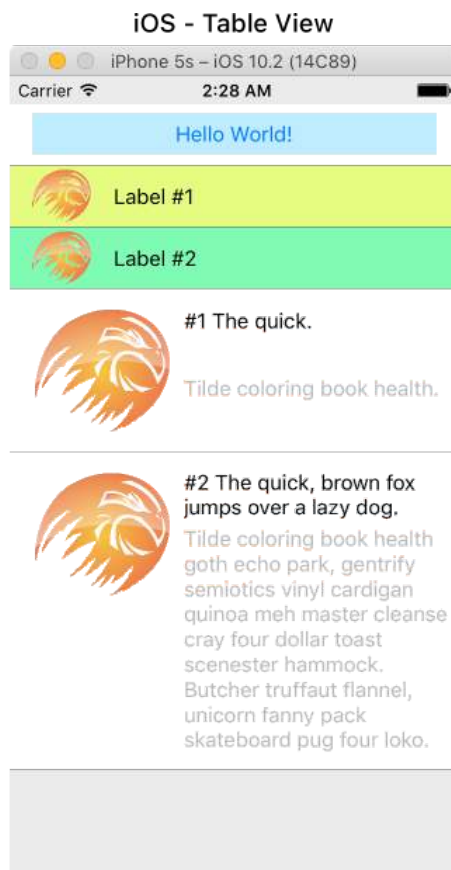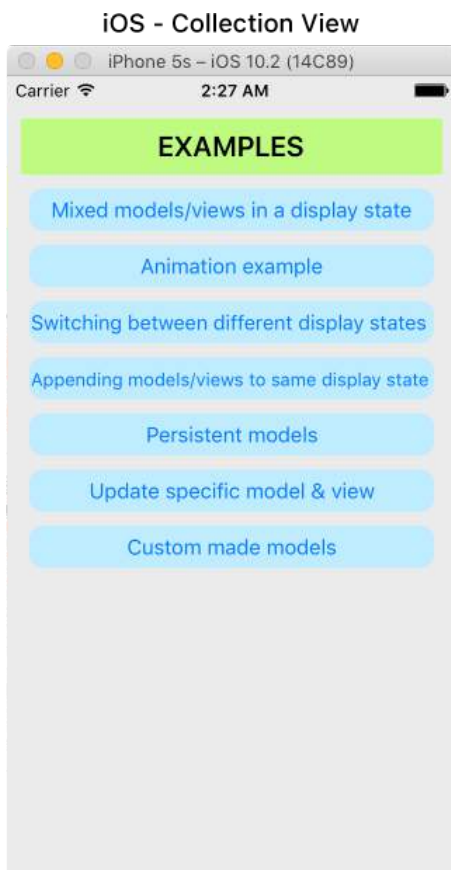


## tvOS - Table View

## Section 58.2: Example Usage

```
// init
mUIPheonix = UIPheonix(with:myCollectionView)
mUIPheonix = UIPheonix(with:myTableView)

// connect model-view
mUIPheonix.setModelViewRelationships([MyModel.nameOfClass:MyView.nameOfClass])

// add models for the UI
models.append(SimpleButtonModel(id:1, title:"Hello World!"))

// render, update UI
mUIPheonix.setDisplayModels(models)
```

# Chapter 59: UIKit Dynamics

UIKit Dynamics is a full real-world physics engine integrated into UIKit. It allows you to create interfaces that feel real by adding behaviors such as gravity, attachments, collision and forces. You define the physical traits that you would like your interface elements to adopt, and the dynamics engine takes care of the rest.

## Section 59.1: Flick View Based on Gesture Velocity

This example shows how to have a view track a pan gesture and depart in a physics-based manner.



**Swift**

```swift
class ViewController: UIViewController
{
    // Adjust to change speed of view from flick
    let magnitudeMultiplier: CGFloat = 0.0008

    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var gravity: UIGravityBehavior =
    {
        let gravity = UIGravityBehavior(items: [self.orangeView])
        return gravity
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height:
widthHeight))
        orangeView.backgroundColor = UIColor.orange
```

```swift
        self.view.addSubview(orangeView)
        return orangeView
    }()

    lazy var panGesture: UIPanGestureRecognizer =
    {
        let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
        return panGesture
    }()

    lazy var attachment: UIAttachmentBehavior =
    {
        let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
        return attachment
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        dynamicAnimator.addBehavior(gravity)
        dynamicAnimator.addBehavior(collision)
        orangeView.addGestureRecognizer(panGesture)
    }

    override func viewDidLayoutSubviews()
    {
        super.viewDidLayoutSubviews()
        orangeView.center = view.center
        dynamicAnimator.updateItem(usingCurrentState: orangeView)
    }

    func handlePan(sender: UIPanGestureRecognizer)
    {
        let location = sender.location(in: view)
        let velocity = sender.velocity(in: view)
        let magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y))
        switch sender.state
        {
        case .began:
            attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
            attachment.anchorPoint = location
        case .cancelled, .ended, .failed, .possible:
            let push = UIPushBehavior(items: [self.orangeView], mode: .instantaneous)
            push.pushDirection = CGVector(dx: velocity.x, dy: velocity.y)
            push.magnitude = magnitude * magnitudeMultiplier
            dynamicAnimator.removeBehavior(attachment)
            dynamicAnimator.addBehavior(push)
        }
    }
}
```

**Objective-C**

```objc
@interface ViewController ()

@property (nonatomic, assign) CGFloat magnitudeMultiplier;
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;
@property (nonatomic, strong) UIGravityBehavior *gravity;
@property (nonatomic, strong) UICollisionBehavior *collision;
@property (nonatomic, strong) UIView *orangeView;
```

```objc
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;
@property (nonatomic, strong) UIAttachmentBehavior *attachment;

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.dynamicAnimator addBehavior:self.gravity];
    [self.dynamicAnimator addBehavior:self.collision];
    [self.orangeView addGestureRecognizer:self.panGesture];
    // Adjust to change speed of view from flick
    self.magnitudeMultiplier = 0.0008f;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.orangeView.center = self.view.center;
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];
}

- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.view];
    CGPoint velocity = [sender velocityInView:self.view];
    CGFloat magnitude = sqrt((velocity.x * velocity.x) + (velocity.y * velocity.y));
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        self.attachment.anchorPoint = location;
        [self.dynamicAnimator addBehavior:self.attachment];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        self.attachment.anchorPoint = location;
    }
    else if (sender.state == UIGestureRecognizerStateCancelled ||
            sender.state == UIGestureRecognizerStateEnded ||
            sender.state == UIGestureRecognizerStateFailed ||
            sender.state == UIGestureRecognizerStatePossible)
    {
        UIPushBehavior *push = [[UIPushBehavior alloc] initWithItems:@[self.orangeView]
mode:UIPushBehaviorModeInstantaneous];
        push.pushDirection = CGVectorMake(velocity.x, velocity.y);
        push.magnitude = magnitude * self.magnitudeMultiplier;
        [self.dynamicAnimator removeBehavior:self.attachment];
        [self.dynamicAnimator addBehavior:push];
    }
}

#pragma mark - Lazy Init
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc]initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}
```

```objc
- (UIGravityBehavior *)gravity
{
    if (!_gravity)
    {
        _gravity = [[UIGravityBehavior alloc]initWithItems:@[self.orangeView]];
    }
    return _gravity;
}

- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc]initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}

- (UIView *)orangeView
{
    if (!_orangeView)
    {
        CGFloat widthHeight = 40.0f;
        _orangeView = [[UIView alloc]initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
        _orangeView.backgroundColor = [UIColor orangeColor];
        [self.view addSubview:_orangeView];
    }
    return _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    if (!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer alloc]initWithTarget:self
action:@selector(handlePan:)];
    }
    return _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    if (!_attachment)
    {
        _attachment = [[UIAttachmentBehavior alloc]initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    return _attachment;
}

@end
```

## Section 59.2: "Sticky Corners" Effect Using UIFieldBehaviors

This example shows how to achieve an effect similar to FaceTime were a view is attracted to point once it enters a particular region, in this case two regions a top and bottom.

**Swift**

```swift
class ViewController: UIViewController
{
    lazy var dynamicAnimator: UIDynamicAnimator =
    {
        let dynamicAnimator = UIDynamicAnimator(referenceView: self.view)
        return dynamicAnimator
    }()

    lazy var collision: UICollisionBehavior =
    {
        let collision = UICollisionBehavior(items: [self.orangeView])
        collision.translatesReferenceBoundsIntoBoundary = true
        return collision
    }()

    lazy var fieldBehaviors: [UIFieldBehavior] =
    {
        var fieldBehaviors = [UIFieldBehavior]()
        for _ in 0 ..< 2
        {
            let field = UIFieldBehavior.springField()
            field.addItem(self.orangeView)
            fieldBehaviors.append(field)
        }
        return fieldBehaviors
    }()

    lazy var itemBehavior: UIDynamicItemBehavior =
    {
        let itemBehavior = UIDynamicItemBehavior(items: [self.orangeView])
        // Adjust these values to change the "stickiness" of the view
        itemBehavior.density = 0.01
        itemBehavior.resistance = 10
        itemBehavior.friction = 0.0
        itemBehavior.allowsRotation = false
        return itemBehavior
    }()

    lazy var orangeView: UIView =
    {
        let widthHeight: CGFloat = 40.0
        let orangeView = UIView(frame: CGRect(x: 0.0, y: 0.0, width: widthHeight, height:
```

```
widthHeight))
        orangeView.backgroundColor = UIColor.orange
        self.view.addSubview(orangeView)
        return orangeView
    }()

    lazy var panGesture: UIPanGestureRecognizer =
    {
        let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.handlePan(sender:)))
        return panGesture
    }()

    lazy var attachment: UIAttachmentBehavior =
    {
        let attachment = UIAttachmentBehavior(item: self.orangeView, attachedToAnchor: .zero)
        return attachment
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        dynamicAnimator.addBehavior(collision)
        dynamicAnimator.addBehavior(itemBehavior)
        for field in fieldBehaviors
        {
            dynamicAnimator.addBehavior(field)
        }

        orangeView.addGestureRecognizer(panGesture)
    }

    override func viewDidLayoutSubviews()
    {
        super.viewDidLayoutSubviews()

        orangeView.center = view.center
        dynamicAnimator.updateItem(usingCurrentState: orangeView)

        for (index, field) in fieldBehaviors.enumerated()
        {
            field.position = CGPoint(x: view.bounds
                .midX, y:  view.bounds.height * (0.25 + 0.5 * CGFloat(index)))
            field.region = UIRegion(size: CGSize(width: view.bounds.width, height:
view.bounds.height * 0.5))
        }
    }

    func handlePan(sender: UIPanGestureRecognizer)
    {
        let location = sender.location(in: view)
        let velocity = sender.velocity(in: view)
        switch sender.state
        {
        case .began:
            attachment.anchorPoint = location
            dynamicAnimator.addBehavior(attachment)
        case .changed:
            attachment.anchorPoint = location
        case .cancelled, .ended, .failed, .possible:
            itemBehavior.addLinearVelocity(velocity, for: self.orangeView)
            dynamicAnimator.removeBehavior(attachment)
```

```
        }
    }
}
```

**Objective-C**

```objc
@interface ViewController ()

@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;
@property (nonatomic, strong) UICollisionBehavior *collision;
@property (nonatomic, strong) UIAttachmentBehavior *attachment;
@property (nonatomic, strong) UIDynamicItemBehavior *itemBehavior;
@property (nonatomic, strong) NSArray <UIFieldBehavior *> *fieldBehaviors;
@property (nonatomic, strong) UIView *orangeView;
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture;

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.dynamicAnimator addBehavior:self.collision];
    [self.dynamicAnimator addBehavior:self.itemBehavior];
    for (UIFieldBehavior *field in self.fieldBehaviors)
    {
        [self.dynamicAnimator addBehavior:field];
    }

    [self.orangeView addGestureRecognizer:self.panGesture];
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.orangeView.center = self.view.center;
    [self.dynamicAnimator updateItemUsingCurrentState:self.orangeView];

    for (NSInteger i = 0; i < self.fieldBehaviors.count; i++)
    {
        UIFieldBehavior *field = self.fieldBehaviors[i];
        field.position = CGPointMake(CGRectGetMidX(self.view.bounds),
CGRectGetHeight(self.view.bounds) * (0.25f + 0.5f * i));
        field.region = [[UIRegion alloc]initWithSize:CGSizeMake(CGRectGetWidth(self.view.bounds),
CGRectGetHeight(self.view.bounds) * 0.5)];
    }
}

- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.view];
    CGPoint velocity = [sender velocityInView:self.view];
    if (sender.state == UIGestureRecognizerStateBegan)
    {
        self.attachment.anchorPoint = location;
        [self.dynamicAnimator addBehavior:self.attachment];
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        self.attachment.anchorPoint = location;
    }
    else if (sender.state == UIGestureRecognizerStateCancelled ||
```

```objc
                sender.state == UIGestureRecognizerStateEnded ||
                sender.state == UIGestureRecognizerStateFailed ||
                sender.state == UIGestureRecognizerStatePossible)
    {
        [self.itemBehavior addLinearVelocity:velocity forItem:self.orangeView];
        [self.dynamicAnimator removeBehavior:self.attachment];
    }
}

#pragma mark - Lazy Init
- (UIDynamicAnimator *)dynamicAnimator
{
    if (!_dynamicAnimator)
    {
        _dynamicAnimator = [[UIDynamicAnimator alloc]initWithReferenceView:self.view];
    }
    return _dynamicAnimator;
}


- (UICollisionBehavior *)collision
{
    if (!_collision)
    {
        _collision = [[UICollisionBehavior alloc]initWithItems:@[self.orangeView]];
        _collision.translatesReferenceBoundsIntoBoundary = YES;
    }
    return _collision;
}


- (NSArray <UIFieldBehavior *> *)fieldBehaviors
{
    if (!_fieldBehaviors)
    {
        NSMutableArray *fields = [[NSMutableArray alloc]init];
        for (NSInteger i =  0; i < 2; i++)
        {
            UIFieldBehavior *field = [UIFieldBehavior springField];
            [field addItem:self.orangeView];
            [fields addObject:field];
        }
        _fieldBehaviors = fields;
    }
    return _fieldBehaviors;
}

- (UIDynamicItemBehavior *)itemBehavior
{
    if (!_itemBehavior)
    {
        _itemBehavior = [[UIDynamicItemBehavior alloc]initWithItems:@[self.orangeView]];
        // Adjust these values to change the "stickiness" of the view
        _itemBehavior.density = 0.01;
        _itemBehavior.resistance = 10;
        _itemBehavior.friction = 0.0;
        _itemBehavior.allowsRotation = NO;
    }
    return _itemBehavior;
}

- (UIView *)orangeView
{
    if (!_orangeView)
```

```
    {
        CGFloat widthHeight = 40.0f;
        _orangeView = [[UIView alloc]initWithFrame:CGRectMake(0.0, 0.0, widthHeight, widthHeight)];
        _orangeView.backgroundColor = [UIColor orangeColor];
        [self.view addSubview:_orangeView];
    }
    return _orangeView;
}

- (UIPanGestureRecognizer *)panGesture
{
    if (!_panGesture)
    {
        _panGesture = [[UIPanGestureRecognizer alloc]initWithTarget:self
action:@selector(handlePan:)];
    }
    return _panGesture;
}

- (UIAttachmentBehavior *)attachment
{
    if (!_attachment)
    {
        _attachment = [[UIAttachmentBehavior alloc]initWithItem:self.orangeView
attachedToAnchor:CGPointZero];
    }
    return _attachment;
}

@end
```

For more information about `UIFieldBehaviors` you can see the [2015 WWDC Session "What's New in UIKit Dynamics and Visual Effects"](#) and accompanying [sample code](#).

## Section 59.3: UIDynamicBehavior Driven Custom Transition
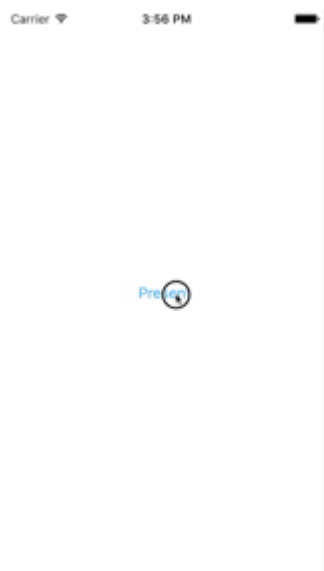


This example shows how to create a custom presentation transition that is driven by a composite `UIDynamicBehavior`. We can start by creating a presenting view controller that will present a modal.

**Swift**

```swift
class PresentingViewController: UIViewController
{
```

```swift
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
            = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Present", for: .normal)
        button.setTextColor(UIColor.blue, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressPresent), for: .touchUpInside)
    }

    func didPressPresent()
    {
        let modal = ModalViewController()
        modal.view.frame = CGRect(x: 0.0, y: 0.0, width: 200.0, height: 200.0)
        modal.modalPresentationStyle = .custom
        modal.transitioningDelegate = modal
        self.present(modal, animated: true)
    }
}
```

**Objective-C**

```objc
@interface PresentingViewController ()
@property (nonatomic, strong) UIButton *button;
@end

@implementation PresentingViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressPresent)
forControlEvents:UIControlEventTouchUpInside];
}

- (void)didPressPresent
{
    ModalViewController *modal = [[ModalViewController alloc] init];
    modal.view.frame = CGRectMake(0.0, 0.0, 200.0, 200.0);
    modal.modalPresentationStyle = UIModalPresentationCustom;
    modal.transitioningDelegate = modal;
    [self presentViewController:modal animated:YES completion:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
```

```
        [_button setTitle:@"Present" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    }
    return _button;
}

@end
```

When the present button is tapped, we create a `ModalViewController` and set its presentation style to `.custom` and set its `transitionDelegate` to itself. This will allow us to vend an animator that will drive its modal transition. We also set `modal`'s view's frame so it will be smaller than the full screen.

Let's now look at `ModalViewController`:

**Swift**

```swift
class ModalViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton()
        button.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(button)
        button.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive
         = true
        button.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        button.setTitle("Dismiss", for: .normal)
        button.setTitleColor(.white, for: .normal)

        return button
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        button.addTarget(self, action: #selector(self.didPressDismiss), for: .touchUpInside)
        view.backgroundColor = .red
        view.layer.cornerRadius = 15.0
    }

    func didPressDismiss()
    {
        dismiss(animated: true)
    }
}

extension ModalViewController: UIViewControllerTransitioningDelegate
{
    func animationController(forPresented presented: UIViewController, presenting:
UIViewController, source: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 1.5, isAppearing: true)
    }

    func animationController(forDismissed dismissed: UIViewController) ->
UIViewControllerAnimatedTransitioning?
    {
        return DropOutAnimator(duration: 4.0, isAppearing: false)
    }
}
```

**Objective-C**

---

```objc
@interface ModalViewController () <UIViewControllerTransitioningDelegate>
@property (nonatomic, strong) UIButton *button;
@end

@implementation ModalViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.button addTarget:self action:@selector(didPressPresent)
forControlEvents:UIControlEventTouchUpInside];
    self.view.backgroundColor = [UIColor redColor];
    self.view.layer.cornerRadius = 15.0f;
}

- (void)didPressPresent
{
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc] init];
        _button.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_button];
        [_button.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_button.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
        [_button setTitle:@"Dismiss" forState:UIControlStateNormal];
        [_button setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    }
    return _button;
}

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[DropOutAnimator alloc]initWithDuration: 1.5 appearing:YES];
}

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewControl
ler *)dismissed
{
    return [[DropOutAnimator alloc] initWithDuration:4.0 appearing:NO];
}

@end
```

Here we create the view controller that is presented. Also because `ModalViewController` is it's own
`transitioningDelegate` it is also responsible for vending an object that will manage its transition animation. For us
that means passing on an instance of our composite `UIDynamicBehavior` subclass.

Our animator will have two different transitions: one for presenting and one for dismissing. For presenting, the
presenting view controller's view will drop in from above. And for dismissing, the view will seem to swing from a
rope and then drop out. Because `DropOutAnimator` conforms to `UIViewControllerAnimatedTransitioning` most of
this work will be done in its implementation of `func animateTransition(using transitionContext:`

UIViewControllerContextTransitioning).

**Swift**

```swift
class DropOutAnimator: UIDynamicBehavior
{
    let duration: TimeInterval
    let isAppearing: Bool

    var transitionContext: UIViewControllerContextTransitioning?
    var hasElapsedTimeExceededDuration = false
    var finishTime: TimeInterval = 0.0
    var collisionBehavior: UICollisionBehavior?
    var attachmentBehavior: UIAttachmentBehavior?
    var animator: UIDynamicAnimator?

    init(duration: TimeInterval = 1.0,  isAppearing: Bool)
    {
        self.duration = duration
        self.isAppearing = isAppearing
        super.init()
    }
}

extension DropOutAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {
        // Get relevant views and view controllers from transitionContext
        guard let fromVC = transitionContext.viewController(forKey: .from),
            let toVC = transitionContext.viewController(forKey: .to),
            let fromView = fromVC.view,
            let toView = toVC.view else { return }

        let containerView = transitionContext.containerView
        let duration = self.transitionDuration(using: transitionContext)

        // Hold refrence to transitionContext to notify it of completion
        self.transitionContext = transitionContext

        // Create dynamic animator
        let animator = UIDynamicAnimator(referenceView: containerView)
        animator.delegate = self
        self.animator = animator

        // Presenting Animation
        if self.isAppearing
        {
            fromView.isUserInteractionEnabled = false

            // Position toView  just off-screen
            let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
            var toViewInitialFrame = toView.frame
            toViewInitialFrame.origin.y -= toViewInitialFrame.height
            toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
toViewInitialFrame.width * 0.5
            toView.frame = toViewInitialFrame

            containerView.addSubview(toView)

            // Prevent rotation and adjust bounce
            let bodyBehavior = UIDynamicItemBehavior(items: [toView])
```

```
            bodyBehavior.elasticity = 0.7
            bodyBehavior.allowsRotation = false

            // Add gravity at exaggerated magnitude so animation doesn't seem slow
            let gravityBehavior = UIGravityBehavior(items: [toView])
            gravityBehavior.magnitude = 10.0

            // Set collision bounds to include off-screen view and have collision in center
            // where our final view should come to rest
            let collisionBehavior = UICollisionBehavior(items: [toView])
            let insets = UIEdgeInsets(top: toViewInitialFrame.minY, left: 0.0, bottom:
fromViewInitialFrame.height * 0.5 - toViewInitialFrame.height * 0.5, right: 0.0)
            collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
            self.collisionBehavior = collisionBehavior

            // Keep track of finish time in case we need to end the animator befor the animator
pauses
            self.finishTime = duration + (self.animator?.elapsedTime ?? 0.0)

            // Closure that is called after every "tick" of the animator
            // Check if we exceed duration
            self.action =
            { [weak self] in
                guard let strongSelf = self,
                  (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}

                strongSelf.hasElapsedTimeExceededDuration = true
                strongSelf.animator?.removeBehavior(strongSelf)
            }

            // `DropOutAnimator` is a composit behavior, so add child behaviors to self
            self.addChildBehavior(collisionBehavior)
            self.addChildBehavior(bodyBehavior)
            self.addChildBehavior(gravityBehavior)

            // Add self to dynamic animator
            self.animator?.addBehavior(self)
        }
        // Dismissing Animation
        else
        {
            // Create allow rotation and have a elastic item
            let bodyBehavior = UIDynamicItemBehavior(items: [fromView])
            bodyBehavior.elasticity = 0.8
            bodyBehavior.angularResistance = 5.0
            bodyBehavior.allowsRotation = true

            // Create gravity with exaggerated magnitude
            let gravityBehavior = UIGravityBehavior(items: [fromView])
            gravityBehavior.magnitude = 10.0

            // Collision boundary is set to have a floor just below the bottom of the screen
            let collisionBehavior = UICollisionBehavior(items: [fromView])
            let insets = UIEdgeInsets(top: 0.0, left: -1000, bottom: -225, right: -1000)
            collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
            self.collisionBehavior = collisionBehavior

            // Attachment behavior so view will have effect of hanging from a rope
            let offset = UIOffset(horizontal: 70.0, vertical: fromView.bounds.height * 0.5)
            var anchorPoint = CGPoint(x: fromView.bounds.maxX - 40.0, y: fromView.bounds.minY)
            anchorPoint = containerView.convert(anchorPoint, from: fromView)
            let attachmentBehavior = UIAttachmentBehavior(item: fromView, offsetFromCenter: offset,
```

```
attachedToAnchor: anchorPoint)
            attachmentBehavior.frequency = 3.0
            attachmentBehavior.damping = 3.0
            self.attachmentBehavior = attachmentBehavior

            // `DropOutAnimator` is a composit behavior, so add child behaviors to self
            self.addChildBehavior(collisionBehavior)
            self.addChildBehavior(bodyBehavior)
            self.addChildBehavior(gravityBehavior)
            self.addChildBehavior(attachmentBehavior)

            // Add self to dynamic animator
            self.animator?.addBehavior(self)

            // Animation has two parts part one is hanging from rope.
            // Part two is bouncying off-screen
            // Divide duration in two
            self.finishTime = (2.0 / 3.0) * duration + (self.animator?.elapsedTime ?? 0.0)

             // After every "tick" of animator check if past time limit
            self.action =
            { [weak self] in
                guard let strongSelf = self,
                  (strongSelf.animator?.elapsedTime ?? 0.0) >= strongSelf.finishTime else { return
}
                strongSelf.hasElapsedTimeExceededDuration = true
                strongSelf.animator?.removeBehavior(strongSelf)
            }
        }

    }

    func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
    {
        // Return the duration of the animation
        return self.duration
    }
}

extension DropOutAnimator: UIDynamicAnimatorDelegate
{
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
      // Animator has reached stasis
      if self.isAppearing
      {
            // Check if we are out of time
            if self.hasElapsedTimeExceededDuration
            {
                // Move to final positions
                let toView = self.transitionContext?.viewController(forKey: .to)?.view
                let containerView = self.transitionContext?.containerView
                toView?.center = containerView?.center ?? .zero
                self.hasElapsedTimeExceededDuration = false
            }

            // Clean up and call completion

self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
            self.childBehaviors.forEach { self.removeChildBehavior($0) }
```

```swift
            animator.removeAllBehaviors()
            self.transitionContext = nil
    }
    else
    {
        if let attachmentBehavior = self.attachmentBehavior
        {
            // If we have an attachment, we are at the end of part one and start part two.
            self.removeChildBehavior(attachmentBehavior)
            self.attachmentBehavior = nil
            animator.addBehavior(self)
            let duration = self.transitionDuration(using: self.transitionContext)
            self.finishTime = 1.0 / 3.0 * duration + animator.elapsedTime
        }
        else
        {
            // Clean up and call completion
            let fromView = self.transitionContext?.viewController(forKey: .from)?.view
            let toView = self.transitionContext?.viewController(forKey: .to)?.view
            fromView?.removeFromSuperview()
            toView?.isUserInteractionEnabled = true

self.transitionContext?.completeTransition(!(self.transitionContext?.transitionWasCancelled ??
false))
            self.childBehaviors.forEach { self.removeChildBehavior($0) }
            animator.removeAllBehaviors()
            self.transitionContext = nil
        }
    }
}
}
```

**Objective-C**

```objc
@interface ObjcDropOutAnimator() <UIDynamicAnimatorDelegate, UIViewControllerAnimatedTransitioning>
@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, assign) BOOL elapsedTimeExceededDuration;
@property (nonatomic, assign, getter=isAppearing) BOOL appearing;
@property (nonatomic, assign) NSTimeInterval duration;
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;
@property (nonatomic, strong) UICollisionBehavior * collisionBehavior;

@end

@implementation ObjcDropOutAnimator

- (instancetype)initWithDuration:(NSTimeInterval)duration appearing:(BOOL)appearing
{
    self = [super init];
    if (self)
    {
        _duration = duration;
        _appearing = appearing;
    }
    return self;
}

- (void) animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    // Get relevant views and view controllers from transitionContext
    UIViewController *fromVC = [transitionContext
```

```
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *fromView = fromVC.view;
    UIView *toView = toVC.view;

    UIView *containerView = transitionContext.containerView;
    NSTimeInterval duration = [self transitionDuration:transitionContext];

    // Hold refrence to transitionContext to notify it of completion
    self.transitionContext = transitionContext;

    // Create dynamic animator
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc]initWithReferenceView:containerView];
    animator.delegate = self;
    self.animator = animator;

    // Presenting Animation
    if (self.isAppearing)
    {
        fromView.userInteractionEnabled = NO;

        // Position toView  just above screen
        CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
        CGRect toViewInitialFrame = toView.frame;
        toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
        toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
CGRectGetWidth(toViewInitialFrame) * 0.5;
        toView.frame = toViewInitialFrame;

        [containerView addSubview:toView];

        // Prevent rotation and adjust bounce
        UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior
alloc]initWithItems:@[toView]];
        bodyBehavior.elasticity = 0.7;
        bodyBehavior.allowsRotation = NO;

        // Add gravity at exaggerated magnitude so animation doesn't seem slow
        UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc]initWithItems:@[toView]];
        gravityBehavior.magnitude = 10.0f;

        // Set collision bounds to include off-screen view and have collision floor in center
        // where our final view should come to rest
        UICollisionBehavior *collisionBehavior = [[UICollisionBehavior
alloc]initWithItems:@[toView]];
        UIEdgeInsets insets = UIEdgeInsetsMake(CGRectGetMinY(toViewInitialFrame), 0.0,
CGRectGetHeight(fromViewInitialFrame) * 0.5 - CGRectGetHeight(toViewInitialFrame) * 0.5, 0.0);
        [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
        self.collisionBehavior = collisionBehavior;

        // Keep track of finish time in case we need to end the animator befor the animator pauses
        self.finishTime = duration + self.animator.elapsedTime;

        // Closure that is called after every "tick" of the animator
        // Check if we exceed duration
        __weak ObjcDropOutAnimator *weakSelf = self;
        self.action = ^{
            __strong ObjcDropOutAnimator *strongSelf = weakSelf;
            if (strongSelf)
            {
                if (strongSelf.animator.elapsedTime >= strongSelf.finishTime)
```

```objc
                {
                    strongSelf.elapsedTimeExceededDuration = YES;
                    [strongSelf.animator removeBehavior:strongSelf];
                }
            }
        };

        // `DropOutAnimator` is a composit behavior, so add child behaviors to self
        [self addChildBehavior:collisionBehavior];
        [self addChildBehavior:bodyBehavior];
        [self addChildBehavior:gravityBehavior];

        // Add self to dynamic animator
        [self.animator addBehavior:self];
    }
    // Dismissing Animation
    else
    {
        // Allow rotation and have a elastic item
        UIDynamicItemBehavior *bodyBehavior = [[UIDynamicItemBehavior alloc]
initWithItems:@[fromView]];
        bodyBehavior.elasticity = 0.8;
        bodyBehavior.angularResistance = 5.0;
        bodyBehavior.allowsRotation = YES;

        // Create gravity with exaggerated magnitude
        UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc] initWithItems:@[fromView]];
        gravityBehavior.magnitude = 10.0f;

        // Collision boundary is set to have a floor just below the bottom of the screen
        UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc]
initWithItems:@[fromView]];
        UIEdgeInsets insets = UIEdgeInsetsMake(0, -1000, -225, -1000);
        [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
        self.collisionBehavior = collisionBehavior;

        // Attachment behavior so view will have effect of hanging from a rope
        UIOffset offset = UIOffsetMake(70, -(CGRectGetHeight(fromView.bounds) / 2.0));

        CGPoint anchorPoint = CGPointMake(CGRectGetMaxX(fromView.bounds) - 40,
                                          CGRectGetMinY(fromView.bounds));
        anchorPoint = [containerView convertPoint:anchorPoint fromView:fromView];
        UIAttachmentBehavior *attachBehavior = [[UIAttachmentBehavior alloc] initWithItem:fromView
offsetFromCenter:offset attachedToAnchor:anchorPoint];
        attachBehavior.frequency = 3.0;
        attachBehavior.damping = 0.3;
        attachBehavior.length = 40;
        self.attachBehavior = attachBehavior;

        // `DropOutAnimator` is a composit behavior, so add child behaviors to self
        [self addChildBehavior:collisionBehavior];
        [self addChildBehavior:bodyBehavior];
        [self addChildBehavior:gravityBehavior];
        [self addChildBehavior:attachBehavior];

        // Add self to dynamic animator
        [self.animator addBehavior:self];

        // Animation has two parts part one is hanging from rope.
        // Part two is bouncying off-screen
        // Divide duration in two
        self.finishTime = (2./3.) * duration + [self.animator elapsedTime];
```

```objc
        // After every "tick" of animator check if past time limit
        __weak ObjcDropOutAnimator *weakSelf = self;
        self.action = ^{
            __strong ObjcDropOutAnimator *strongSelf = weakSelf;
            if (strongSelf)
            {
                if ([strongSelf.animator elapsedTime] >= strongSelf.finishTime)
                {
                    strongSelf.elapsedTimeExceededDuration = YES;
                    [strongSelf.animator removeBehavior:strongSelf];
                }
            }
        };
    }
}

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return self.duration;
}

- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    // Animator has reached stasis
    if (self.isAppearing)
    {
        // Check if we are out of time
        if (self.elapsedTimeExceededDuration)
        {
            // Move to final positions
            UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
            UIView *containerView = [self.transitionContext containerView];
            toView.center = containerView.center;
            self.elapsedTimeExceededDuration = NO;
        }

        // Clean up and call completion
        [self.transitionContext completeTransition:![self.transitionContext
transitionWasCancelled]];
        for (UIDynamicBehavior *behavior in self.childBehaviors)
        {
            [self removeChildBehavior:behavior];
        }
        [animator removeAllBehaviors];
        self.transitionContext = nil;
    }
    // Dismissing
    else
    {
        if (self.attachBehavior)
        {
            // If we have an attachment, we are at the end of part one and start part two.
            [self removeChildBehavior:self.attachBehavior];
            self.attachBehavior = nil;
            [animator addBehavior:self];
            NSTimeInterval duration = [self transitionDuration:self.transitionContext];
            self.finishTime = 1./3. * duration + [animator elapsedTime];
        }
        else
        {
```

```
            // Clean up and call completion
            UIView *fromView = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey].view;
            UIView *toView = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey].view;
            [fromView removeFromSuperview];
            toView.userInteractionEnabled = YES;

            [self.transitionContext completeTransition:![self.transitionContext
transitionWasCancelled]];
            for (UIDynamicBehavior *behavior in self.childBehaviors)
            {
                [self removeChildBehavior:behavior];
            }
            [animator removeAllBehaviors];
            self.transitionContext = nil;
        }
    }
}
```

As composite behavior, `DropOutAnimator`, can combine a number of different behaviors to perform its presenting and dismissing animations. `DropOutAnimator` also demonstrates how to use the `action` block of a behavior to inspect the locations of its items as well as the time elapsed a technique that can be used to remove views that move offscreen or truncate animations that have yet to reach stasis.

For more information [2013 WWDC Session "Advanced Techniques with UIKit Dynamics"](#) as well as [SOLPresentingFun](#)

## Section 59.4: Shade Transition with Real-World Physics Using UIDynamicBehaviors

This example shows how to make an interactive presentation transition with "real-world" physics similar to iOS's notifications screen.
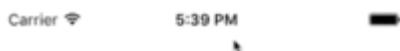


To start with, we need a presenting view controller that the shade will appear over. This view controller will also act as our `UIViewControllerTransitioningDelegate` for our presented view controller and will vend animators for our

transition. So we'll create instances of our interactive animators (one for presenting, one for dismissing). We'll also create an instance of the shade view controller, which, in this example, is just a view controller with a label. Because we want the same pan gesture to drive the entire interaction we pass references to the presenting view controller and the shade into our interactive animators.

**Swift**

```swift
class ViewController: UIViewController
{
    var presentingAnimator: ShadeAnimator!
    var dismissingAnimator: ShadeAnimator!
    let shadeVC = ShadeViewController()

    lazy var label: UILabel =
    {
        let label = UILabel()
        label.textColor = .blue
        label.translatesAutoresizingMaskIntoConstraints = false
        self.view.addSubview(label)
        label.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).isActive = true
        label.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).isActive = true
        return label
    }()

    override func viewDidLoad()
    {
        super.viewDidLoad()
        label.text = "Swipe Down From Top"
        presentingAnimator = ShadeAnimator(isAppearing: true, presentingVC: self, presentedVC:
shadeVC, transitionDelegate: self)
        dismissingAnimator = ShadeAnimator(isAppearing: false, presentingVC: self, presentedVC:
shadeVC, transitionDelegate: self)
    }
}
extension ViewController: UIViewControllerTransitioningDelegate
{
    func animationController(forPresented presented: UIViewController, presenting:
UIViewController, source: UIViewController) -> UIViewControllerAnimatedTransitioning?
    {
        return EmptyAnimator()
    }

    func animationController(forDismissed dismissed: UIViewController) ->
UIViewControllerAnimatedTransitioning?
    {
        return EmptyAnimator()
    }

    func interactionControllerForPresentation(using animator:
UIViewControllerAnimatedTransitioning) -> UIViewControllerInteractiveTransitioning?
    {
        return presentingAnimator
    }

    func interactionControllerForDismissal(using animator: UIViewControllerAnimatedTransitioning)
-> UIViewControllerInteractiveTransitioning?
    {
        return dismissingAnimator
    }
}
```

**Objective-C**

```objc
@interface ObjCViewController () <UIViewControllerTransitioningDelegate>
@property (nonatomic, strong) ShadeAnimator *presentingAnimator;
@property (nonatomic, strong) ShadeAnimator *dismissingAnimator;
@property (nonatomic, strong) UILabel *label;
@property (nonatomic, strong) ShadeViewController *shadeVC;
@end

@implementation ObjCViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.label.text = @"Swipe Down From Top";
    self.shadeVC = [[ShadeViewController alloc] init];
    self.presentingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:YES presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
    self.dismissingAnimator = [[ShadeAnimator alloc] initWithIsAppearing:NO presentingVC:self
presentedVC:self.shadeVC transitionDelegate:self];
}

- (UILabel *)label
{
    if (!_label)
    {
        _label = [[UILabel alloc] init];
        _label.textColor = [UIColor blueColor];
        _label.translatesAutoresizingMaskIntoConstraints = NO;
        [self.view addSubview:_label];
        [_label.centerXAnchor constraintEqualToAnchor:self.view.centerXAnchor].active = YES;
        [_label.centerYAnchor constraintEqualToAnchor:self.view.centerYAnchor].active = YES;
    }
    return _label;
}

#pragma mark - UIViewControllerTransitioningDelegate

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForPresentedController:(UIViewControl
ler *)presented presentingController:(UIViewController *)presenting
sourceController:(UIViewController *)source
{
    return [[EmptyAnimator alloc] init];
}

-
(id<UIViewControllerAnimatedTransitioning>)animationControllerForDismissedController:(UIViewControl
ler *)dismissed
{
    return [[EmptyAnimator alloc] init];
}

-
(id<UIViewControllerInteractiveTransitioning>)interactionControllerForPresentation:(id<UIViewContro
llerAnimatedTransitioning>)animator
{
    return self.presentingAnimator;
}

-
(id<UIViewControllerInteractiveTransitioning>)interactionControllerForDismissal:(id<UIViewControlle
rAnimatedTransitioning>)animator
{
```

```
    return self.dismissingAnimator;
}

@end
```

We want really only ever want to present our shade through an interactive transition but because of how
UIViewControllerTransitioningDelegate works if we don't return a regular animation controller our interactive
controller will never be used. Because of that we create an EmptyAnimator class that conforms to
UIViewControllerAnimatedTransitioning.

**Swift**

```
class EmptyAnimator: NSObject
{

}

extension EmptyAnimator: UIViewControllerAnimatedTransitioning
{
    func animateTransition(using transitionContext: UIViewControllerContextTransitioning)
    {

    }

    func transitionDuration(using transitionContext: UIViewControllerContextTransitioning?) ->
TimeInterval
    {
        return 0.0
    }
}
```

**Objective-C**

```
@implementation EmptyAnimator

- (void)animateTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{

}

- (NSTimeInterval)transitionDuration:(id<UIViewControllerContextTransitioning>)transitionContext
{
    return 0.0;
}

@end
```

Finally we need to actually create the ShadeAnimator which is a subclass of UIDynamicBehavior which conforms to
UIViewControllerInteractiveTransitioning.

**Swift**

```
class ShadeAnimator: UIDynamicBehavior
{
    // Whether we are presenting or dismissing
    let isAppearing: Bool

    // The view controller that is not the shade
    weak var presentingVC: UIViewController?

    // The view controller that is the shade
    weak var presentedVC: UIViewController?
```

```
    // The delegate will vend the animator
    weak var transitionDelegate: UIViewControllerTransitioningDelegate?

    // Feedback generator for haptics on collisions
    let impactFeedbackGenerator = UIImpactFeedbackGenerator(style: .light)

    // The context given to the animator at the start of the transition
    var transitionContext: UIViewControllerContextTransitioning?

    // Time limit of the dynamic part of the animation
    var finishTime: TimeInterval = 4.0

    // The Pan Gesture that drives the transition. Not using EdgePan because triggers Notifications
screen
    lazy var pan: UIPanGestureRecognizer =
    {
        let pan = UIPanGestureRecognizer(target: self, action: #selector(self.handlePan(sender:)))
        return pan
    }()

    // The dynamic animator that we add `ShadeAnimator` to
    lazy var animator: UIDynamicAnimator! =
    {
        let animator = UIDynamicAnimator(referenceView: self.transitionContext!.containerView)
        return animator
    }()

    // init with all of our dependencies
    init(isAppearing: Bool, presentingVC: UIViewController, presentedVC: UIViewController,
transitionDelegate: UIViewControllerTransitioningDelegate)
    {
        self.isAppearing = isAppearing
        self.presentingVC = presentingVC
        self.presentedVC = presentedVC
        self.transitionDelegate = transitionDelegate
        super.init()
        self.impactFeedbackGenerator.prepare()

        if isAppearing
        {
            self.presentingVC?.view.addGestureRecognizer(pan)
        }
        else
        {
            self.presentedVC?.view.addGestureRecognizer(pan)
        }

    }

    // Setup and moves shade view controller to just above screen if appearing
    func setupViewsForTransition(with transitionContext: UIViewControllerContextTransitioning)
    {
        // Get relevant views and view controllers from transitionContext
        guard let fromVC = transitionContext.viewController(forKey: .from),
            let toVC = transitionContext.viewController(forKey: .to),
            let toView = toVC.view else { return }

        let containerView = transitionContext.containerView

        // Hold refrence to transitionContext to notify it of completion
        self.transitionContext = transitionContext
        if isAppearing
```

```
        {
            // Position toView  just off-screen
            let fromViewInitialFrame = transitionContext.initialFrame(for: fromVC)
            var toViewInitialFrame = toView.frame
            toViewInitialFrame.origin.y -= toViewInitialFrame.height
            toViewInitialFrame.origin.x = fromViewInitialFrame.width * 0.5 -
toViewInitialFrame.width * 0.5
            toView.frame = toViewInitialFrame

            containerView.addSubview(toView)
        }
        else
        {
            fromVC.view.addGestureRecognizer(pan)
        }
    }


    // Handles the entire interaction from presenting/dismissing to completion
    func handlePan(sender: UIPanGestureRecognizer)
    {
        let location = sender.location(in: transitionContext?.containerView)
        let velocity = sender.velocity(in: transitionContext?.containerView)
        let fromVC = transitionContext?.viewController(forKey: .from)
        let toVC = transitionContext?.viewController(forKey: .to)

        let touchStartHeight: CGFloat = 90.0
        let touchLocationFromBottom: CGFloat = 20.0

        switch sender.state
        {
        case .began:
            let beginLocation = sender.location(in: sender.view)
            if isAppearing
            {
                guard beginLocation.y <= touchStartHeight,
                    let presentedVC = self.presentedVC else { break }
                presentedVC.modalPresentationStyle = .custom
                presentedVC.transitioningDelegate = transitionDelegate
                presentingVC?.present(presentedVC, animated: true)
            }
            else
            {
                guard beginLocation.y >= (sender.view?.frame.height ?? 0.0) - touchStartHeight else
{ break }
                presentedVC?.dismiss(animated: true)
            }
        case .changed:
            guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
            UIView.animate(withDuration: 0.2)
            {
                view.frame.origin.y = location.y - view.bounds.height + touchLocationFromBottom
            }

            transitionContext?.updateInteractiveTransition(view.frame.maxY / view.frame.height
            )
        case .ended, .cancelled:
            guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
            let isCancelled = isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) : (velocity.y
> 0.5 || view.center.y > 0.0)
            addAttachmentBehavior(with: view, isCancelled: isCancelled)
            addCollisionBehavior(with: view)
            addItemBehavior(with: view)
```

```
                animator.addBehavior(self)
                animator.delegate = self

                self.action =
                { [weak self] in
                    guard let strongSelf = self else { return }
                    if strongSelf.animator.elapsedTime > strongSelf.finishTime
                    {
                        strongSelf.animator.removeAllBehaviors()
                    }
                    else
                    {
                        strongSelf.transitionContext?.updateInteractiveTransition(view.frame.maxY /
view.frame.height
                        )
                    }
                }
        default:
            break
        }
    }


    // Add collision behavior that causes bounce when finished
    func addCollisionBehavior(with view: UIView)
    {
        let collisionBehavior = UICollisionBehavior(items: [view])
        let insets = UIEdgeInsets(top: -view.bounds.height, left: 0.0, bottom: 0.0, right: 0.0)
        collisionBehavior.setTranslatesReferenceBoundsIntoBoundary(with: insets)
        collisionBehavior.collisionDelegate = self
        self.addChildBehavior(collisionBehavior)
    }

    // Add attachment behavior that pulls shade either to top or bottom
    func addAttachmentBehavior(with view: UIView, isCancelled: Bool)
    {
        let anchor: CGPoint
        switch (isAppearing, isCancelled)
        {
        case (true, true), (false, false):
            anchor = CGPoint(x: view.center.x, y: -view.frame.height)
        case (true, false), (false, true):
            anchor = CGPoint(x: view.center.x, y: view.frame.height)
        }
        let attachmentBehavior = UIAttachmentBehavior(item: view, attachedToAnchor: anchor)
        attachmentBehavior.damping = 0.1
        attachmentBehavior.frequency = 3.0
        attachmentBehavior.length = 0.5 * view.frame.height
        self.addChildBehavior(attachmentBehavior)
    }

    // Makes view more bouncy
    func addItemBehavior(with view: UIView)
    {
        let itemBehavior = UIDynamicItemBehavior(items: [view])
        itemBehavior.allowsRotation = false
        itemBehavior.elasticity = 0.6
        self.addChildBehavior(itemBehavior)
    }

}
extension ShadeAnimator: UIDynamicAnimatorDelegate
```

```
{
    // Determines transition has ended
    func dynamicAnimatorDidPause(_ animator: UIDynamicAnimator)
    {
        guard let transitionContext = self.transitionContext else { return }
        let fromVC = transitionContext.viewController(forKey: .from)
        let toVC = transitionContext.viewController(forKey: .to)
        guard let view = isAppearing ? toVC?.view : fromVC?.view else { return }
        switch (view.center.y < 0.0, isAppearing)
        {
        case (true, true), (true, false):
            view.removeFromSuperview()
            transitionContext.finishInteractiveTransition()
            transitionContext.completeTransition(!isAppearing)
        case (false, true):
            toVC?.view.frame = transitionContext.finalFrame(for: toVC!)
            transitionContext.finishInteractiveTransition()
            transitionContext.completeTransition(true)
        case (false, false):
            fromVC?.view.frame = transitionContext.initialFrame(for: fromVC!)
            transitionContext.cancelInteractiveTransition()
            transitionContext.completeTransition(false)
        }
        childBehaviors.forEach { removeChildBehavior($0) }
        animator.removeAllBehaviors()
        self.animator = nil
        self.transitionContext = nil
    }
}
extension ShadeAnimator: UICollisionBehaviorDelegate
{
    // Triggers haptics
    func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item: UIDynamicItem,
withBoundaryIdentifier identifier: NSCopying?, at p: CGPoint)
    {
        guard p.y > 0.0 else { return }
        impactFeedbackGenerator.impactOccurred()
    }
}
extension ShadeAnimator: UIViewControllerInteractiveTransitioning
{
    // Starts transition
    func startInteractiveTransition(_ transitionContext: UIViewControllerContextTransitioning)
    {
        setupViewsForTransition(with: transitionContext)
    }
}
```

**Objective-C**

```objc
@interface ShadeAnimator() <UIDynamicAnimatorDelegate, UICollisionBehaviorDelegate>
@property (nonatomic, assign) BOOL isAppearing;
@property (nonatomic, weak) UIViewController *presentingVC;
@property (nonatomic, weak) UIViewController *presentedVC;
@property (nonatomic, weak) NSObject<UIViewControllerTransitioningDelegate> *transitionDelegate;
@property (nonatomic, strong) UIImpactFeedbackGenerator *impactFeedbackGenerator;
@property (nonatomic, strong) id<UIViewControllerContextTransitioning> transitionContext;
@property (nonatomic, assign) NSTimeInterval finishTime;
@property (nonatomic, strong) UIPanGestureRecognizer *pan;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ShadeAnimator
```

```objectivec
- (instancetype)initWithIsAppearing:(BOOL)isAppearing presentingVC:(UIViewController *)presentingVC
presentedVC:(UIViewController *)presentedVC
transitionDelegate:(id<UIViewControllerTransitioningDelegate>)transitionDelegate
{
    self = [super init];
    if (self)
    {
        _isAppearing = isAppearing;
        _presentingVC = presentingVC;
        _presentedVC = presentedVC;
        _transitionDelegate = transitionDelegate;
        _impactFeedbackGenerator = [[UIImpactFeedbackGenerator
alloc]initWithStyle:UIImpactFeedbackStyleLight];
        [_impactFeedbackGenerator prepare];
        if (_isAppearing)
        {
            [_presentingVC.view addGestureRecognizer:self.pan];
        }
        else
        {
            [_presentedVC.view addGestureRecognizer:self.pan];
        }
    }
    return self;
}

#pragma mark - Lazy Init
- (UIPanGestureRecognizer *)pan
{
    if (!_pan)
    {
        _pan = [[UIPanGestureRecognizer alloc]initWithTarget:self action:@selector(handlePan:)];
    }
    return _pan;
}

- (UIDynamicAnimator *)animator
{
    if (!_animator)
    {
        _animator = [[UIDynamicAnimator
alloc]initWithReferenceView:self.transitionContext.containerView];
    }
    return _animator;
}

#pragma mark - Setup
-
(void)setupViewForTransitionWithContext:(id<UIViewControllerContextTransitioning>)transitionContext
{
    UIViewController *fromVC = [transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *toView = toVC.view;
    UIView *containerView = transitionContext.containerView;
    self.transitionContext = transitionContext;
    if (self.isAppearing)
    {
        CGRect fromViewInitialFrame = [transitionContext initialFrameForViewController:fromVC];
        CGRect toViewInitialFrame = toView.frame;
        toViewInitialFrame.origin.y -= CGRectGetHeight(toViewInitialFrame);
```

```
        toViewInitialFrame.origin.x = CGRectGetWidth(fromViewInitialFrame) * 0.5 -
CGRectGetWidth(toViewInitialFrame) * 0.5;

        [containerView addSubview:toView];
    }
    else
    {
        [fromVC.view addGestureRecognizer:self.pan];
    }
}

#pragma mark - Gesture
- (void)handlePan:(UIPanGestureRecognizer *)sender
{
    CGPoint location = [sender locationInView:self.transitionContext.containerView];
    CGPoint velocity = [sender velocityInView:self.transitionContext.containerView];
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];

    CGFloat touchStartHeight = 90.0;
    CGFloat touchLocationFromBottom = 20.0;

    if (sender.state == UIGestureRecognizerStateBegan)
    {
        CGPoint beginLocation = [sender locationInView:sender.view];
        if (self.isAppearing)
        {
            if (beginLocation.y <= touchStartHeight)
            {
                self.presentedVC.modalPresentationStyle = UIModalPresentationCustom;
                self.presentedVC.transitioningDelegate = self.transitionDelegate;
                [self.presentingVC presentViewController:self.presentedVC animated:YES
completion:nil];
            }
        }
        else
        {
            if (beginLocation.y >= [sender locationInView:sender.view].y - touchStartHeight)
            {
                [self.presentedVC dismissViewControllerAnimated:true completion:nil];
            }
        }
    }
    else if (sender.state == UIGestureRecognizerStateChanged)
    {
        UIView *view = self.isAppearing ? toVC.view : fromVC.view;
        [UIView animateWithDuration:0.2 animations:^{
            CGRect frame = view.frame;
            frame.origin.y = location.y - CGRectGetHeight(view.bounds) + touchLocationFromBottom;
            view.frame = frame;
        }];
        [self.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
CGRectGetHeight(view.frame)];
    }
    else if (sender.state == UIGestureRecognizerStateEnded || sender.state ==
UIGestureRecognizerStateCancelled)
    {
        UIView *view = self.isAppearing ? toVC.view : fromVC.view;
        BOOL isCancelled = self.isAppearing ? (velocity.y < 0.5 || view.center.y < 0.0) :
(velocity.y > 0.5 || view.center.y > 0.0);
```

```objc
        [self addAttachmentBehaviorWithView:view isCancelled:isCancelled];
        [self addCollisionBehaviorWithView:view];
        [self addItemBehaviorWithView:view];

        [self.animator addBehavior:self];
        self.animator.delegate = self;

        __weak ShadeAnimator *weakSelf = self;
        self.action =
        ^{
            if (weakSelf.animator.elapsedTime > weakSelf.finishTime)
            {
                [weakSelf.animator removeAllBehaviors];
            }
            else
            {
                [weakSelf.transitionContext updateInteractiveTransition:CGRectGetMaxY(view.frame) /
CGRectGetHeight(view.frame)];
            }
        };
    }
}

#pragma mark - UIViewControllerInteractiveTransitioning
- (void)startInteractiveTransition:(id<UIViewControllerContextTransitioning>)transitionContext
{
    [self setupViewForTransitionWithContext:transitionContext];
}

#pragma mark - Behaviors
- (void)addCollisionBehaviorWithView:(UIView *)view
{
    UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc]initWithItems:@[view]];
    UIEdgeInsets insets = UIEdgeInsetsMake(-CGRectGetHeight(view.bounds), 0.0, 0.0, 0.0);
    [collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:insets];
    collisionBehavior.collisionDelegate = self;
    [self addChildBehavior:collisionBehavior];
}

- (void)addItemBehaviorWithView:(UIView *)view
{
    UIDynamicItemBehavior *itemBehavior = [[UIDynamicItemBehavior alloc]initWithItems:@[view]];
    itemBehavior.allowsRotation = NO;
    itemBehavior.elasticity = 0.6;
    [self addChildBehavior:itemBehavior];
}

- (void)addAttachmentBehaviorWithView:(UIView *)view isCancelled:(BOOL)isCancelled
{
    CGPoint anchor;
    if ((self.isAppearing && isCancelled) || (!self.isAppearing && isCancelled))
    {
        anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
    }
    else
    {
        anchor = CGPointMake(view.center.x, -CGRectGetHeight(view.frame));
    }
    UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior alloc]initWithItem:view
attachedToAnchor:anchor];
    attachmentBehavior.damping = 0.1;
    attachmentBehavior.frequency = 3.0;
```

```objc
    attachmentBehavior.length = 0.5 * CGRectGetHeight(view.frame);
    [self addChildBehavior:attachmentBehavior];
}

#pragma mark - UICollisionBehaviorDelegate
- (void)collisionBehavior:(UICollisionBehavior *)behavior
beganContactForItem:(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier
atPoint:(CGPoint)p
{
    if (p.y > 0.0)
    {
        [self.impactFeedbackGenerator impactOccurred];
    }
}

#pragma mark - UIDynamicAnimatorDelegate
- (void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    UIViewController *fromVC = [self.transitionContext
viewControllerForKey:UITransitionContextFromViewControllerKey];
    UIViewController *toVC = [self.transitionContext
viewControllerForKey:UITransitionContextToViewControllerKey];
    UIView *view = self.isAppearing ? toVC.view : fromVC.view;
    if (view.center.y < 0.0 && (self.isAppearing ||  !self.isAppearing))
    {
        [view removeFromSuperview];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:!self.isAppearing];
    }
    else if (view.center.y >= 0.0 && self.isAppearing)
    {
        toVC.view.frame = [self.transitionContext finalFrameForViewController:toVC];
        [self.transitionContext finishInteractiveTransition];
        [self.transitionContext completeTransition:YES];
    }
    else
    {
        fromVC.view.frame = [self.transitionContext initialFrameForViewController:fromVC];
        [self.transitionContext cancelInteractiveTransition];
        [self.transitionContext completeTransition:NO];
    }
    for (UIDynamicBehavior *behavior in self.childBehaviors)
    {
        [self removeChildBehavior:behavior];
    }
    [animator removeAllBehaviors];
    self.animator = nil;
    self.transitionContext = nil;
}

@end
```

The animator triggers the start of the transition when the pan gesture begins. And simply moves the view as the gesture changes. But when the gesture ends that is when `UIDynamicBehaviors` determines if the transition should be completed or cancelled. To do so it uses an attachment and collision behavior. For more information see the 2013 WWDC Session "Advanced Techniques with UIKit Dynamics.

# Section 59.5: Map Dynamic Animation Position Changes to

# Bounds

This example shows how to customize the `UIDynamicItem` protocol to map position changes of a view being dynamically animated to bounds changes to create a `UIButton` that expands and contracts in a elastic fashion.
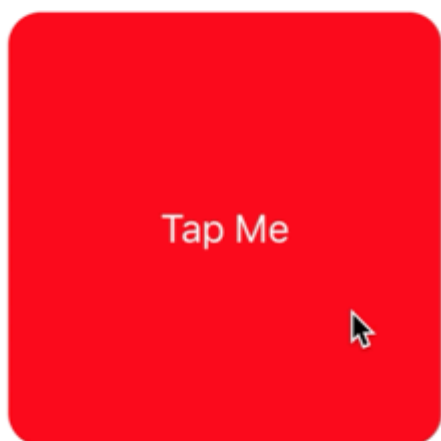
To start we need to create a new protocol that implements `UIDynamicItem` but that also has a settable and gettable bounds property.

**Swift**

```swift
protocol ResizableDynamicItem: UIDynamicItem
{
    var bounds: CGRect { set get }
}
extension UIView: ResizableDynamicItem {}
```

**Objective-C**

```
@protocol ResizableDynamicItem <UIDynamicItem>
@property (nonatomic, readwrite) CGRect bounds;
@end
```

We'll then create a wrapper object that will wrap a `UIDynamicItem` but will map center changes to the item's width and height. We will also provide passthroughs for `bounds` and `transform` of the underlying item. This will cause any changes the dynamic animator makes to the center x and y values of the underlying item will be applied to the items width and height.

**Swift**

```swift
final class PositionToBoundsMapping: NSObject, UIDynamicItem
{
    var target: ResizableDynamicItem

    init(target: ResizableDynamicItem)
    {
        self.target = target
        super.init()
    }

    var bounds: CGRect
    {
        get
        {
            return self.target.bounds
        }
    }

    var center: CGPoint
    {
        get
        {
            return CGPoint(x: self.target.bounds.width, y: self.target.bounds.height)
        }

        set
        {
            self.target.bounds = CGRect(x: 0.0, y: 0.0, width: newValue.x, height: newValue.y)
        }
    }

    var transform: CGAffineTransform
    {
        get
        {
            return self.target.transform
        }

        set
        {
            self.target.transform = newValue
        }
    }
}
```

**Objective-C**

```objc
@interface PositionToBoundsMapping ()
@property (nonatomic, strong) id<ResizableDynamicItem> target;
@end
```

```
@implementation PositionToBoundsMapping

- (instancetype)initWithTarget:(id<ResizableDynamicItem>)target
{
    self = [super init];
    if (self)
    {
        _target = target;
    }
    return self;
}

- (CGRect)bounds
{
    return self.target.bounds;
}

- (CGPoint)center
{
    return CGPointMake(self.target.bounds.size.width, self.target.bounds.size.height);
}

- (void)setCenter:(CGPoint)center
{
    self.target.bounds = CGRectMake(0, 0, center.x, center.y);
}

- (CGAffineTransform)transform
{
    return self.target.transform;
}

- (void)setTransform:(CGAffineTransform)transform
{
    self.target.transform = transform;
}

@end
```

Finally, we'll create a `UIViewController` that will have a button. When the button is pressed we will create `PositionToBoundsMapping` with the button as the wrapped dynamic item. We create a `UIAttachmentBehavior` to it's current position then add an instantaneous `UIPushBehavior` to it. However because we have mapped changes its bounds, the button does not move but rather grows and shrinks.

**Swift**
```
final class ViewController: UIViewController
{
    lazy var button: UIButton =
    {
        let button = UIButton(frame: CGRect(x: 0.0, y: 0.0, width: 300.0, height: 200.0))
        button.backgroundColor = .red
        button.layer.cornerRadius = 15.0
        button.setTitle("Tap Me", for: .normal)
        self.view.addSubview(button)
        return button
    }()

    var buttonBounds = CGRect.zero
    var animator: UIDynamicAnimator?

    override func viewDidLoad()
```

```
    {
        super.viewDidLoad()
        view.backgroundColor = .white
        button.addTarget(self, action: #selector(self.didPressButton(sender:)), for:
.touchUpInside)
        buttonBounds = button.bounds
    }

    override func viewDidLayoutSubviews()
    {
        super.viewDidLayoutSubviews()
        button.center = view.center
    }

    func didPressButton(sender: UIButton)
    {
        // Reset bounds so if button is press twice in a row, previous changes don't propogate
        button.bounds = buttonBounds
        let animator = UIDynamicAnimator(referenceView: view)

        // Create mapping
        let buttonBoundsDynamicItem = PositionToBoundsMapping(target: button)

        // Add Attachment behavior
        let attachmentBehavior = UIAttachmentBehavior(item: buttonBoundsDynamicItem,
attachedToAnchor: buttonBoundsDynamicItem.center)

        // Higher frequency faster oscillation
        attachmentBehavior.frequency = 2.0

        // Lower damping longer oscillation lasts
        attachmentBehavior.damping = 0.1
        animator.addBehavior(attachmentBehavior)

        let pushBehavior = UIPushBehavior(items: [buttonBoundsDynamicItem], mode: .instantaneous)

        // Change angle to determine how much height/ width should change 45° means heigh:width is
1:1
        pushBehavior.angle = .pi / 4.0

        // Larger magnitude means bigger change
        pushBehavior.magnitude = 30.0
        animator.addBehavior(pushBehavior)
        pushBehavior.active = true

        // Hold refrence so animator is not released
        self.animator = animator
    }
}
```

**Objective-C**

```objectivec
@interface ViewController ()
@property (nonatomic, strong) UIButton *button;
@property (nonatomic, assign) CGRect buttonBounds;
@property (nonatomic, strong) UIDynamicAnimator *animator;
@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
```

```
    self.view.backgroundColor = [UIColor whiteColor];
    [self.button addTarget:self action:@selector(didTapButton:)
forControlEvents:UIControlEventTouchUpInside];
    self.buttonBounds = self.button.bounds;
}

- (void)viewDidLayoutSubviews
{
    [super viewDidLayoutSubviews];
    self.button.center = self.view.center;
}

- (UIButton *)button
{
    if (!_button)
    {
        _button = [[UIButton alloc]initWithFrame:CGRectMake(0.0, 0.0, 200.0, 200.0)];
        _button.backgroundColor = [UIColor redColor];
        _button.layer.cornerRadius = 15.0;
        [_button setTitle:@"Tap Me" forState:UIControlStateNormal];
        [self.view addSubview:_button];
    }
    return _button;
}

- (void)didTapButton:(id)sender
{
    self.button.bounds = self.buttonBounds;
    UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
    PositionToBoundsMapping *buttonBoundsDynamicItem = [[PositionToBoundsMapping
alloc]initWithTarget:sender];
    UIAttachmentBehavior *attachmentBehavior = [[UIAttachmentBehavior
alloc]initWithItem:buttonBoundsDynamicItem attachedToAnchor:buttonBoundsDynamicItem.center];
    [attachmentBehavior setFrequency:2.0];
    [attachmentBehavior setDamping:0.3];
    [animator addBehavior:attachmentBehavior];

    UIPushBehavior *pushBehavior = [[UIPushBehavior alloc] initWithItems:@[buttonBoundsDynamicItem]
mode:UIPushBehaviorModeInstantaneous];
    pushBehavior.angle = M_PI_4;
    pushBehavior.magnitude = 2.0;
    [animator addBehavior:pushBehavior];

    [pushBehavior setActive:TRUE];

    self.animator = animator;
}

@end
```

For more information see UIKit Dynamics Catalog

## Section 59.6: The Falling Square

Lets draw a square in the middle of our view and make it fall to the bottom and stop at the bottom edge collising
with the screen bottom boundary.
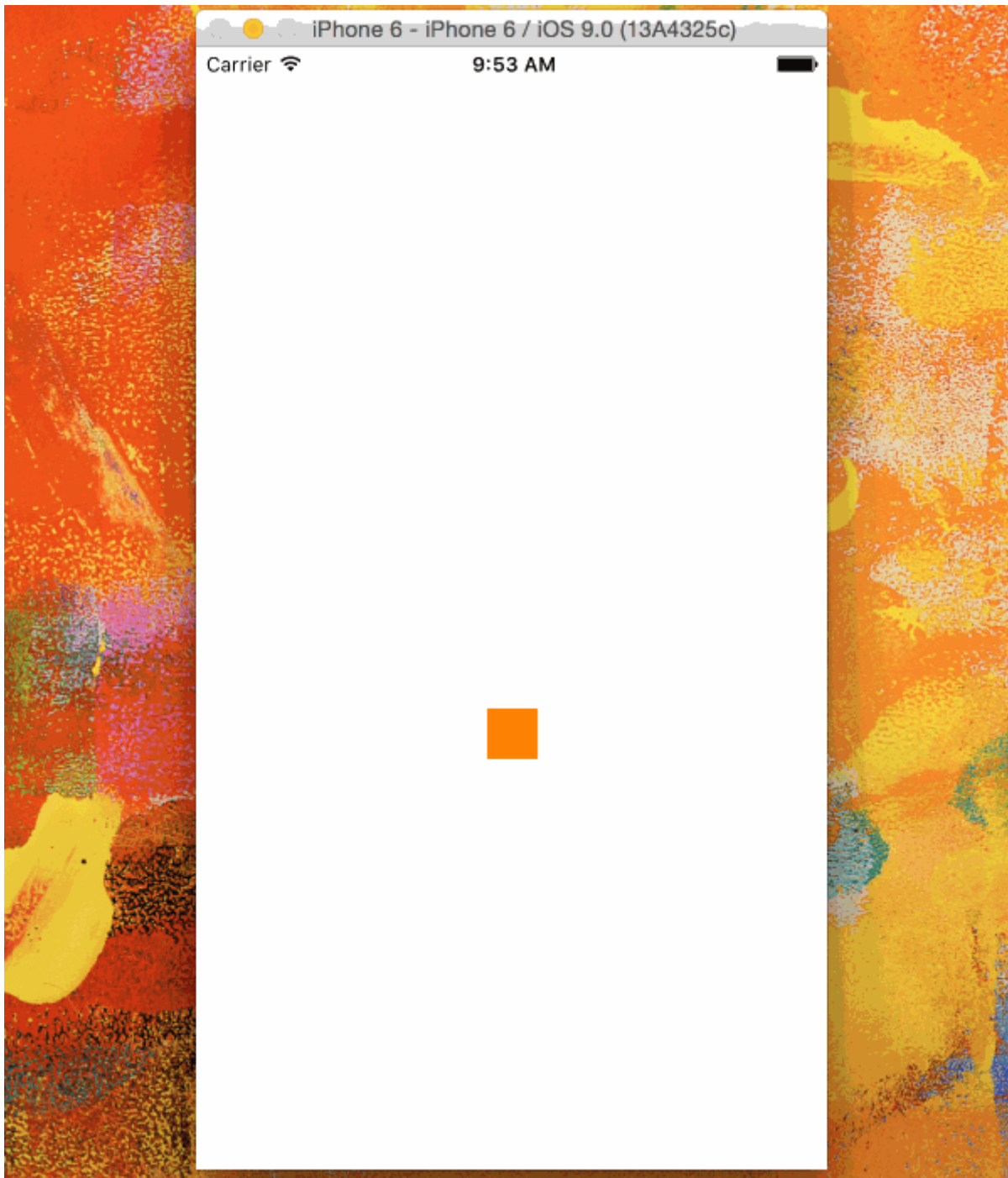
```
@IBOutlet var animationView: UIView!
var squareView:UIView!
var collision: UICollisionBehavior!
var animator: UIDynamicAnimator!
var gravity: UIGravityBehavior!

override func viewDidLoad() {
    super.viewDidLoad()
    let squareSize = CGSize(width: 30.0, height: 30.0)
    let centerPoint = CGPoint(x: self.animationView.bounds.midX - (squareSize.width/2), y:
self.animationView.bounds.midY - (squareSize.height/2))
    let frame = CGRect(origin: centerPoint, size: squareSize)
    squareView = UIView(frame: frame)
    squareView.backgroundColor = UIColor.orangeColor()
    animationView.addSubview(squareView)
    animator = UIDynamicAnimator(referenceView: view)
    gravity = UIGravityBehavior(items: [squareView])
    animator.addBehavior(gravity)
```

```
    collision = UICollisionBehavior(items: [square])
    collision.translatesReferenceBoundsIntoBoundary = true
    animator.addBehavior(collision)
}
```

# Chapter 60: UI Testing

## Section 60.1: Accessibility Identifier

**When Accessibility enabled in Utilities**

- Select `storyboard`.
- Expand `the Utilities`
- Select `Identity Inspector`
- Select your element on storyboard
- Add new Accessibility Identifier (in example `addButton`)
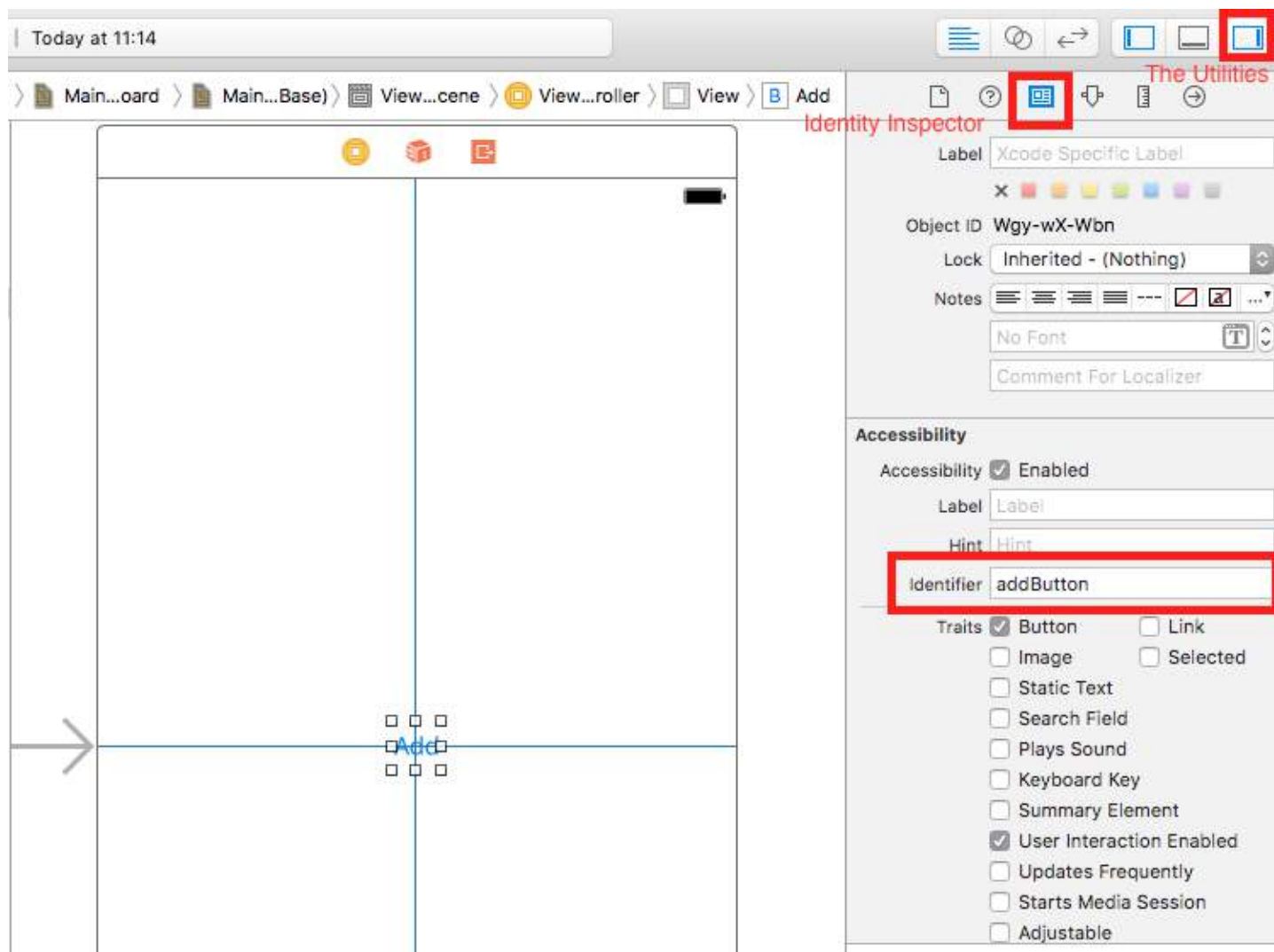


**When Accessibility disabled in Utilities**

- Select `storyboard`.
- Expand `the Utilities`
- Select `Identity Inspector`
- Select your element on storyboard
- Add attribute in `User Defined Runtime Attributes`
- For `Key Path` type - `accessibilityIdentifier`
- For `Type` - `String
- For `Value` - new accessibility identifier for your element (in example `view`)
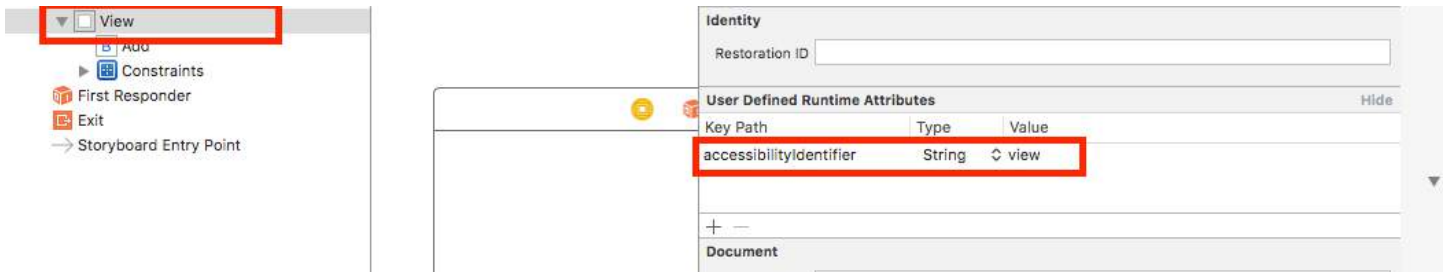
**Setting up in UITest file**

```
import XCTest

class StackOverFlowUITests: XCTestCase {

    private let app = XCUIApplication()

    //Views

    private var view: XCUIElement!

    //Buttons

    private var addButton: XCUIElement!


    override func setUp() {
        super.setUp()

        app.launch()

        //Views

        view = app.otherElements["view"]

        //Buttons

        addButton = app.buttons["addButton"]
    }

    func testMyApp() {

        addButton.tap()
        view.tap()
    }
}
```

In [  ] add Accessibility Identifier for element.

**UIView, UIImageView, UIScrollView**

```
let imageView = app.images["imageView"]
let scrollView = app.scrollViews["scrollView"]
let view = app.otherElements["view"]
```

**UILabel**

```
let label = app.staticTexts["label"]
```

**UIStackView**

```
let stackView = app.otherElements["stackView"]
```

**UITableView**

```
let tableView = app.tables["tableView"]
```

**UITableViewCell**

```swift
let tableViewCell = tableView.cells["tableViewCell"]
```

**UITableViewCell elements**

```swift
let tableViewCellButton = tableView.cells.element(boundBy: 0).buttons["button"]
```

**UICollectionView**

```swift
let collectionView = app.collectionViews["collectionView"]
```

**UIButton, UIBarButtonItem**

```swift
let button = app.buttons["button"]
let barButtonItem = app.buttons["barButtonItem"]
```

**UITextField**

- normal UITextField

```swift
let textField = app.textFields["textField"]
```

- password UITextField

```swift
let passwordTextField = app.secureTextFields["passwordTextField"]
```

**UITextView**

```swift
let textView = app.textViews["textView"]
```

**UISwitch**

```swift
let switch = app.switches["switch"]
```

**Alerts**

```swift
let alert = app.alerts["About yourself"] // Title of presented alert
```

# Section 60.2: Adding Test Files to Xcode Project

**When creating the project**

You should check "Include UI Tests" in the project creation dialog.
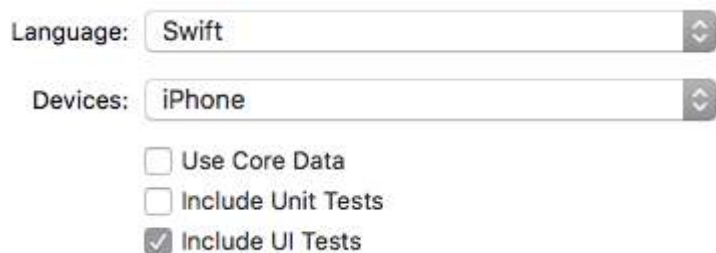


**After creating the project**

If you missed checking `UI target` while creating project, you could always add test target later.

Setps:

- While project open go to `File -> New -> Target`
- Find `iOS UI Testing Bundle`

# Section 60.3: Disable animations during UI Testing

In a test you can disable animations by adding in `setUp`:

```
app.launchEnvironment = ["animations": "0"]
```

Where `app` is instance of XCUIApplication.

# Section 60.4: Lunch and Terminate application while executing

**Lunch application for testing**

```
override func setUp() {
    super.setUp()

    let app = XCUIApplication()

    app.launch()
}
```

**Terminating application**

```
func testStacOverFlowApp() {

    app.terminate()
}
```

# Section 60.5: Rotate devices

Device can be rotate by changing `orientation` in `XCUIDevice.shared().orientation`:

```
XCUIDevice.shared().orientation = .landscapeLeft
XCUIDevice.shared().orientation = .portrait
```

# Chapter 61: Change Status Bar Color

## Section 61.1: For non-UINavigationBar status bars

1. In info.plist set `View controller-based status bar appearance` to YES
2. In view controllers not contained by `UINavigationController` implement this method.

**In Objective-C:**

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

**In Swift:**

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return UIStatusBarStyle.LightContent
}
```

## Section 61.2: For UINavigationBar status bars

Subclass UINavigationController and then override these methods:

In Objective-C:

```
- (UIStatusBarStyle)preferredStatusBarStyle
{
    return UIStatusBarStyleLightContent;
}
```

In Swift:

```
override func preferredStatusBarStyle() -> UIStatusBarStyle {
    return .lightContent
}
```

Alternatively, you can set `barStyle` on the `UINavigationBar` instance:

Objective C:

```
// e.g. in your view controller's viewDidLoad method:
self.navigationController.navigationBar.barStyle = UIBarStyleBlack;  // this will give you a white
status bar
```

Swift

```
// e.g. in your view controller's viewDidLoad method:
navigationController?.navigationBar.barStyle = .black // this will give you a white status bar
```

`UIBarStyle` options are `default`, `black`, `blackOpaque`, `blackTranslucent`. The latter 3 should all give you a status bar with white text, just the last two specify the opacity of the bar.

Note: you can still change the appearance of your navigation bar as you like.

# Section 61.3: For ViewController containment

If you are using `UIViewControllerContainment` there are a few other methods that are worth looking at.

When you want a child viewController to control the presentation of the status bar (i.e. if the child is positioned at the top of the screen

in Swift

```swift
class RootViewController: UIViewController {

    private let messageBarViewController = MessageBarViewController()

    override func childViewControllerForStatusBarStyle() -> UIViewController? {
        return messageBarViewController
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        //add child vc code here...

        setNeedsStatusBarAppearanceUpdate()
    }
}

class MessageBarViewController: UIViewController {

    override func preferredStatusBarStyle() -> UIStatusBarStyle {
        return .Default
    }
}
```

# Section 61.4: If you cannot change ViewController's code

If you are using library that contains (for example) AwesomeViewController with a wrong status bar color you can try this:

```swift
let awesomeViewController = AwesomeViewController()
awesomeViewController.navigationBar.barStyle = .blackTranslucent // or other style
```

# Section 61.5: Changing the status bar style for the entire application

**SWIFT:**

**Step 1:**

In your **Info.plist** add the following attribute:

```
View controller-based status bar appearance
```

and set its value to

```
NO
```

as described in the image below:

| Key | | Type | Value |
|---|---|---|---|
| ▼ Information Property List | | Dictionary | (15 items) |
| View controller-based status... ⬍ ⊕ ⊖ | | Boolean ⬍ | NO |

**Step 2:**

In your **AppDelegate.swift** file, in `didFinishLaunchingWithOptions` method, add this code:

```
UIApplication.shared.statusBarStyle = .lightContent
```

**or**

```
UIApplication.shared.statusBarStyle = .default
```

- The **.lightContent** option will set the colour of the **statusBar** to white, for the entire app.

- The **.default** option will set the colour of the **statusBar** to the original black colour, for the entire app.

**OBJECTIVE-C:**

Follow the first step from the **SWIFT** Section. Then add this code to the **AppDelegate.m** file:

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleLightContent];
```

**or**

```
[[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleDefault];
```

# Chapter 62: UISegmentedControl

A UISegmentedControl object is a horizontal control made of multiple segments, each segment functioning as a discrete button. A segmented control affords a compact means to group together a number of controls.

## Section 62.1: Creating UISegmentedControl via code

1. Create new instance of UISegmentedControl filled with 3 items (segments):

```
let mySegmentedControl = UISegmentedControl (items: ["One", "Two", "Three"])
```

2. Setup frame;

```
mySegmentedControl.frame = CGRect(x: 0.0, y: 0.0, width: 300, height: 50)
```

3. Make default selection (not that segments are indexed by 0):

```
mySegmentedControl.selectedSegmentIndex = 0
```

4. Configure target:

```
mySegmentedControl.addTarget(self, action: #selector(segmentedValueChanged(_:)), for:
.valueChanged)
```

5 Handle value changed:

```
func segmentedValueChanged(_ sender:UISegmentedControl!) {
    print("Selected Segment Index is : \(sender.selectedSegmentIndex)")
}
```

6. Add UISegmentedControl to views hierarchy

```
yourView.addSubview(mySegmentedControl)
```

# Chapter 63: Passing Data between View Controllers

## Section 63.1: Using the Delegate Pattern (passing data back)

To pass data from the current view controller back to the previous view controller, you can use the delegate pattern.
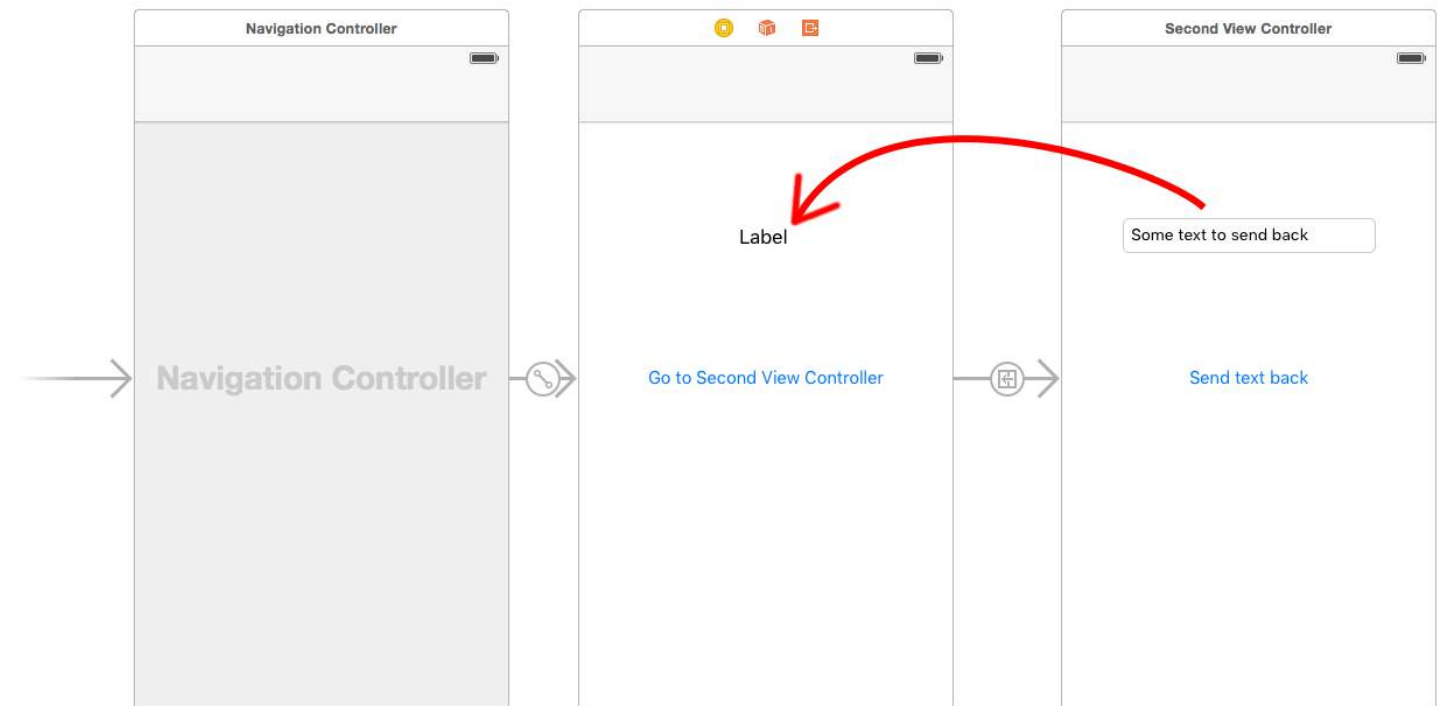


This example assumes that you have made a segue in the Interface Builder and that you set the segue identifier to showSecondViewController. The outlets and actions must also be hooked up to the names in the following code.

**First View Controller**

The code for the First View Controller is

**Swift**

```swift
class FirstViewController: UIViewController, DataEnteredDelegate {

    @IBOutlet weak var label: UILabel!

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "showSecondViewController", let secondViewController =
segue.destinationViewController as? SecondViewController  {
            secondViewController.delegate = self
        }
    }

    // required method of our custom DataEnteredDelegate protocol
    func userDidEnterInformation(info: String) {
        label.text = info
        navigationController?.popViewControllerAnimated(true)
    }
}
```

**Objective-C**

```
@interface FirstViewController : UIViewController <DataEnteredDelegate>
@property (weak, nonatomic) IBOutlet UILabel *label;
@end

@implementation FirstViewController
- (void)viewDidLoad {
    [super viewDidLoad];
}
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    SecondViewController *secondViewController = segue.destinationViewController;
    secondViewController.delegate = self;

}
-(void)userDidEnterInformation:(NSString *)info {
    _label.text = info
    [self.navigationController popViewControllerAnimated:YES];
}
@end
```

Note the use of our custom `DataEnteredDelegate` protocol.

**Second View Controller and Protocol**

The code for the second view controller is

**Swift**
```
// protocol used for sending data back
protocol DataEnteredDelegate: class {
    func userDidEnterInformation(info: String)
}

class SecondViewController: UIViewController {

    // making this a weak variable so that it won't create a strong reference cycle
    weak var delegate: DataEnteredDelegate?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {

        // call this method on whichever class implements our delegate protocol (the first view
controller)
        delegate?.userDidEnterInformation(textField.text ?? "")
    }
}
```

**Objective-C**
```
@protocol DataEnteredDelegate <NSObject>
-(void)userDidEnterInformation:(NSString *)info;
@end

@interface SecondViewController : UIViewController
@property (nonatomic) id <DataEnteredDelegate> delegate;
@property (weak, nonatomic) IBOutlet UITextField *textField;
@end

@implementation SecondViewController
- (void)viewDidLoad {
    [super viewDidLoad];
}
```

```
- (IBAction) sendTextBackButton:(id)sender{
    [_delegate userDidEnterInformation:textField.text];
}
@end
```

Note that the `protocol` is outside of the View Controller class.

# Section 63.2: Using Segues (passing data forward)

To pass data from the current view controller to the next new view controller (not a previous view controller) using segues, first create a segue with an identifier in the relevant storyboard. Override your current view controller's `prepareForSegue` method. Inside the method check for the segue you just created by its identifier. Cast the destination view controller and pass data to it by setting properties on the downcast view controller.

Setting an identifier for a segue:
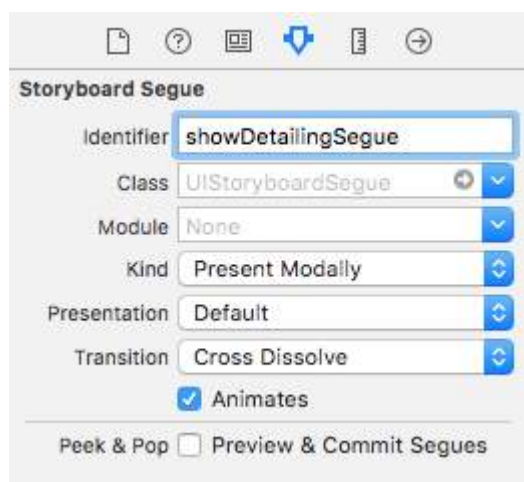


Segues can be performed programatically or using button action event set in the storyboard by ctrl+drag to destination view controller. You can call for a segue programatically, when needed, using segue identifier in the view controller:

**Objective-C**

```
- (void)showDetail {
    [self performSegueWithIdentifier:@"showDetailingSegue" sender:self];
}
```

**Swift**

```
func showDetail() {
    self.performSegue(withIdentifier: "showDetailingSegue", sender: self)
}
```

You can configure segue payload in the overridden version of `prepareForSegue` method. You can set required properties before destination view controller is loaded.

**Objective-C**

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if([segue.identifier isEqualToString:@"showDetailingSegue"]){
        DetailViewController *controller = (DetailViewController *)segue.destinationViewController;
        controller.isDetailingEnabled = YES;
    }
```

```
    }
```

**Swift**

```swift
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetailingSegue" {
        let controller = segue.destinationViewController as! DetailViewController
        controller.isDetailingEnabled = true
    }
}
```

`DetailViewController` is the name of the second view controller and `isDetailingEnabled` is a public variable in that view controller.

To expand on this pattern, you can treat a public method on `DetailViewController` as a pseudo initializer, to help initialize any required variables. This will self document variables that need to be set on `DetailViewController` without having to read through it's source code. It's also a handy place to put defaults.

**Objective-C**

```objc
- (void)initVC:(BOOL *)isDetailingEnabled {
    self.isDetailingEnabled = isDetailingEnabled
}
```

**Swift**

```swift
func initVC(isDetailingEnabled: Bool) {
    self.isDetailingEnabled = isDetailingEnabled
}
```

# Section 63.3: Passing data backwards using unwind to segue

In contrast to segue that lets you pass data "forward" from current view controller to destination view controller:

*(VC1) -> (VC2)*

Using "unwind" you can do the opposite, pass data from the destination or current view controller to its presenting view controller:

*(VC1) <- (VC2)*

**NOTE**: Pay attention that using unwind lets you pass the data first and afterwards the current view controller (VC2) will get deallocated.

Here's how to do it:

First, you will need to add the following declaration at the presenting view controller (VC1) which is the view controller that we want to pass the data to:

```
@IBAction func unwindToPresentingViewController(segue:UIStoryboardSegue)
```

The important thing is to use the prefix `unwind`, this "informs" Xcode that this is an unwind method giving you the option to use it in storyboard as well.

Afterwards you will need to implement the method, it looks almost the same as an actual segue:

```
@IBAction func unwindToPresentingViewController(segue:UIStoryboardSegue)
{
    if segue.identifier == "YourCustomIdentifer"
    {
        if let VC2 = segue.sourceViewController as? VC2
        {
            //    Your custom code in here to access VC2 class member
        }
    }
```

Now you have 2 options to invoke the unwind calls:

1. You can "hard code" invoke the: `self.performSegueWithIdentifier("YourCustomIdentifier", sender: self)` which will do the unwind for you whenever you will `performSegueWithIdentifier`.
2. You can link the unwind method using the `storyboard` to your "Exit" object: ctrl + drag the button you want to invoke the unwind method, to the "Exit" object:



Release and you will have the option to choose your custom unwind method:



# Section 63.4: Passing data using closures (passing data back)

Instead of using the **delegate pattern**, that split the implementation in various part of the `UIViewController` class, you can even use `closures` to pass data back and forward. By assuming that you're using the `UIStoryboardSegue`, in the `prepareForSegue` method you can easily setup the new controller in one step

```
final class DestinationViewController: UIViewController {
    var onCompletion: ((success: Bool) -> ())?

    @IBAction func someButtonTapped(sender: AnyObject?) {
        onCompletion?(success: true)
    }
}

final class MyViewController: UIViewController {
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

        guard let destinationController = segue.destinationViewController as?
DestinationViewController else { return }

        destinationController.onCompletion = { success in
            // this will be executed when `someButtonTapped(_:)` will be called
```

```
                print(success)
        }
    }
}
```

This is an example of use and it's better to use on Swift, Objective-C block's syntax is not so easy to make the code more readable

# Section 63.5: Using callback closure(block) passing data back

this topic is a classical issue in iOS development, and its solution is various as other example already shown. In this example I'll show another daily common use one: passing data using `closure` by adapting `delegate pattern` example on this page into callback `closure`!

one thing this method is superior to `delegate pattern` is instead of split the setting up code in two different place( look at delegate example on this page, `prepareForSegue` ,`userDidEnterInformation` ) rather gathering them together( only in `prepareForSegue`, I'll show it )

**Start from Second View Controller**

we must figure out how to use callback, then can we write it, this is why we start from second view controller since it's where we use callback: when we got the new text input, we call our callback, **using callback's parameter** as a medium to passing data back to first ViewController, notice that I said using callback's parameter, this is very important, novices(as I was) always overlook this and don't know where to start to write callback closure properly

so in this case, we know that our callback only take one parameter: text and its type is `String`, let's declare it and make it property since we need populate from our first view controller

I just comment all the `delegate` part and keep it for comparing

```swift
class SecondViewController: UIViewController {

    //weak var delegate: DataEnteredDelegate? = nil
    var callback: ((String?)->())?

    @IBOutlet weak var textField: UITextField!

    @IBAction func sendTextBackButton(sender: AnyObject) {

        //delegate?.userDidEnterInformation(textField.text!)
        callback?(input.text)

        self.navigationController?.popViewControllerAnimated(true)
    }
}
```

**Finish first view controller**

all you have to do is passing callback closure, and we are done, closure will do the future work for us since we already set it up in second view controller

look how it make our code shorter compared to the `delegate pattern`

```swift
//no more DataEnteredDelegate
class FirstViewController: UIViewController {

    @IBOutlet weak var label: UILabel!
```

```swift
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if segue.identifier == "showSecondViewController" {
            let secondViewController = segue.destinationViewController as! SecondViewController
            //secondViewController.delegate = self
            secondViewController.callback = { text in self.label.text = text }
        }
    }

    // required method of our custom DataEnteredDelegate protocol
    //func userDidEnterInformation(info: String) {
    //    label.text = info
    //}
}
```

and in the last, maybe someone of you will confused by the looking that we only passing the data(closure in this case) only in one way, from first view controller to second, no directly coming back from second view controller, how can we consider it as a communicating tool? maybe you really should run it and prove it yourself, all I will say it's **parameter**, it's **callback closure's parameter** that passing data back!

# Section 63.6: By assigning property (Passing data forward)

You can pass data directly by assigning the property of the next view controller before you push or present it.

```swift
class FirstViewController: UIViewController {

    func openSecondViewController() {

        // Here we initialize SecondViewController and set the id property to 492
        let secondViewController = SecondViewController()
        secondViewController.id = 492

        // Once it was assign we now push or present the view controller
        present(secondViewController, animated: true, completion: nil)
    }

}

class SecondViewController: UIViewController {

    var id: Int?

    override func viewDidLoad() {
        super.viewDidLoad()

        // Here we unwrapped the id and will get the data from the previous view controller.
        if let id = id {
            print("Id was set: \(id)")
        }
    }
}
```

# Chapter 64: Managing the Keyboard

## Section 64.1: Create a custom in-app keyboard
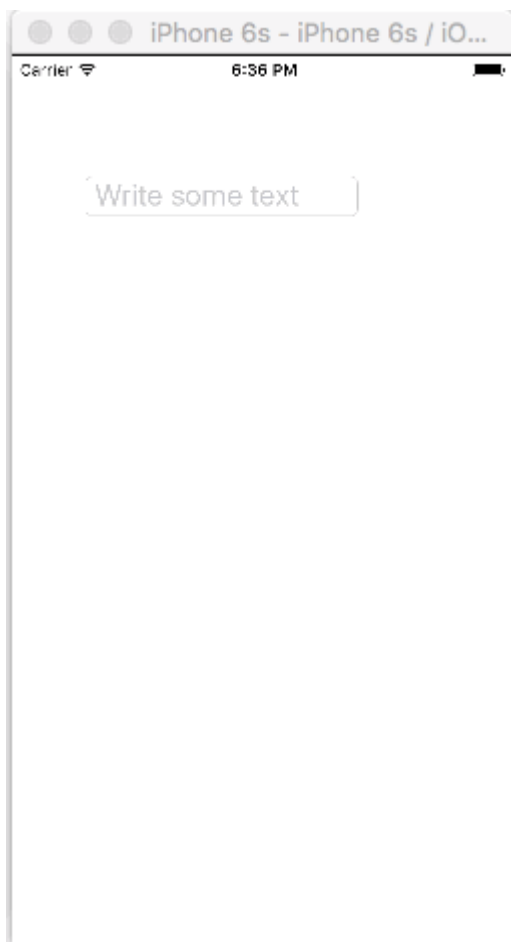


This is a basic in-app keyboard. The same method could be used to make just about any keyboard layout. Here are the main things that need to be done:

- Create the keyboard layout in an .xib file, whose owner is a Swift or Objective-C class that is a `UIView` subclass.
- Tell the `UITextField` to use the custom keyboard.
- Use a delegate to communicate between the keyboard and the main view controller.

**Create the .xib keyboard layout file**

- In Xcode go to **File > New > File… > iOS > User Interface > View** to create the .xib file.
- I called mine Keyboard.xib
- Add the buttons that you need.
- Use auto layout constraints so that no matter what size the keyboard is, the buttons will resize accordingly.
- Set the File's Owner (not the root view) to be the `Keyboard` class. This is a common source of error. You'll create this class in the next step. See the note at the end.

**Create the .swift UIView subclass keyboard file**

- In Xcode go to **File > New > File… > iOS > Source > Cocoa Touch Class** to create the Swift or Objective-C class. Choose `UIView` as a superclass for newly created class

- I called mine `Keyboard.swift` (Keyboard class in Objective-C)

- Add the following code for Swift:

```swift
import UIKit

// The view controller will adopt this protocol (delegate)
// and thus must contain the keyWasTapped method
protocol KeyboardDelegate: class {
    func keyWasTapped(character: String)
}

class Keyboard: UIView {

    // This variable will be set as the view controller so that
    // the keyboard can send messages to the view controller.
    weak var delegate: KeyboardDelegate?

    // MARK: keyboard initialization

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        initializeSubviews()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        initializeSubviews()
    }

    func initializeSubviews() {
        let xibFileName = "Keyboard" // xib extention not included
        let view = NSBundle.mainBundle().loadNibNamed(xibFileName, owner: self, options:
nil)[0] as! UIView
        self.addSubview(view)
        view.frame = self.bounds
    }

    // MARK: Button actions from .xib file

    @IBAction func keyTapped(sender: UIButton) {
        // When a button is tapped, send that information to the
        // delegate (ie, the view controller)
        self.delegate?.keyWasTapped(sender.titleLabel!.text!) // could alternatively send a
tag value
    }

}
```

- Add the following code for Objective-C:

**Keyboard.h File**

```objc
#import <UIKit/UIKit.h>

// The view controller will adopt this protocol (delegate)
// and thus must contain the keyWasTapped method
@protocol KeyboardDelegate<NSObject>
- (void)keyWasTapped:(NSString *)character;
@end

@interface Keyboard : UIView
```

```objc
@property (nonatomic, weak) id<KeyboardDelegate> delegate;
@end
```

**Keyboard.m File**

```objc
#import "Keyboard.h"

@implementation Keyboard

- (id)initWithCoder:(NSCoder *)aDecoder {
    self = [super initWithCoder:aDecoder];
    [self initializeSubviews];
    return self;
}

- (id)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    [self initializeSubviews];
    return self;
}

- (void)initializeSubviews {
    NSString *xibFileName = @"Keyboard"; // xib extention not included
    UIView *view = [[[NSBundle mainBundle] loadNibNamed:xibFileName owner:self options:nil]
firstObject];
    [self addSubview:view];
    view.frame = self.bounds;
}

// MARK: Button actions from .xib file

-(IBAction)keyTapped:(UIButton *)sender {
    // When a button is tapped, send that information to the
    // delegate (ie, the view controller)
    [self.delegate keyWasTapped:sender.titleLabel.text]; // could alternatively send a tag
value
}

@end
```

- Control drag actions from the buttons to button callback in the .xib file to the `@IBAction` method in the Swift or Objective-C owner to hook them all up.
- Note that the protocol and delegate code. See [this answer](#) for a simple explanation about how delegates work.

**Set up the View Controller**

- Add a `UITextField` to your main storyboard and connect it to your view controller with an `IBOutlet`. Call it `textField`.

- Use the following code for the View Controller in Swift:

```swift
import UIKit

class ViewController: UIViewController, KeyboardDelegate {

    @IBOutlet weak var textField: UITextField!

    override func viewDidLoad() {
```

```
        super.viewDidLoad()

        // initialize custom keyboard
        let keyboardView = Keyboard(frame: CGRect(x: 0, y: 0, width: 0, height: 300))
        keyboardView.delegate = self // the view controller will be notified by the
keyboard whenever a key is tapped

        // replace system keyboard with custom keyboard
        textField.inputView = keyboardView
    }

    // required method for keyboard delegate protocol
    func keyWasTapped(character: String) {
        textField.insertText(character)
    }
}
```

- Use the following code for Objective-C:

**.h File**

```
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@end
```

**.m File**

```
#import "ViewController.h"
#import "Keyboard.h"

@interface ViewController ()<KeyboardDelegate>

@property (nonatomic, weak) IBOutlet UITextField *textField;

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.

    // initialize custom keyboard
    Keyboard *keyboardView = [[Keyboard alloc] initWithFrame:CGRectMake(0, 0, 0, 300)];
    keyboardView.delegate = self; // the view controller will be notified by the keyboard
whenever a key is tapped

    // replace system keyboard with custom keyboard
    self.textField.inputView = keyboardView;
}

- (void)keyWasTapped:(NSString *)character {
    [self.textField insertText:character];
}

@end
```

- Note that the view controller adopts the `KeyboardDelegate` protocol that we defined above.

**Common error**

If you are getting an `EXC_BAD_ACCESS` error, it is probably because you set the view's custom class as `Keyboard` rather than do this for the nib File's Owner.

Select `Keyboard.nib` and then choose File's Owner.
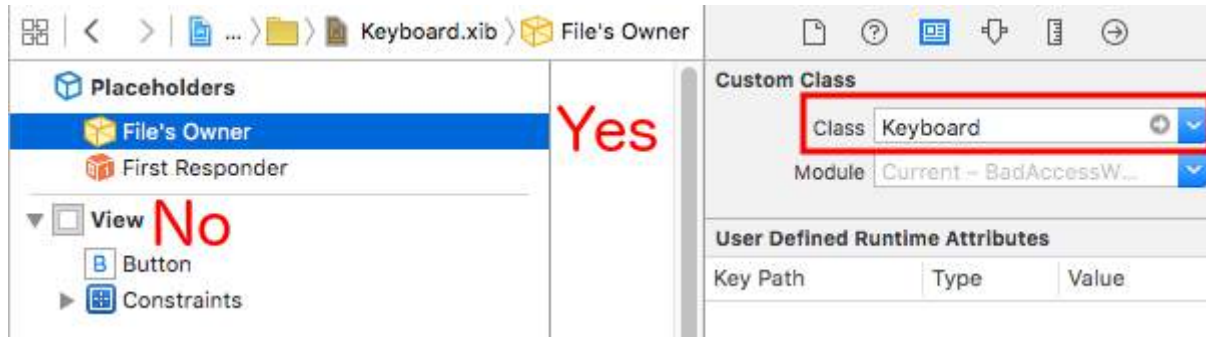


Make sure that the custom class for the root view is blank.
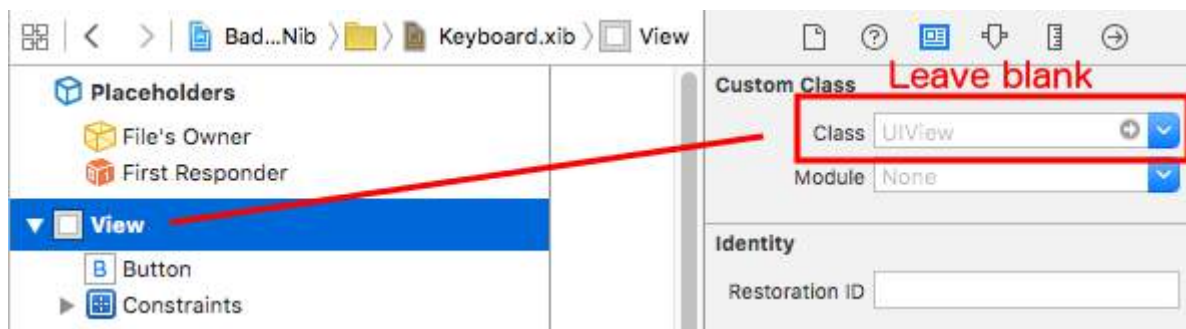


**Notes**

This example comes originally from this Stack Overflow answer.

# Section 64.2: Dismiss a keyboard with tap on view

If you want to hide a keyboard by tap outside of it, it's possible to use this hacky trick (works only with Objective-C):

```objc
- (void)viewDidLoad {
    [super viewDidLoad];

    // dismiss keyboard when tap outside a text field
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer alloc]
initWithTarget:self.view action:@selector(endEditing:)];
    [tapGestureRecognizer setCancelsTouchesInView:NO];
    [self.view addGestureRecognizer:tapGestureRecognizer];
}
```

for Swift there will be a bit more code:

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    // dismiss keyboard when tap outside a text field
    let tapGestureRecognizer: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(YourVCName.dismissKeyboard))
    view.addGestureRecognizer(tapGestureRecognizer)
```

```
}

//Calls this function when the tap is recognized.
func dismissKeyboard() {
    //Causes the view (or one of its embedded text fields) to resign the first responder status.
    view.endEditing(true)
}
```

Another Swift 3/iOS 10 example

```
class vc: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        txtSomeField.delegate = self
    }
}

extension vc: UITextFieldDelegate {
    //Hide the keyboard for any text field when the UI is touched outside of the keyboard.
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)
    {
        self.view.endEditing(true) //Hide the keyboard
    }
}
```

# Section 64.3: Managing the Keyboard Using a Singleton + Delegate

When I first started managing the keyboard I would use separate Notifications in each ViewController.

Notification Method (Using NSNotification):

```
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(ViewController.keyboardNotification(_:)), name: UIKeyboardWillChangeFrameNotification,
object: nil)
    }

    func keyboardNotification(notification: NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: NSTimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = endFrame?.size.height ?? 0.0
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options:
```

```
animationCurve, completion: nil)
    }
}
```

My problem was that I found myself writing this code again and again for every single ViewController. After experimenting a bit I found using a Singleton + Delegate pattern allowed me to reuse a bunch of code and organize all of the Keyboard Management in a single place!

Singleton + Delegate Method:

```
protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve:
UIViewAnimationOptions)
}

class KeyboardManager {

    weak var delegate: KeyboardManagerDelegate?

    class var sharedInstance: KeyboardManager {
        struct Singleton {
            static let instance = KeyboardManager()
        }
        return Singleton.instance
    }

    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name:
UIKeyboardWillChangeFrameNotification, object: nil)
    }

    @objc func keyboardWillChangeFrameNotification(notification: NSNotification) {
        guard let userInfo = notification.userInfo else { return }

        let endFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as? NSValue)?.CGRectValue()
        let duration: NSTimeInterval = (userInfo[UIKeyboardAnimationDurationUserInfoKey] as?
NSNumber)?.doubleValue ?? 0
        let animationCurveRawNSN = userInfo[UIKeyboardAnimationCurveUserInfoKey] as? NSNumber
        let animationCurveRaw = animationCurveRawNSN?.unsignedLongValue ??
UIViewAnimationOptions.CurveEaseOut.rawValue
        let animationCurve: UIViewAnimationOptions = UIViewAnimationOptions(rawValue:
animationCurveRaw)

        delegate?.keyboardWillChangeFrame(endFrame, duration: duration, animationCurve:
animationCurve)
    }
}
```

Now when I want to manage the keyboard from a ViewController all I need to do is set the delegate to that ViewController and implement any delegate methods.

```
class ViewController: UIViewController {
    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)
        KeyboardManager.sharedInstance.delegate = self
    }
}

// MARK: - Keyboard Manager
```

```
extension ViewController: KeyboardManagerDelegate {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve:
UIViewAnimationOptions) {
        if endFrame?.origin.y >= UIScreen.mainScreen().bounds.size.height {
            lowerViewBottomConstraint.constant = 0
        } else {
            lowerViewBottomConstraint.constant = (endFrame?.size.height ?? 0.0)
        }
        view.animateConstraintWithDuration(duration, delay: NSTimeInterval(0), options:
animationCurve, completion: nil)
    }
}
```

This method is very customizable too! Say we want to add functionality for `UIKeyboardWillHideNotification`. This is as easy as adding a method to our `KeyboardManagerDelegate`.

`KeyboardManagerDelegate` with `UIKeyboardWillHideNotification`:

```
protocol KeyboardManagerDelegate: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve:
UIViewAnimationOptions)
    func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])
}

class KeyboardManager {
    init() {
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(KeyboardManager.keyboardWillChangeFrameNotification(_:)), name:
UIKeyboardWillChangeFrameNotification, object: nil)
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(KeyboardManager.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
    }

    func keyboardWillHide(notification: NSNotification) {
        guard let userInfo = notification.userInfo else { return }
        delegate?.keyboardWillHide(userInfo)
    }
}
```

Say we only want to implement `func keyboardWillHide(notificationUserInfo: [NSObject: AnyObject])` in one ViewController. We can also make this method optional.

```
typealias KeyboardManagerDelegate = protocol<KeyboardManagerModel, KeyboardManagerConfigureable>

protocol KeyboardManagerModel: class {
    func keyboardWillChangeFrame(endFrame: CGRect?, duration: NSTimeInterval, animationCurve:
UIViewAnimationOptions)
}

@objc protocol KeyboardManagerConfigureable {
    optional func keyboardWillHide(userInfo: [NSObject: AnyObject])
}
```

*Note this pattern helps avoid overuse of `@objc`. See http://www.jessesquires.com/avoiding-objc-in-swift/ for more details!

In summary, I've found using a Singleton + Delegate to manage the keyboard is both more efficient and easier to use than using Notifications

# Section 64.4: Moving view up or down when keyboard is present

**Note: This only works for the built-in keyboard provided by iOS**
**SWIFT:**

In order for the view of a **UIViewController** to increase the origin of the frame when it is presented and decrease it when it is hidden, add the following functions to your class:

```swift
func keyboardWillShow(notification: NSNotification) {

    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?
NSValue)?.cgRectValue {
        if self.view.frame.origin.y == 0{
            self.view.frame.origin.y -= keyboardSize.height
        }
    }

}

func keyboardWillHide(notification: NSNotification) {
    if let keyboardSize = (notification.userInfo?[UIKeyboardFrameBeginUserInfoKey] as?
NSValue)?.cgRectValue {
        if self.view.frame.origin.y != 0{
            self.view.frame.origin.y += keyboardSize.height
        }
    }
}
```

And in the `viewDidLoad()` method of your class, add the following observers:

```swift
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillShow), name:
NSNotification.Name.UIKeyboardWillShow, object: nil)
NotificationCenter.default.addObserver(self, selector: #selector(Login.keyboardWillHide), name:
NSNotification.Name.UIKeyboardWillHide, object: nil)
```

And this will work for any screen size, using the height property of the keyboard.

**OBJECTIVE-C:**

To do the same thing in Objective-C, this code can be used:

```objc
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)
name:UIKeyboardWillShowNotification object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)
name:UIKeyboardWillHideNotification object:nil];
}

- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillShowNotification
object:nil];
    [[NSNotificationCenter defaultCenter] removeObserver:self name:UIKeyboardWillHideNotification
object:nil];
}

- (void)keyboardWillShow:(NSNotification *)notification
```

```
{
    CGSize keyboardSize = [[[notification userInfo] objectForKey:UIKeyboardFrameBeginUserInfoKey]
CGRectValue].size;

    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = -keyboardSize.height;
        self.view.frame = f;
    }];
}

-(void)keyboardWillHide:(NSNotification *)notification
{
    [UIView animateWithDuration:0.3 animations:^{
        CGRect f = self.view.frame;
        f.origin.y = 0.0f;
        self.view.frame = f;
    }];
}
```

# Section 64.5: Scrolling a UIScrollView/UITableView When Displaying the Keyboard

There are few approaches available there:

1.  You can subscribe for keyboard appearance events notifications and change offset manually:

```
//Swift 2.0+
override func viewDidLoad() {
    super.viewDidLoad()

    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillShow(_:)), name: UIKeyboardWillShowNotification, object: nil)
    NSNotificationCenter.defaultCenter().addObserver(self, selector:
#selector(YourVCClassName.keyboardWillHide(_:)), name: UIKeyboardWillHideNotification, object: nil)
}

func keyboardWillShow(notification: NSNotification) {
    if let userInfo = notification.userInfo {
        if let keyboardHeight = userInfo[UIKeyboardFrameEndUserInfoKey]?.CGRectValue.size.height {
            tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardHeight, 0)
        }
    }
}

func keyboardWillHide(notification: NSNotification) {
    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0)
}

//Objective-C
- (void)viewDidLoad {

    [super viewDidLoad];


    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillShow:)
name:UIKeyboardWillShowNotification object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(keyboardWillHide:)
name:UIKeyboardWillHideNotification object:nil];

}
```

```objc
- (void)keyboardWillShow:(NSNotification *)notification {

    NSDictionary *userInfo = [notification userInfo];

    if (userInfo) {

        CGRect keyboardEndFrame;
        [[userInfo objectForKey:UIKeyboardFrameEndUserInfoKey] getValue:&keyboardEndFrame];
        tableView.contentInset = UIEdgeInsetsMake(0, 0, keyboardEndFrame.size.height, 0);

    }

}

- (void)keyboardWillHide:(NSNotification *)notification {

    tableView.contentInset = UIEdgeInsetsMake(0, 0, 0, 0);

}
```

2. Or use ready-made solutions like TPKeyboardAvoidingTableView or TPKeyboardAvoidingScrollView
   https://github.com/michaeltyson/TPKeyboardAvoiding

# Chapter 65: Checking for Network Connectivity

## Section 65.1: Creating a Reachability listener

Apple's Reachability class periodically checks the network status and alerts observers to changes.

```
Reachability *internetReachability = [Reachability reachabilityForInternetConnection];
[internetReachability startNotifier];
```

## Section 65.2: Add observer to network changes

Reachability uses NSNotification messages to alert observers when the network state has changed. Your class will need to become an observer.

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(reachabilityChanged:)
name:kReachabilityChangedNotification object:nil];
```

Elsewhere in your class, implement method signature

```
- (void) reachabilityChanged:(NSNotification *)note {
    //code which reacts to network changes
}
```

## Section 65.3: Alert when network becomes unavailable

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    if (netStatus == NotReachable) {
        NSLog(@"Network unavailable");
    }
}
```

## Section 65.4: Alert when connection becomes a WIFI or cellular network

```
- (void)reachabilityChanged:(NSNotification *)note {
    Reachability* reachability = [note object];
    NetworkStatus netStatus = [reachability currentReachabilityStatus];

    switch (netStatus) {
        case NotReachable:
            NSLog(@"Network unavailable");
            break;
        case ReachableViaWWAN:
            NSLog(@"Network is cellular");
            break;
        case ReachableViaWiFi:
            NSLog(@"Network is WIFI");
            break;
    }
}
```

# Section 65.5: Verify if is connected to network

**Swift**

```swift
import SystemConfiguration

/// Class helps to code reuse in handling internet network connections.
class NetworkHelper {

    /**
     Verify if the device is connected to internet network.
     - returns:          true if is connected to any internet network, false if is not
     connected to any internet network.
     */
    class func isConnectedToNetwork() -> Bool {
        var zeroAddress = sockaddr_in()

        zeroAddress.sin_len = UInt8(sizeofValue(zeroAddress))
        zeroAddress.sin_family = sa_family_t(AF_INET)

        let defaultRouteReachability = withUnsafePointer(&zeroAddress) {
            SCNetworkReachabilityCreateWithAddress(nil, UnsafePointer($0))
        }

        var flags = SCNetworkReachabilityFlags()

        if !SCNetworkReachabilityGetFlags(defaultRouteReachability!, &flags) {
            return false
        }

        let isReachable = (flags.rawValue & UInt32(kSCNetworkFlagsReachable)) != 0
        let needsConnection = (flags.rawValue & UInt32(kSCNetworkFlagsConnectionRequired)) != 0

        return (isReachable && !needsConnection)
    }
}



if NetworkHelper.isConnectedToNetwork() {
    // Is connected to network
}
```

**Objective-C:**

we can check network connectivity within few lines of code as:

```objc
-(BOOL)isConntectedToNetwork
{
    Reachability *networkReachability = [Reachability reachabilityForInternetConnection];
    NetworkStatus networkStatus = [networkReachability currentReachabilityStatus];
    if (networkStatus == NotReachable)
    {
        NSLog(@"There IS NO internet connection");
        return false;
    } else
    {
        NSLog(@"There IS internet connection");
        return true;
    }
```

```
}
```

# Chapter 66: Accessibility

Accessibility in iOS allows users with hearing disabilities and visual impairments to access iOS and your application by supporting various features like VoiceOver, Voice Control, White on Black, Mono Audio, Speech to Text and so on. Providing Accessibility in the iOS app means making the app usable for everyone.

## Section 66.1: Make a View Accessible

Mark your `UIView` subclass as an accessible element so that it is visible to VoiceOver.

```
myView.isAccessibilityElement = YES;
```

Ensure that the view speaks a meaningful label, value, and hint. Apple provides more details on how to choose good descriptions in the [Accessibility Programming Guide](#).

## Section 66.2: Accessibility Frame

The accessibility frame is used by VoiceOver for hit testing touches, drawing the VoiceOver cursor, and calculating where in the focused element to simulate a tap when the user double-taps the screen. Note that the frame is in screen coordinates!

```
myElement.accessibilityFrame = frameInScreenCoordinates;
```

If your elements or screen layouts change often, consider overriding -`accessibilityFrame` to always provide an up-to-date rect. Calculating the screen-relative frame of scroll view subviews can be error-prone and tedious. iOS 10 introduces a new API to make this easier: `accessibilityFrameInContainerSpace`.

## Section 66.3: Layout Change

In many cases, content within a single screen will update with new or different content. For example, imagine a form that reveals additional options based on the user's answer to a previous question. In this case, a "layout change" notification lets you either announce the change or focus on a new element. This notification accepts the same parameters as the screen change notification.

```
UIAccessibilityPostNotification(UIAccessibilityLayoutChangedNotification, firstElement);
```

## Section 66.4: Accessibility Container

VoiceOver can navigate many apps on iOS because most `UIKit` classes implement `UIAccessibilityProtocol`. Features that don't represent onscreen elements using `UIView`, including apps that leverage Core Graphics or Metal to perform drawing, must describe these elements for accessibility. As of iOS 8.0, this can be done by assigning a property on the `UIView` containing inaccessible elements:

```
myInaccessibleContainerView.accessibilityElements = @[elements, that, should, be, accessible];
```

Each object in the array can be an instance of `UIAccessibilityElement` or any other class that adheres to `UIAccessibilityProtocol`. The child elements should be returned in the order the user should navigate them. As an application author, you can use accessibility containers to override the default top-left to bottom-right ordering of VoiceOver swipe navigation. Given that `UIView` implements `UIAccessibilityProtocol`, you can combine instances of `UIAccessibilityElement` and `UIView` in the same array of child accessibility elements. Note that if you assign elements manually, you do not need to implement any dynamic accessibility protocol methods, though you

may need to issue a screen change notification for the elements to be detected by VoiceOver.

## Section 66.5: Hiding Elements

Most UIKit classes, including UIView, adhere to `UIAccessibilityProtocol` and return correct values by default. It's easy to take for granted that a `UIView` set to hidden is also absent from the accessibility hierarchy and won't be navigated by VoiceOver. While this default behavior is usually sufficient, there are times where a view will be present in the view hierarchy but not visible or navigable. For example, a collection of buttons may be overlapped by another view, rendering them invisible to a sighted user. VoiceOver, however, will still try to navigate them since they are technically not hidden from `UIKit` and therefore are still present in the accessibility hierarchy. In such cases, you must hint to VoiceOver that the parent view isn't accessible. You can do this by explicitly hiding the view from UIKit by setting hidden when the view goes offscreen:

```
myViewFullofButtons.hidden = YES;
```

Alternatively, you can leave the parent view visible and simply hide its children from the accessibility hierarchy:

```
myViewFullofButtons.accessibilityElementsHidden = YES;
```

Temporary views are a another place you'll want to hide elements from the accessibility hierarchy while leaving them visible to users. For example, the view that pops up when you hit the volume button is visible to sighted users but doesn't demand attention the way a normal alert does. You wouldn't want VoiceOver to interrupt the user and move the cursor from away from whatever they were doing to announce the new volume, especially given that adjusting volume already provides auditory feedback through the clicking sound it makes. In cases like this, you'll want to hide the view using `accessibilityElementsHidden`.

## Section 66.6: Screen Change

VoiceOver works great most of the time, breezily reading aloud screens full of content and intuitively following the user. Alas, no general solution is perfect. Sometimes only you, the app developer, know where VoiceOver should be focused for an optimal user experience. Fortunately, VoiceOver listens to system accessibility notifications for clues about where focus belongs. To move the VoiceOver cursor manually, post an accessibility screen changed notification:

```
UIAccessibilityPostNotification(UIAccessibilityScreenChangedNotification, firstElement);
```

When this notification is posted, a short series of tones notify users of the change. The second parameter can be either the next element to focus or a string announcing the change. Only post a screen change notification if the VoiceOver experience is poor without it and no other workaround exists. Moving the VoiceOver cursor is like poking at a sighted user's screen. It can be annoying and disorienting to be led around that way.

## Section 66.7: Announcement

Announcements are useful for alerting users to events that don't require any interaction, such as "screen locked" or "finished loading." Use a more specific announcement to notify users of screen changes or more minor layout changes.

```
UIAccessibilityPostNotification(UIAccessibilityAnnouncementNotification, @"The thing happened!");
```

## Section 66.8: Ordering Elements

VoiceOver navigates from top-left to bottom-right, irrespective of the view hierarchy. This is usually how content is

arranged in left-to-right languages since sighted individuals tend to scan the screen in an "F-shaped pattern". VoiceOver users will expect to navigate the same way as typical users. Predictability and consistency are very important to accessibility. Please refrain from making customizations that "improve" on default behavior (eg. ordering the tab bar first in the swipe order). That said, if you have received feedback that the order of elements in your app is surprising, there are a couple of ways you can improve the experience.

If VoiceOver should read a view's subviews one after the next but is not, you may need to hint to VoiceOver that the elements contained within a single view are related. You can do this by setting `shouldGroupAccessibiltyChildren`:

```
myView.shouldGroupAccessibilityChildren = YES;
```

To support complex navigation structures that span multiple containers or include interfaces rendered without UIKit, consider implementing the container protocol on the parent view.

## Section 66.9: Modal View

Modal views completely capture the user's attention until a task is complete. iOS clarifies this to users by dimming and disabling all other content when a modal view, such as an alert or popover, is visible. An app that implements a custom modal interface needs to hint to VoiceOver that this view deserve the user's undivided attention by setting `accessibilityViewIsModal`. Note that this property should only be set on the view containing modal content, not elements contained within a modal view.

```
myModalView.accessibilityViewIsModal = YES;
```

Tagging a view as modal encourages VoiceOver to ignore sibling views. If, after setting this property, you find that VoiceOver still navigates other elements in your app, try hiding problem views until the modal dismisses.

# Chapter 67: Auto Layout

Auto Layout dynamically calculates the size and position of all the views in your view hierarchy, based on constraints placed on those views. [Source](#)

## Section 67.1: Space Views Evenly
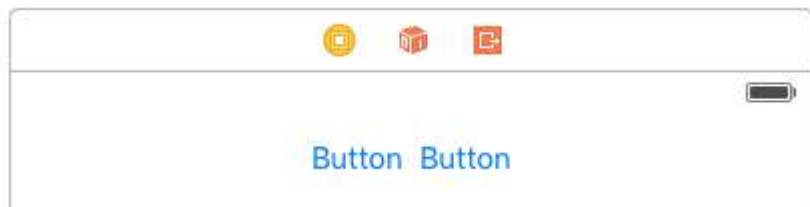


It is common to want two views to be side by side, centered in their superview. The common answer given on Stack Overflow is to embed these two views in a `UIView` and center the `UIView`. This is not necessary or recommended. From the [UILayoutGuide](#) docs:

> There are a number of costs associated with adding dummy views to your view hierarchy. First, there is the cost of creating and maintaining the view itself. Second, the dummy view is a full member of the view hierarchy, which means that it adds overhead to every task the hierarchy performs. Worst of all, the invisible dummy view can intercept messages that are intended for other views, causing problems that are very difficult to find.

You can use `UILayoutGuide` to do this, instead of adding the buttons into an unnecessary `UIView`. A `UILayoutGuide` is essentially a rectangular space that can interact with Auto Layout. You put a `UILayoutGuide` on the left and right sides of the buttons and set their widths to be equal. This will center the buttons. Here is how to do it in code:

*Visual Format Language style*

```
view.addSubview(button1)
view.addSubview(button2)

let leftSpace = UILayoutGuide()
view.addLayoutGuide(leftSpace)

let rightSpace = UILayoutGuide()
view.addLayoutGuide(rightSpace)

let views = [
    "leftSpace" : leftSpace,
    "button1" : button1,
    "button2" : button2,
    "rightSpace" : rightSpace
]

// Lay the buttons and layout guides out horizontally in a line.
// Put the layout guides on each end.
NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisualFormat("H:|[leftSpace][button1]-[button2][rightSpace]|", options: [], metrics: nil, views: views))

// Now set the layout guides widths equal, so that the space on the
// left and the right of the buttons will be equal
leftSpace.widthAnchor.constraintEqualToAnchor(rightSpace.widthAnchor).active = true
```

```
let leadingSpace = UILayoutGuide()
let trailingSpace = UILayoutGuide()
view.addLayoutGuide(leadingSpace)
view.addLayoutGuide(trailingSpace)

leadingSpace.widthAnchor.constraintEqualToAnchor(trailingSpace.widthAnchor).active = true

leadingSpace.leadingAnchor.constraintEqualToAnchor(view.leadingAnchor).active = true
leadingSpace.trailingAnchor.constraintEqualToAnchor(button1.leadingAnchor).active = true

trailingSpace.leadingAnchor.constraintEqualToAnchor(button2.trailingAnchor).active = true
trailingSpace.trailingAnchor.constraintEqualToAnchor(view.trailingAnchor).active = true
```

You will need to add vertical constraints to this as well, but this will center the buttons in the view without adding any "dummy" views! This will save the system from wasting CPU time on displaying those "dummy" views. This example uses buttons, but you can swap buttons out for any view you want to put constraints on.

If you are supporting iOS 8 or earlier the easiest way to create this layout is to add hidden dummy views. With iOS 9 you can replace dummy views with layout guides.

Note: Interface Builder does not support layout guides yet (Xcode 7.2.1). So if you want to use them you must create your constraints in code. Source.

# Section 67.2: Center Constraints

Select your button (or whatever view you want to center) on the **storyboard**. Then click the align button on the bottom right. Select `Horizontally in Container` and `Vertically in Container`. Click "Add 2 Constraints".
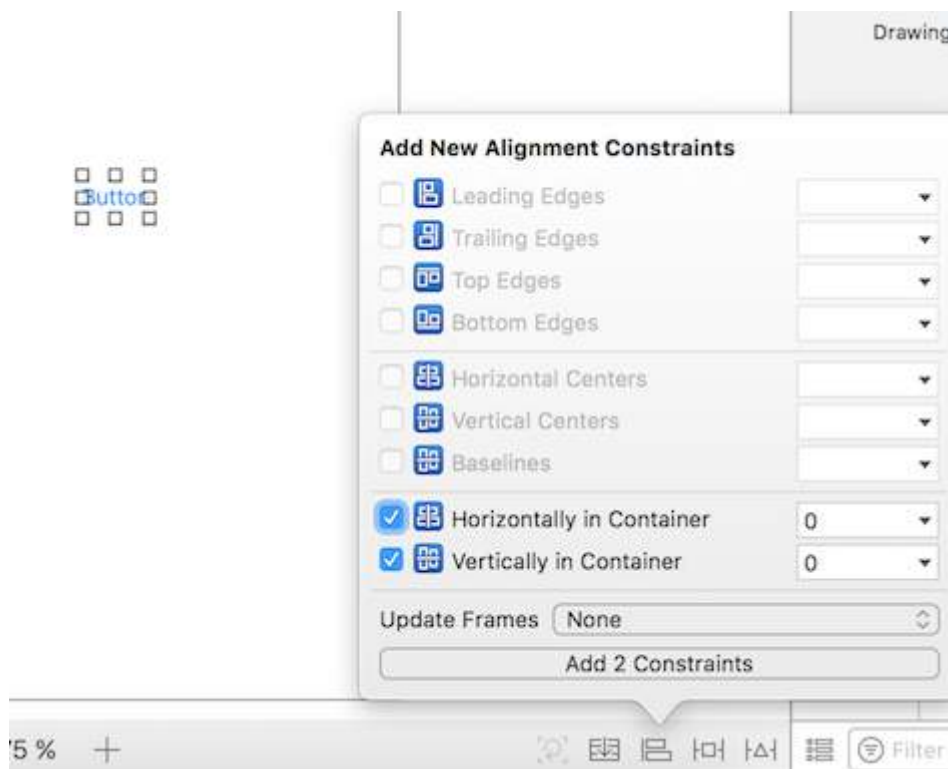


If it wasn't perfectly centered already, you may need to do one more thing. Click the "Update Frames" button that is two to the left of the "Embed In Stack" button on the bottom bar.
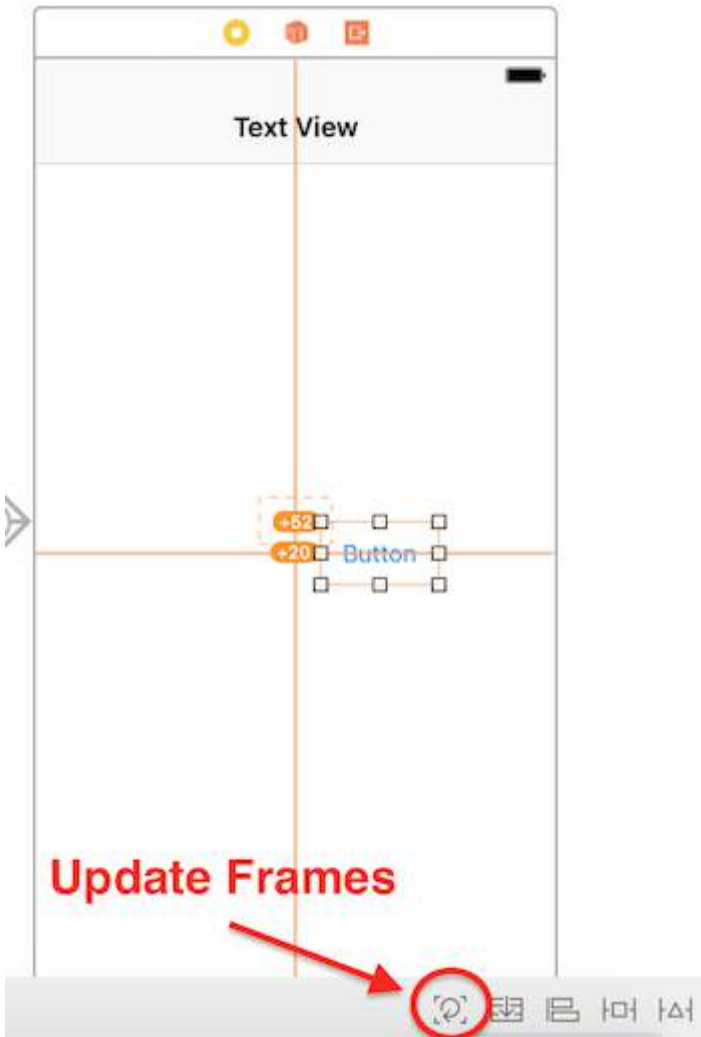
You can also "update frames as necessary" by pressing together ⌘ + ⌥ + = (Command + Option and equals) after selecting the view, this might save some time.

Now when you run your app it should be centered, no matter what device size you are using.

Another way to center views using Interface Builder is by control-click-dragging. Say you want to center a `UILabel` in a view. Open the `Document Outline` in your storyboard by clicking the sidebar button at the bottom left. Click and drag from the label to the view while holding `ctrl` (control), and a blue line should appear:
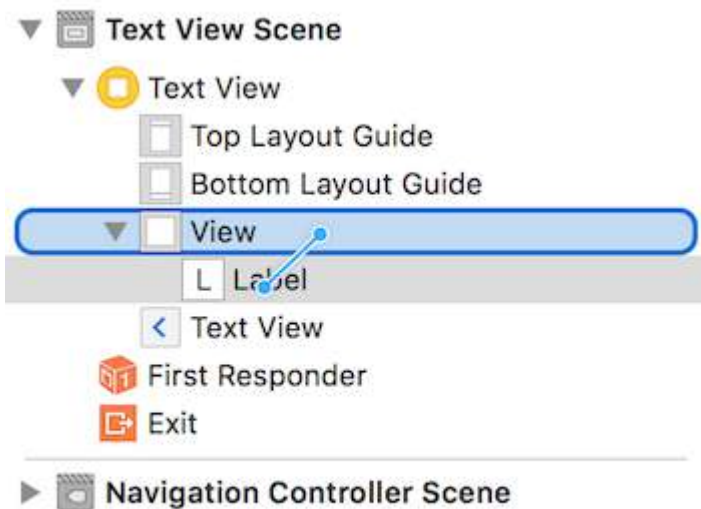
Upon release, a menu of constraint options will appear:



Select "Center Horizontally in Container" and "Center Vertically in Container". Update frames as necessary, and voila! A centered label.

Alternatively, you can add the constraints programmatically. Create the constraints and add them to the desired UI elements and views as the following example describes, where we create a button and align it in the center, horizontally and vertically to its superview:

**Objective-C**

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    UIButton *yourButton = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 100, 18)];
    [yourButton setTitle:@"Button" forState:UIControlStateNormal];


    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterY relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterY multiplier:1 constant:0]]; //Align veritcally center to
superView

    [self.view addConstraint:[NSLayoutConstraint constraintWithItem:yourButton
attribute:NSLayoutAttributeCenterX relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeCenterX multiplier:1 constant:0]]; //Align horizontally center to
superView

    [self.view addSubview:yourButton]; //Add button to superView
}
```

**Swift**

```
override func viewDidLoad()
{
    super.viewDidLoad()
    let yourButton: UIButton = UIButton(frame: CGRect(x: 0, y: 0, width: 100, height: 18))
    yourButton.setTitle("Button", forState: .Normal)

    let centerVertically = NSLayoutConstraint(item: yourButton,
                                  attribute: .CenterX,
                                  relatedBy: .Equal,
                                     toItem: view,
                                  attribute: .CenterX,
                                 multiplier: 1.0,
                                   constant: 0.0)
```

```swift
    let centerHorizontally = NSLayoutConstraint(item: yourButton,
                                    attribute: .CenterY,
                                    relatedBy: .Equal,
                                       toItem: view,
                                    attribute: .CenterY,
                                   multiplier: 1.0,
                                     constant: 0.0)
    NSLayoutConstraint.activateConstraints([centerVertically, centerHorizontally])
}
```

# Section 67.3: Setting constraints programmatically

Boilerplate code example

```swift
override func viewDidLoad() {
    super.viewDidLoad()

    let myView = UIView()
    myView.backgroundColor = UIColor.blueColor()
    myView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(myView)

    // Add constraints code here
    // ...
}
```

In the examples below the Anchor Style is the preferred method over NSLayoutConstraint Style, however it is only available from iOS 9, so if you are supporting iOS 8 then you should still use NSLayoutConstraint Style.

**Pinning**

*Anchor Style*

```swift
let margins = view.layoutMarginsGuide
myView.leadingAnchor.constraintEqualToAnchor(margins.leadingAnchor, constant: 20).active = true
```

- In addition to leadingAnchor, there is also trailingAnchor, topAnchor, and bottomAnchor.

*NSLayoutConstraint Style*

```swift
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.LeadingMargin, multiplier: 1.0,
constant: 20.0).active = true
```

- In addition to .Leading there is also .Trailing, .Top, and .Bottom.
- In addition to .LeadingMargin there is also .TrailingMargin, .TopMargin, and .BottomMargin.

*Visual Format Language style*

```swift
NSLayoutConstraint.constraintsWithVisualFormat("H:|-20-[myViewKey]", options: [], metrics: nil,
views: ["myViewKey": myView])
```

**Width and Height**

*Anchor Style*

```swift
myView.widthAnchor.constraintEqualToAnchor(nil, constant: 200).active = true
myView.heightAnchor.constraintEqualToAnchor(nil, constant: 100).active = true
```

*NSLayoutConstraint Style*

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Width, relatedBy:
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,
constant: 200).active = true
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Height, relatedBy:
NSLayoutRelation.Equal, toItem: nil, attribute: NSLayoutAttribute.NotAnAttribute, multiplier: 1,
constant: 100).active = true
```

*Visual Format Language style*

```
NSLayoutConstraint.constraintsWithVisualFormat("H:[myViewKey(200)]", options: [], metrics: nil,
views: ["myViewKey": myView])
NSLayoutConstraint.constraintsWithVisualFormat("V:[myViewKey(100)]", options: [], metrics: nil,
views: ["myViewKey": myView])
```

**Center in container**

*Anchor Style*

```
myView.centerXAnchor.constraintEqualToAnchor(view.centerXAnchor).active = true
myView.centerYAnchor.constraintEqualToAnchor(view.centerYAnchor).active = true
```

*NSLayoutConstraint Style*

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterX, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterX, multiplier: 1,
constant: 0).active = true
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.CenterY, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.CenterY, multiplier: 1,
constant: 0).active = true
```

*Visual Format Language style*

```
NSLayoutConstraint.constraintsWithVisualFormat("V:[viewKey]-(<=0)-[myViewKey]", options:
NSLayoutFormatOptions.AlignAllCenterX, metrics: nil, views: ["myViewKey": myView, "viewKey": view])
NSLayoutConstraint.constraintsWithVisualFormat("H:[viewKey]-(<=0)-[myViewKey]", options:
NSLayoutFormatOptions.AlignAllCenterY, metrics: nil, views: ["myViewKey": myView, "viewKey": view])
```

# Section 67.4: UILabel & Parentview size According to Text in UILabel

**Step by Step Guide:**

**Step 1:** Set constraint to UIView

1. Leading. 2) Top. 3) Trailing. (From mainview)
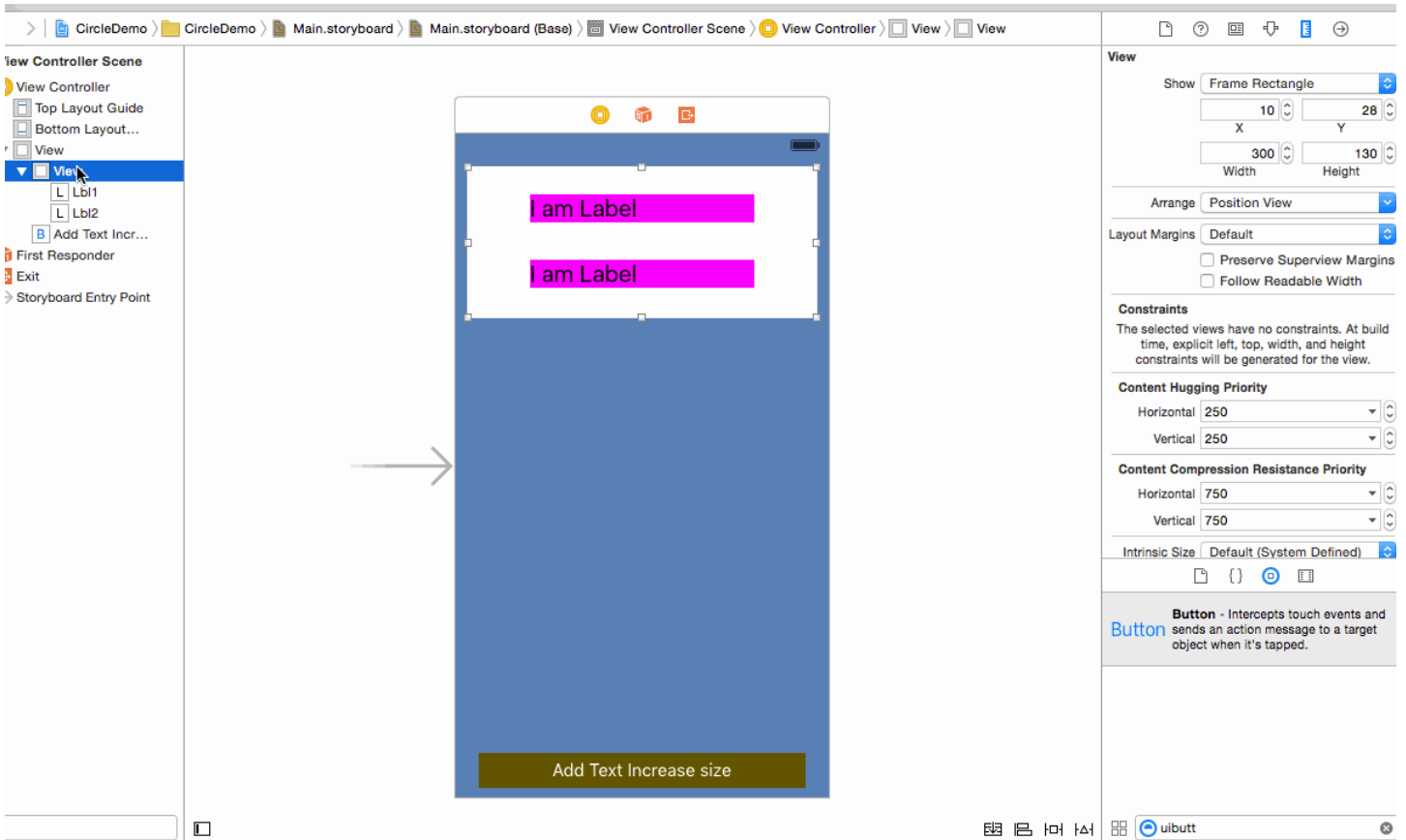
**Step 2:** Set constrain to Label 1

1. Leading 2) Top 3) Trailing (From it's superview)
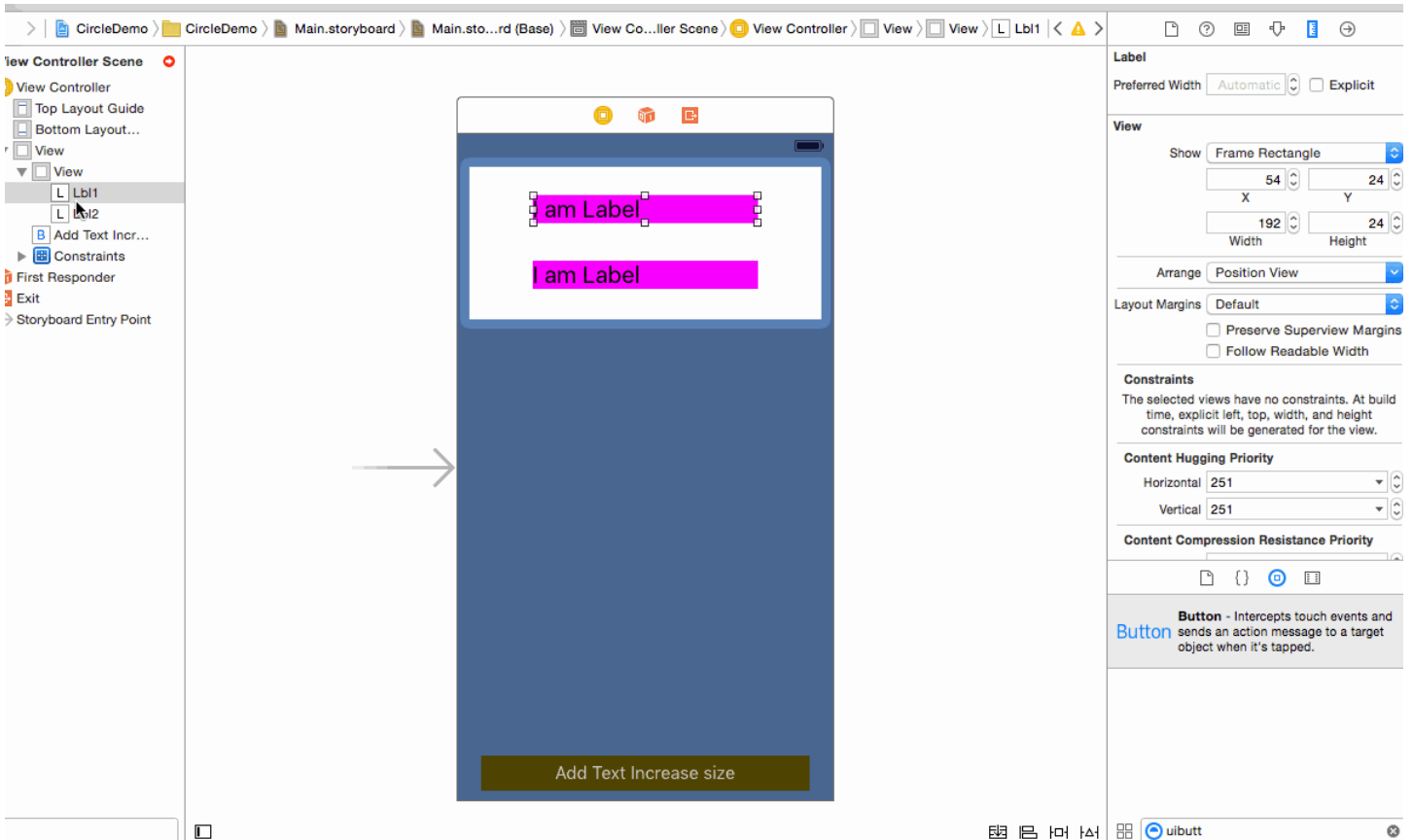


**Step 3:** Set constraint to Label 2

1. Leading 2) Top 3) Trailing (From its superview)
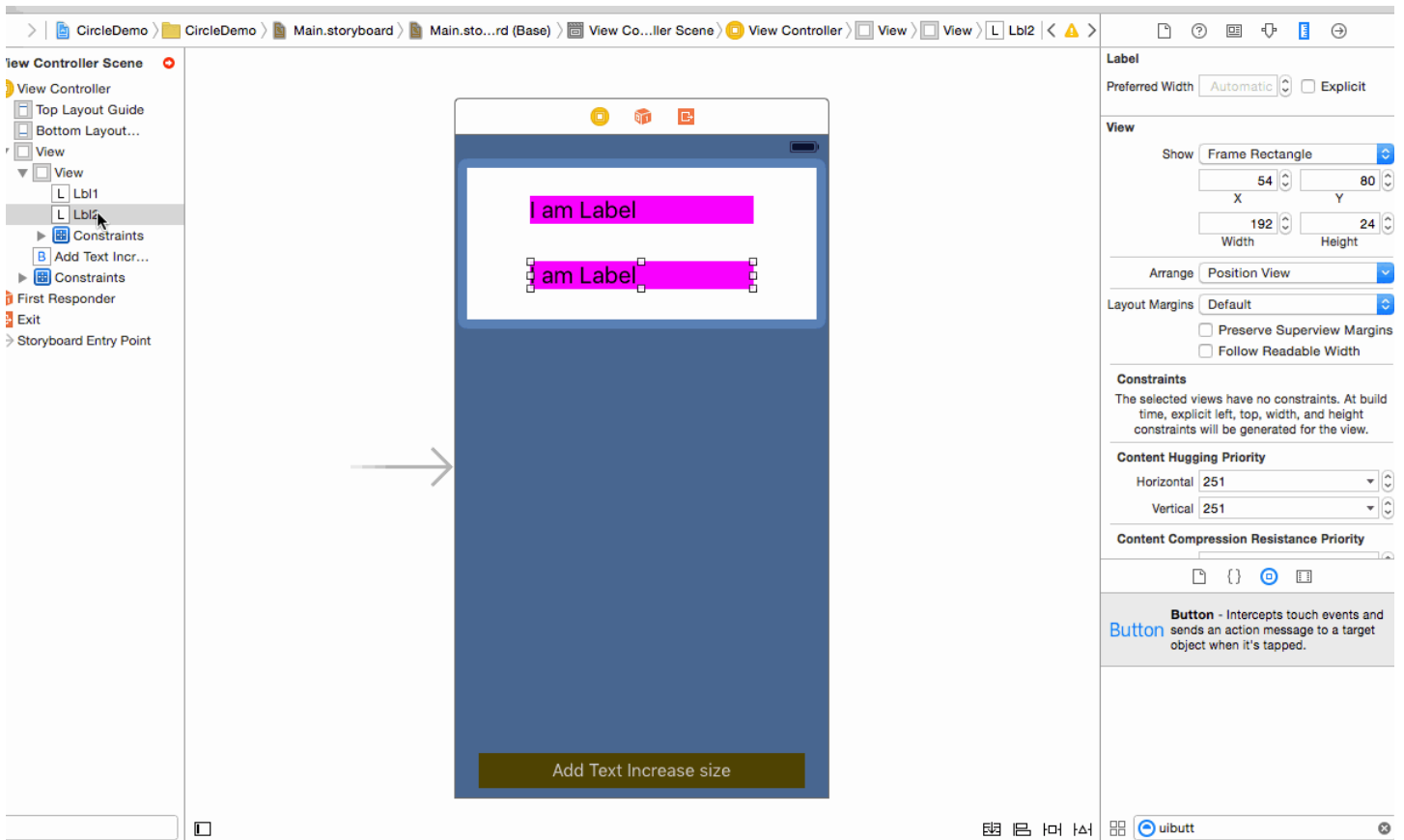
**Step 4: Most tricky** give a bottom to UILabel from UIView .
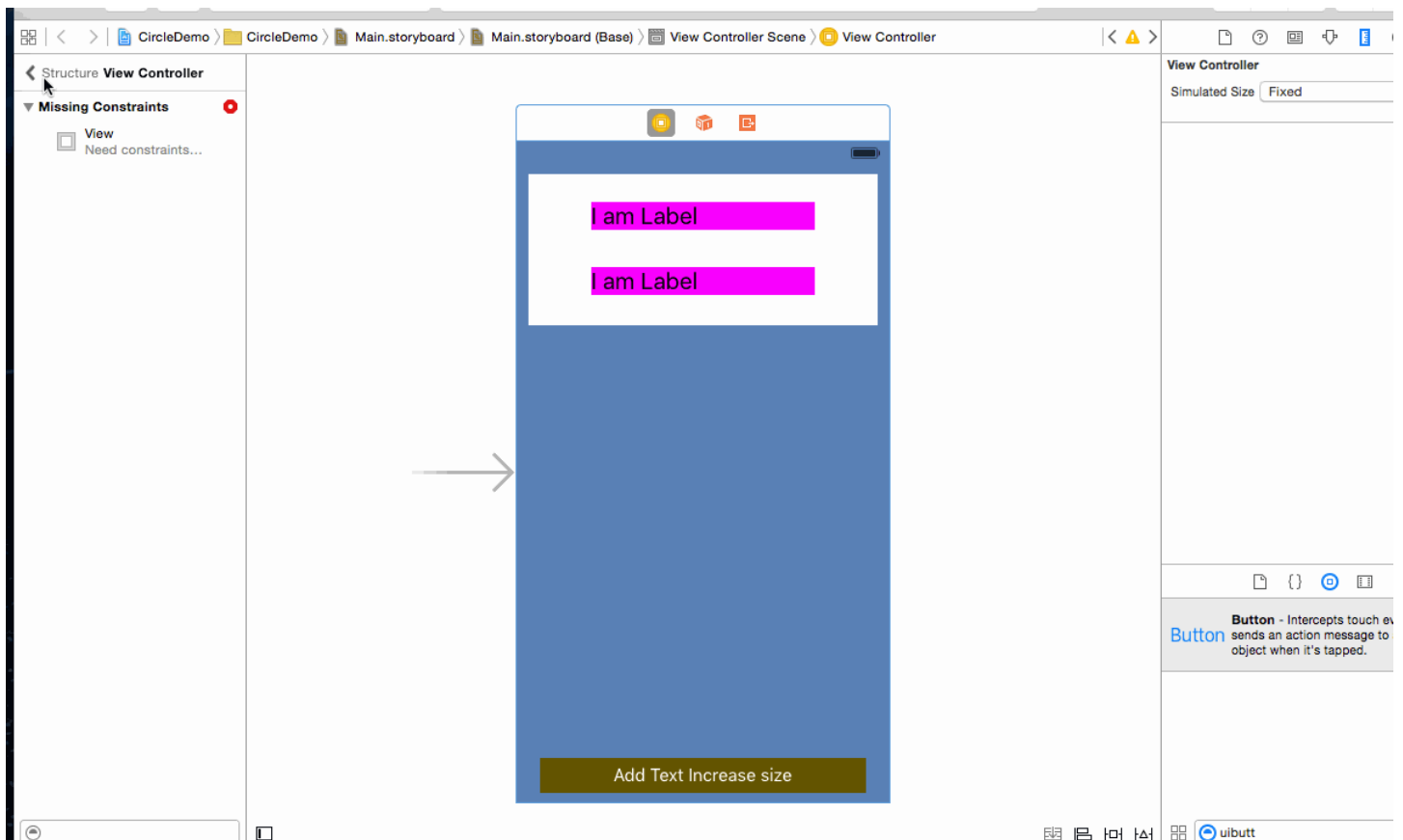


**Step 5:** (Optional) Set constrain to UIButton

1. Leading 2) Bottom 3) Trailing 4) Fixed Height (From mainview)
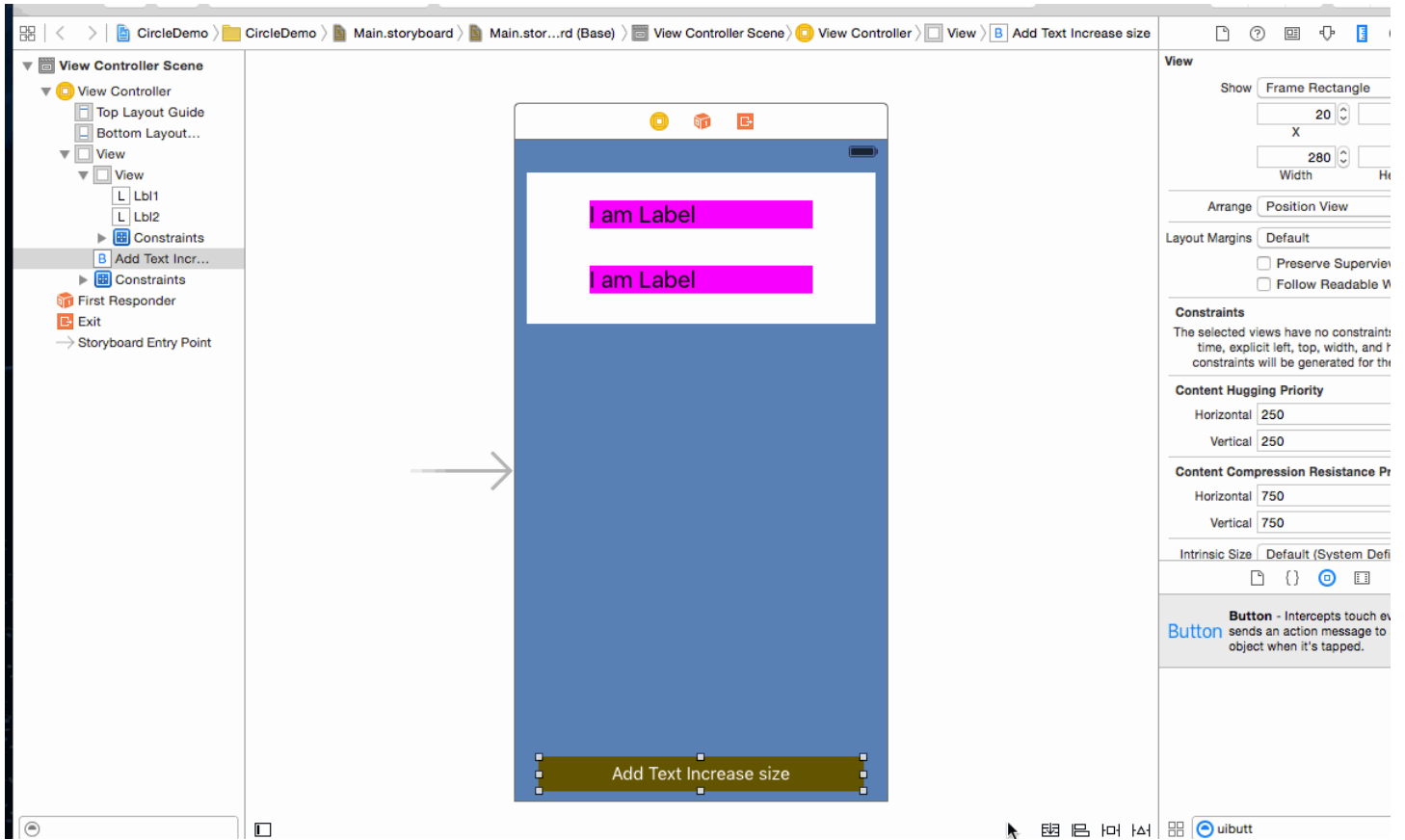
**Output:**

**Note:** Make sure you have set Number of lines =0 in Label property.
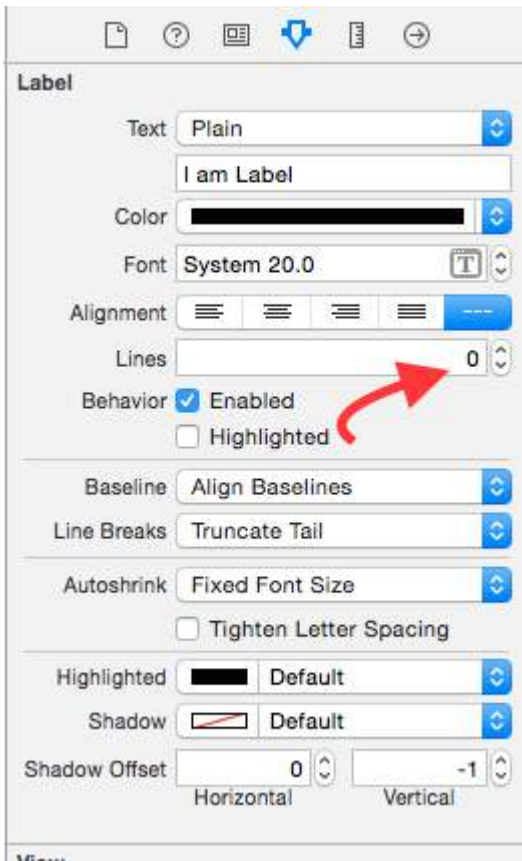
I hope this info enough to understand Autoresize UIView according to UILabel's height and Autoresize UILabel According to text.

## Section 67.5: UILabel Intrinsic Size

We have to create a view which will have a image prefix to a text. text could be of variable length.We have to achieve a result where in Image + text is always in center of a parent view.

text + image always in center

**Step 1:** First create a single view project and name it something of your choice and open the story board fist view.Drag a view with some reasonable size and set its background color to yellow.I have resized my viewcontroller to 3.5?.The resultant view should look some thing like this
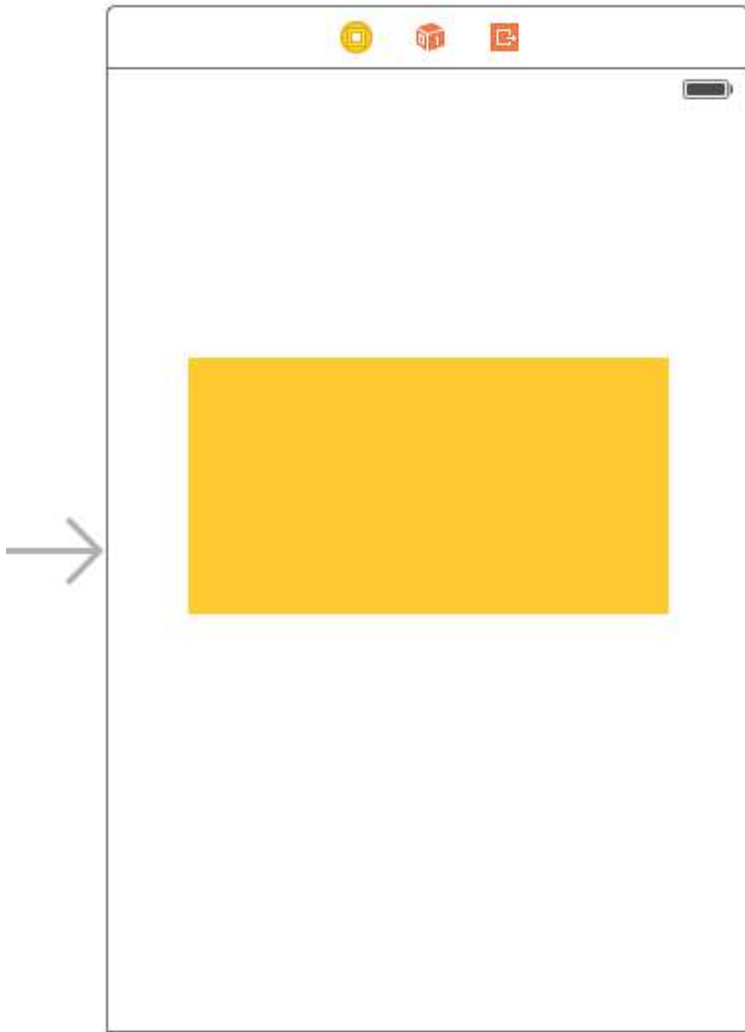
**Step 2:** Now we will add constraints to the yellow view .To begin with we will add width and height constraints (Wait a minute didn't we say that view will have dynamic width?Ok we will get back to it later) Add the following constraints as per the image below do not bother with width value any value will do just fine for width just keep it large enough so that we can add autolayouts properly.
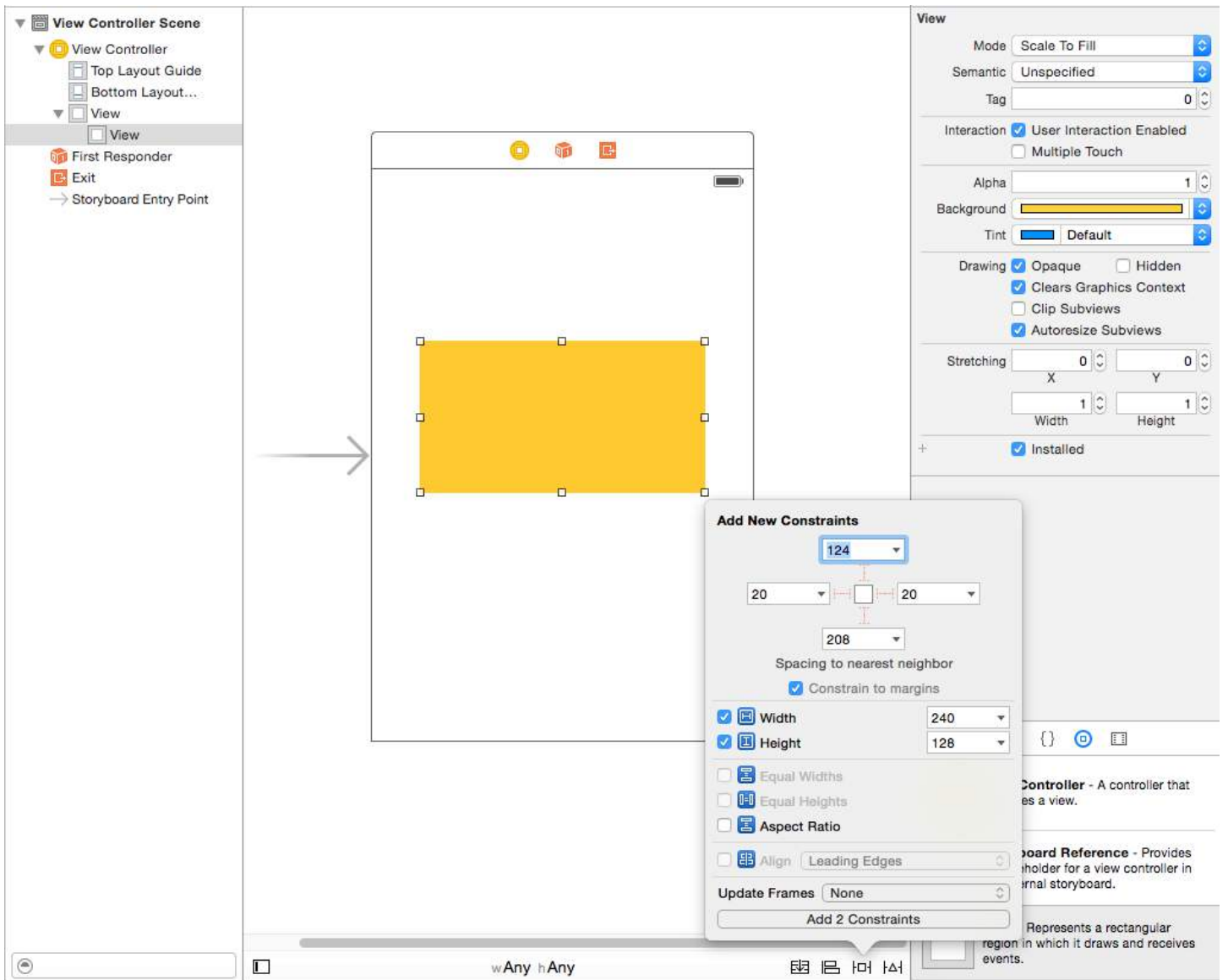
After adding these two constraints you will see that XCode is giving you errors as in below image lets see them and understand them.
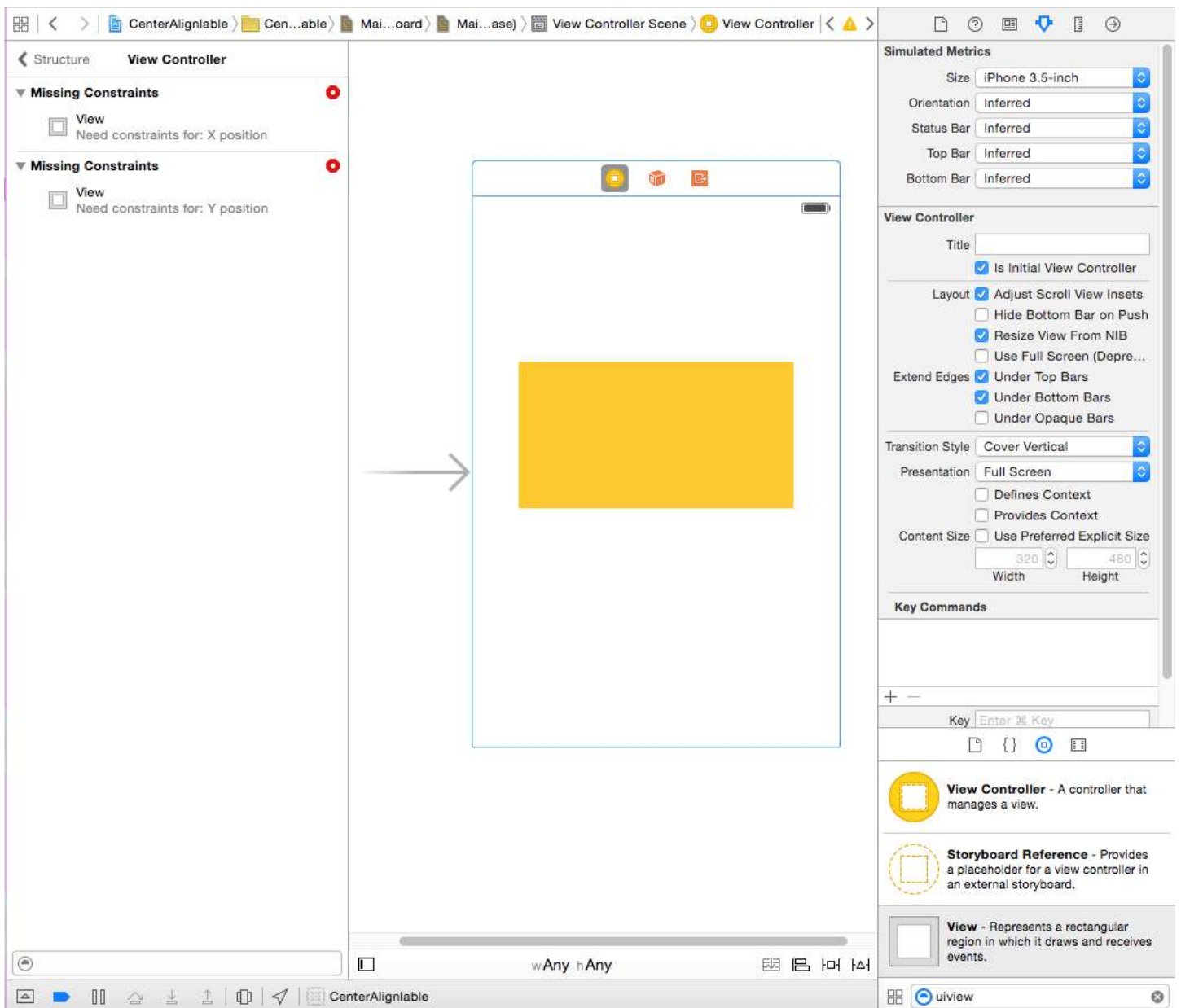
We have two errors (red means error)As discussed above lets revisit the ambiguity part

**Missing Constraints :** Need constraints for : X position: As discussed above we have given the view a width and a height so its "BOUNDS" is defined but we have not given its origin so its "FRAME" is not defined. Autolayout is not able to determine what will be the X position of our yellow view

**Missing Constraints :** Need constraints for : Y position: As discussed above we have given the view a width and a height so its "BOUNDS" is defined but we have not given its origin so its "FRAME" is not defined. Autolayout is not able to determine what will be the Y position of our yellow view To solve this we have to give autolayout some thing to resole X and Y. Since we cannot set frames we will do it autolayout way.Add following constraints as per the image given below I will explain it later
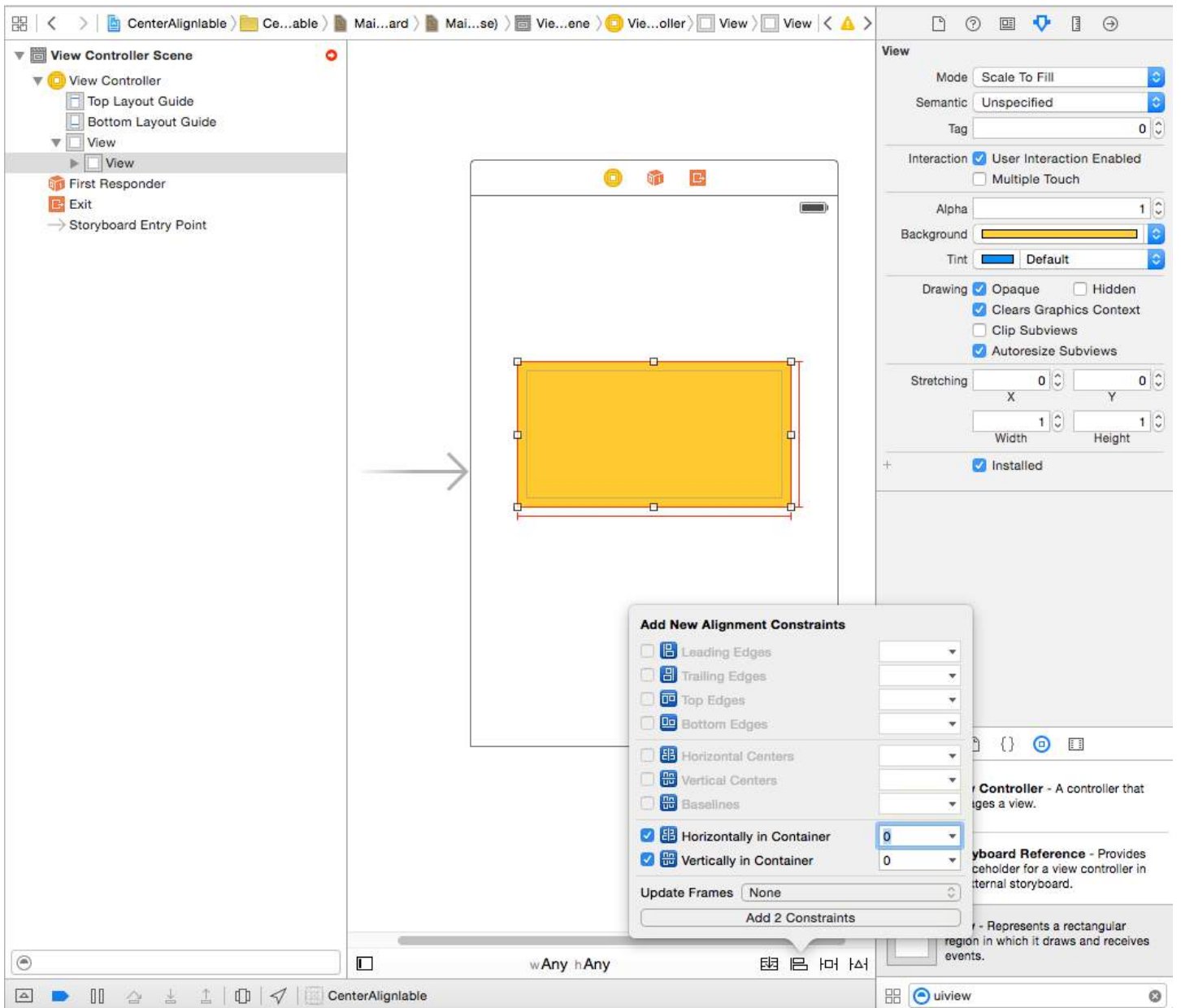
What we have done is,We have added a "Vertical Center" and "Horizontal Center" these constrain tell autolayout that our yellow view will always be in center Horizontally: so X in determined same is with vertical constraint and Y is determined.(you might have to adjust frame).

**Step 3:** By now our base yellow view is ready. We will add the prefix image as subview of our yellow view with following constraints.You can choose any image of your choice.
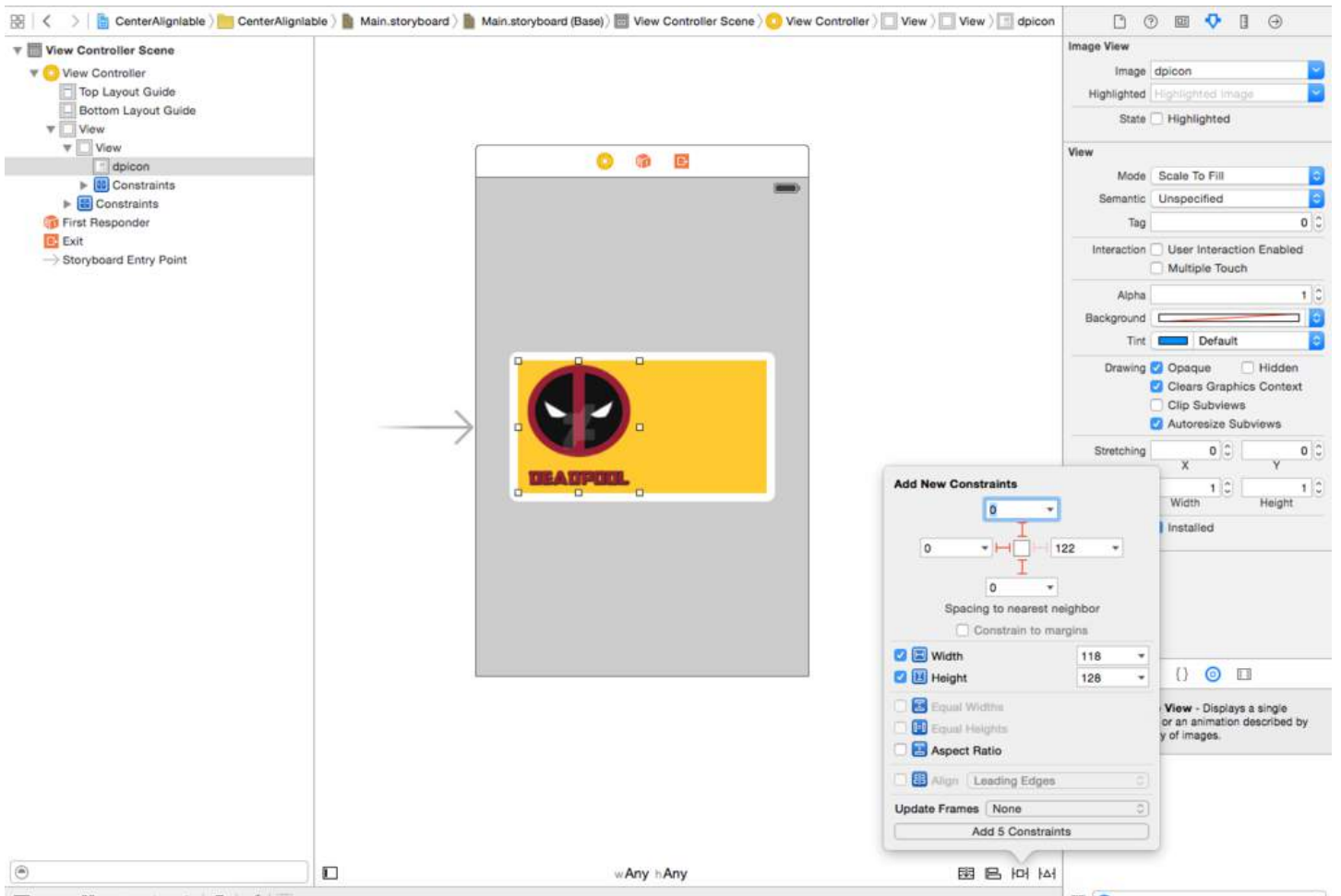
Since we have fixed dimension for our prefix image we will have fixed width height for this imageview. Add the constraints and proceed to next step.

**Step4:** Add a UILabel as the sub view of our yellow view and add following constraints
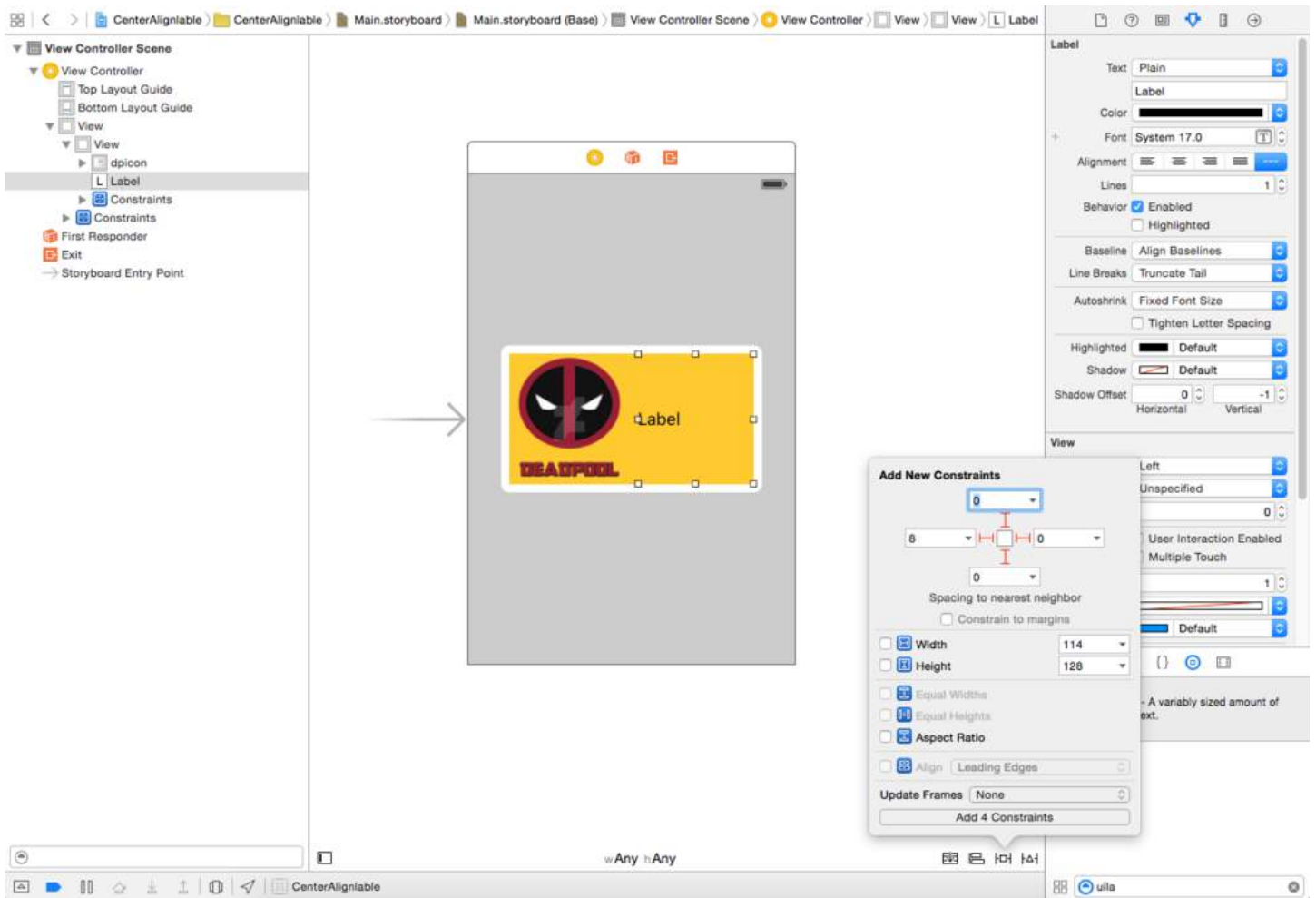
As you can see i have given only relative constraints to our UILabel.Its 8 points from prefix image and 0,0,0 top trailing and bottom from yellow view.Since we want the width to be dynamic we will not give width or height constraints.

Q: Why we are not getting any errors now , we have not given any width and height? Ans: We get error or warning only when auto layout is not able to resolve any thing which is must to render a view on screen.Be it height width or origin.Since our label is relative to yellow view and prefix image and their frames is well defined autolayout is able to calculate the frame of our Label.

**Step 5:** Now if we recall we will realize that we have given fixed view to out yellow view but we want it to be dynamic dependent upon the text of our label.So We will modify our Width Constraint of yellow view.Width of yellow view is necessary to resolve ambiguity but we want it to be overruled at runtime based upon the content of UILabel. So we will select our yellow view and go to Size inspector and reduce the priority of width constraint to 1 so that it will be over ruled. Follow the image given below.
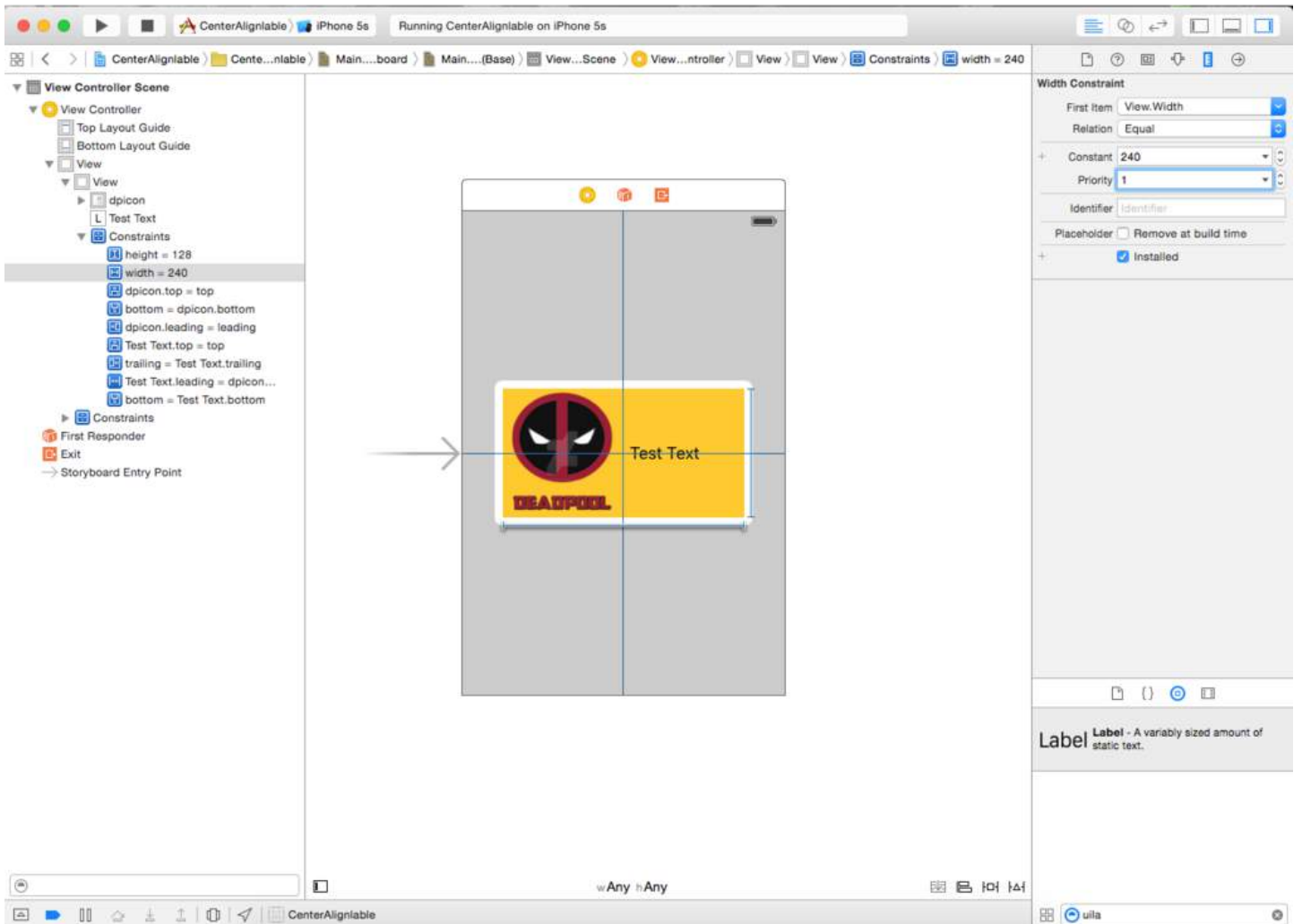
**Step 6:** We want out UILabel to expand according to text and push our yellow view.So we have reduced the priority of yellow view width.Now we will increase the priority of text compression resistance of our UILabel.We want our view to reduce as well so we will increase the priority of content hugging of UILabel.Follow the image below
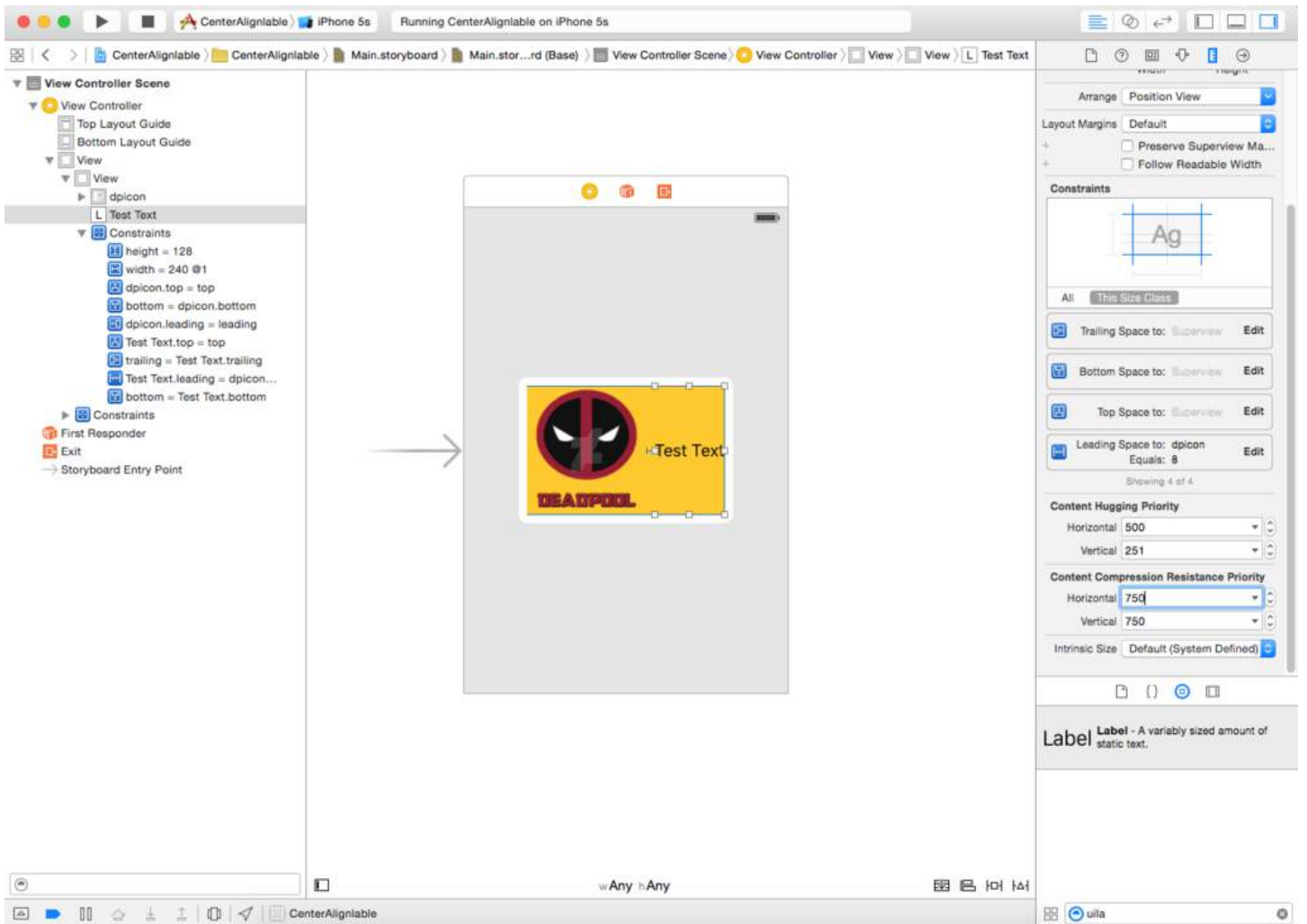
As you can see we have increased content hugging priority to 500 and compression resistance priority to 751 which will successfully over rule the width constraint's 1 priority.

Now build and run you will see some thing as following.
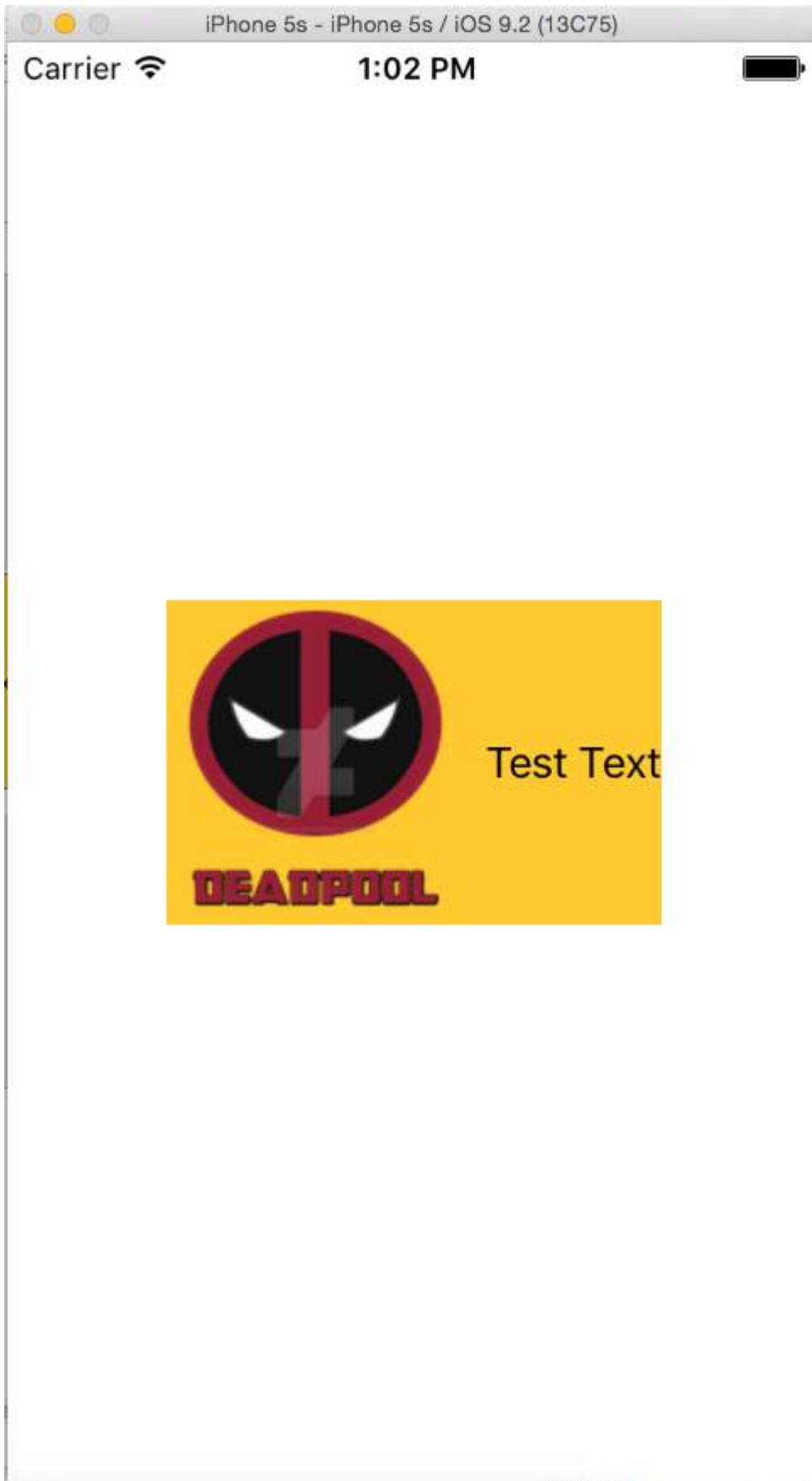
## Section 67.6: Visual Format Language Basics: Constraints in Code!

HVFL is a language designed to constrain UI elements in a simple and quick fashion. Generally, VFL has an advantage over traditional UI customization in the Interface Builder because it's much more readable, accessible and compact.

Here's an example of VFL, in which three UIViews are constrained from left to right, filling up `superView.width`, with aGradeView

```
"H:|[bgView][aGradeView(40)][bGradeView(40)]|"
```

There are two axes in which we can constrain UI Objects to, Horizontally and Vertically.

Each line of VFL always begins with `H:` or `V:`. If neither are present, the default option is `H:`

Moving on, we have a pipeline. `|` This symbol, or the pipe, refers to the superview. If you take a closer look at the snippet of VFL code above, you'd notice two of these pipelines.

This signifies the two horizontal ends of the superview, the outerleft and outerright boundaries.

Next up you'll see some square brackets, within the first set of square brackets, we have `bgView`. When we've got square brackets, it's referring to a UI element, now you might wonder how we establish a link between the name and the actual UI element, an outlet perhaps?

I'll cover that at the end of the post.

If you take a look at the second pair of square brackets `[aGradeView(50)]`, we have some parentheses encapsulated within as well, when that is present, it defines the width/height depending on the axes, which in this case is 50 pixels in width.

The first square brackets `[bgView]` did not have a width explicitly defined, meaning that it'll span out as far as possible.

Alright, that's it for the basics, more on the advanced stuff in another example.

for example:



```objc
// 1. create views
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];
```

```
    // 2. forbid Autoresizing
    blueView.translatesAutoresizingMaskIntoConstraints = NO;
    redView.translatesAutoresizingMaskIntoConstraints = NO;

    // 3. make contraints
    // horizontal
    NSArray *blueH = [NSLayoutConstraint constraintsWithVisualFormat:@"H:|-20-[blueView]-20-|"
options:NSLayoutFormatAlignAllLeft metrics:nil views:@{@"blueView" : blueView}];
    [self.view addConstraints:blueH];

    // vertical
    NSArray *blueVandRedV = [NSLayoutConstraint constraintsWithVisualFormat:@"V:|-20-
[blueView(50)]-20-[redView(==blueView)]" options:NSLayoutFormatAlignAllTrailing metrics:nil
views:@{@"blueView" : blueView, @"redView" : redView}];
    [self.view addConstraints:blueVandRedV];

    NSLayoutConstraint *redW = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeWidth multiplier:0.5 constant:0];
    [self.view addConstraint:redW];
```

# Section 67.7: Resolve UILabel Priority Conflict

**Problem**: When you use many labels inside a view, you maybe get a **warning**:
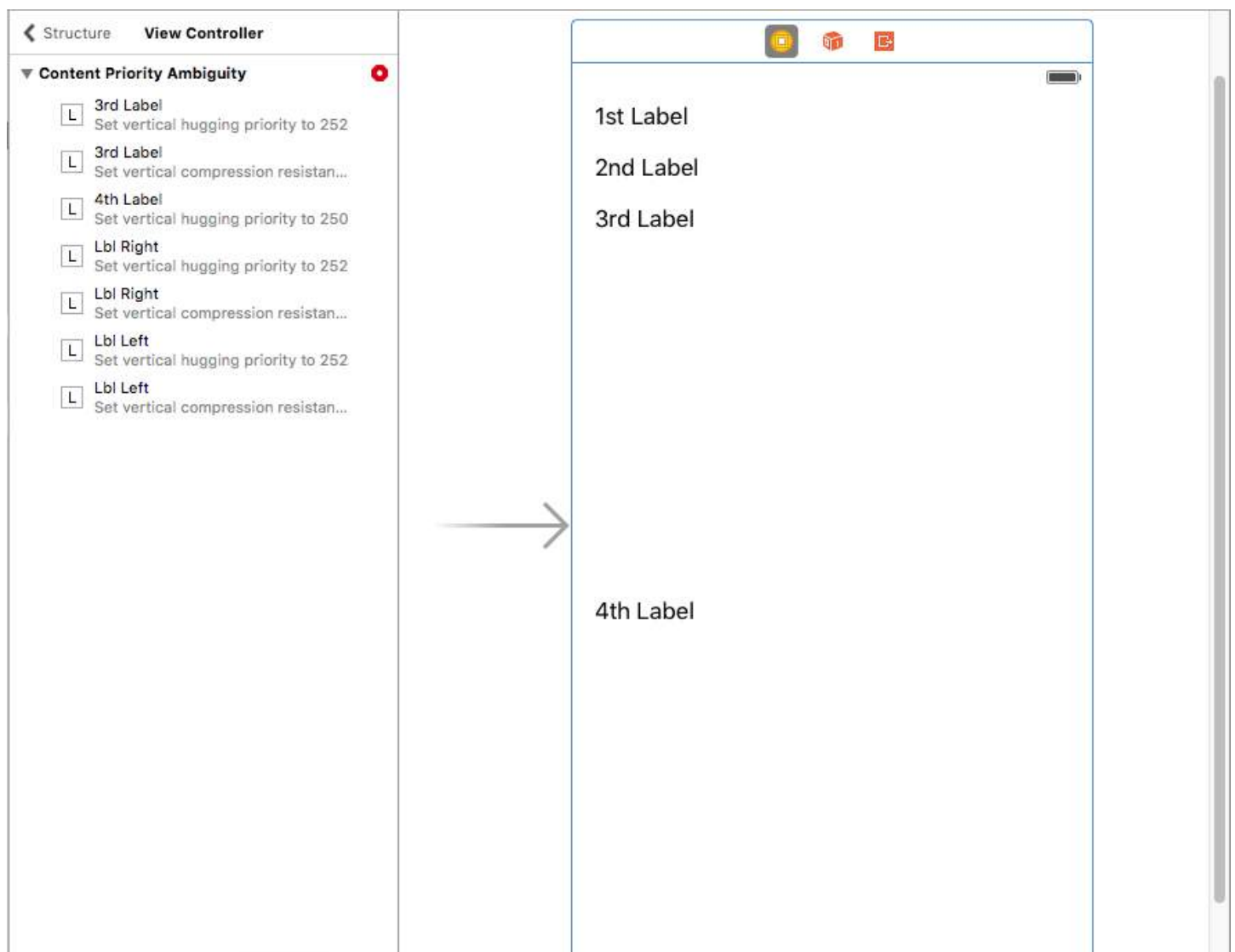


How can we fix this **warning**?

**Solution**: We calculate and set the priorities in order. The priorities must be different from labels. It means which is important will get higher priority. For example, in my case, I set the vertical priorities for my labels look like this:

I set the highest priority for 1st label and the lowest for 4th label.
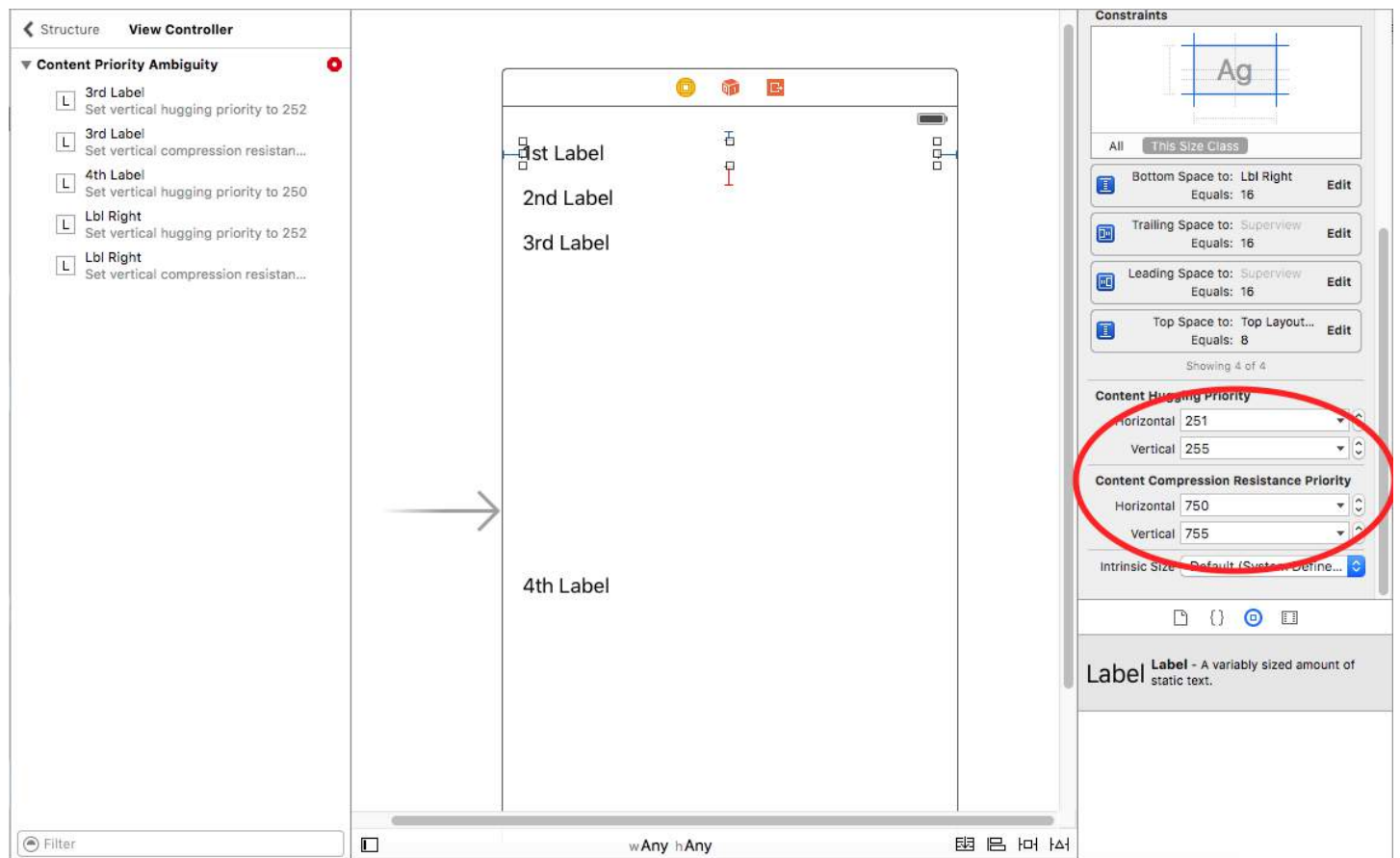
In a ViewController, I think you're hard to see the effect of those priorities. However, it's very clearly with UITableViewCell + estimate cell height.

Hope this help.

# Section 67.8: How to animate with Auto Layout

Without Auto Layout, animation is accomplished changing a view's frame over time. With Auto Layout, the constraints dictate the view frame, so you have to animate the constraints instead. This indirection makes animation harder to visualize.

Here are the ways to animate with Auto Layout:

1. **Change the constant of the constraint** after creation using periodic calls (`CADisplayLink`, `dispatch_source_t`, `dispatch_after`, `NSTimer`). Then call `layoutIfNeeded` to update the constraint. Example:

**Objective-C:**

```
self.someConstraint.constant = 10.0;
[UIView animateWithDuration:0.25 animations:^{
    [self.view layoutIfNeeded];
}];
```

**Swift:**

```
self.someConstraint.constant = 10.0
UIView.animate(withDuration: 0.25, animations: self.view.layoutIfNeeded)
```

2. **Change the constraints** and call `[view layoutIfNeeded]` inside an animation block. This interpolates between the two positions ignoring constraints during the animation.

```
[UIView animateWithDuration:0.5 animations:^{
    [view layoutIfNeeded];
}]
```

3. **Change the priority of the constraints**. This is less CPU intensive than adding and removing constraints.

4. **Remove all constraints and use autosizing masks**. For the later, you have to set `view.translatesAutoresizingMaskIntoConstraints = YES`.

5. **Use constraints that don't interfere with the intended animation**.

6. **Use a container view**. Position the superview using constraints. Then add a subview with constraints that don't fight the animation, eg: a center relative to the superview. This unloads part of the constraints to the superview, so they don't fight the animation in the subview.

7. **Animate layers instead views**. Layer transforms don't trigger the Auto Layout.

```
CABasicAnimation* ba = [CABasicAnimation animationWithKeyPath:@"transform"];
ba.autoreverses = YES;
ba.duration = 0.3;
ba.toValue = [NSValue valueWithCATransform3D:CATransform3DMakeScale(1.1, 1.1, 1)];
[v.layer addAnimation:ba forKey:nil];
```

8. **Override layoutSubviews**. Call `[super layoutSubviews]` and fine tune the constraints.

9. **Change the frame in viewDidLayoutSubviews**. Auto Layout is applied in `layoutSubviews`, so once done,

change it in `viewDidLayoutSubviews`.

10. **Opt out from Auto Layout** and set views manually. You can do this overriding `layoutSubviews/layout` without calling the super class's implementation.

Quick tip: if the parent of the animated view is not being interpolated (that is, the animation jumps from beginning to end state), call `layoutIfNeeded()` in the deepest view that is the parent of the view that is animated (in other words, that is not affected by the animation). I don't know exactly why this works.

## Section 67.9: NSLayoutConstraint: Contraints in code!

When we are working on a framework, if the constraints are not too complex, we'd better use Interface Builder or NSLayoutConstraint in code to make it smaller enough, instead of import Masonry or SnapKit.

for example:
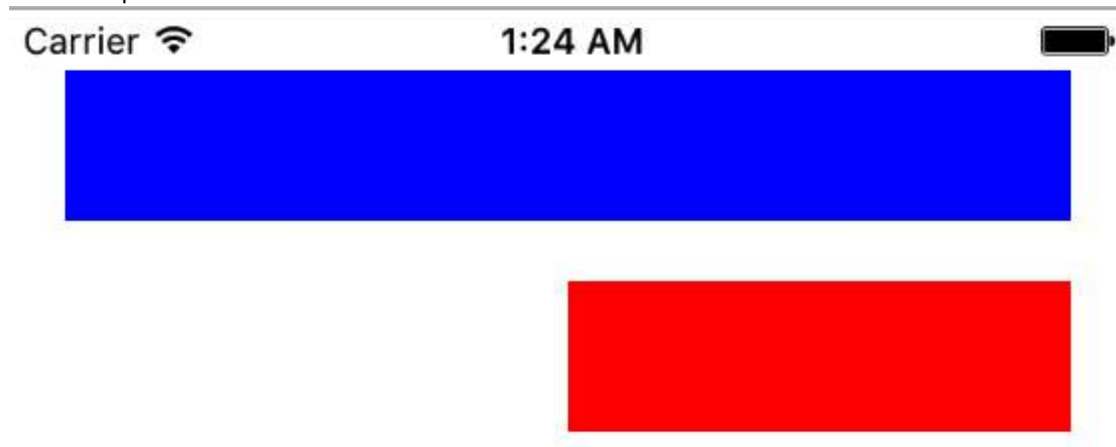


- Objective-C

```
// 1. create views
UIView *blueView = [[UIView alloc] init];
blueView.backgroundColor = [UIColor blueColor];
[self.view addSubview:blueView];

UIView *redView = [[UIView alloc] init];
redView.backgroundColor = [UIColor redColor];
[self.view addSubview:redView];

// 2. forbid Autoresizing
blueView.translatesAutoresizingMaskIntoConstraints = NO;
redView.translatesAutoresizingMaskIntoConstraints = NO;

// 3. make contraints
// 3.1 blueView
NSLayoutConstraint *blueLeft = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeLeft relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeLeft multiplier:1 constant:20];
    [self.view addConstraint:blueLeft];

NSLayoutConstraint *blueTop = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeTop multiplier:1 constant:20];
    [self.view addConstraint:blueTop];

NSLayoutConstraint *blueRight = [NSLayoutConstraint constraintWithItem:blueView
```

```
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view addConstraint:blueRight];

    NSLayoutConstraint *blueHeight = [NSLayoutConstraint constraintWithItem:blueView
attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:nil
attribute:NSLayoutAttributeNotAnAttribute multiplier:1 constant:50];
    [self.view addConstraint:blueHeight];

    // 3.2 redView
    NSLayoutConstraint *redTop = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeTop relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeBottom multiplier:1 constant:20];
    [self.view addConstraint:redTop];

    NSLayoutConstraint *redRight = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeRight relatedBy:NSLayoutRelationEqual toItem:self.view
attribute:NSLayoutAttributeRight multiplier:1 constant:-20];
    [self.view addConstraint:redRight];

    NSLayoutConstraint *redHeight = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeHeight relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeHeight multiplier:1 constant:0];
    [self.view addConstraint:redHeight];

    NSLayoutConstraint *redWidth = [NSLayoutConstraint constraintWithItem:redView
attribute:NSLayoutAttributeWidth relatedBy:NSLayoutRelationEqual toItem:blueView
attribute:NSLayoutAttributeWidth multiplier:0.5 constant:0];
    [self.view addConstraint:redWidth];
```

# Section 67.10: Proportional Layout

Constraint created as

```
NSLayoutConstraint(item: myView, attribute: NSLayoutAttribute.Leading, relatedBy:
NSLayoutRelation.Equal, toItem: view, attribute: NSLayoutAttribute.LeadingMargin, multiplier: 1.0,
constant: 20.0)
```

or, from math point of view:

```
view.attribute * multiplier + constant          (1)
```

You can use multiplier to create proportional layout for different size factor.

Example:

Turquoise View (V1) is a square with width proportional superview width with ratio 1:1.1

Gary square(V2) is a subview of V1. Bottom space set by constant = 60, Trailing space set by multiplier = 1.125 and constant = 0

Trailing space set proportionally, bottom space set as a constant.
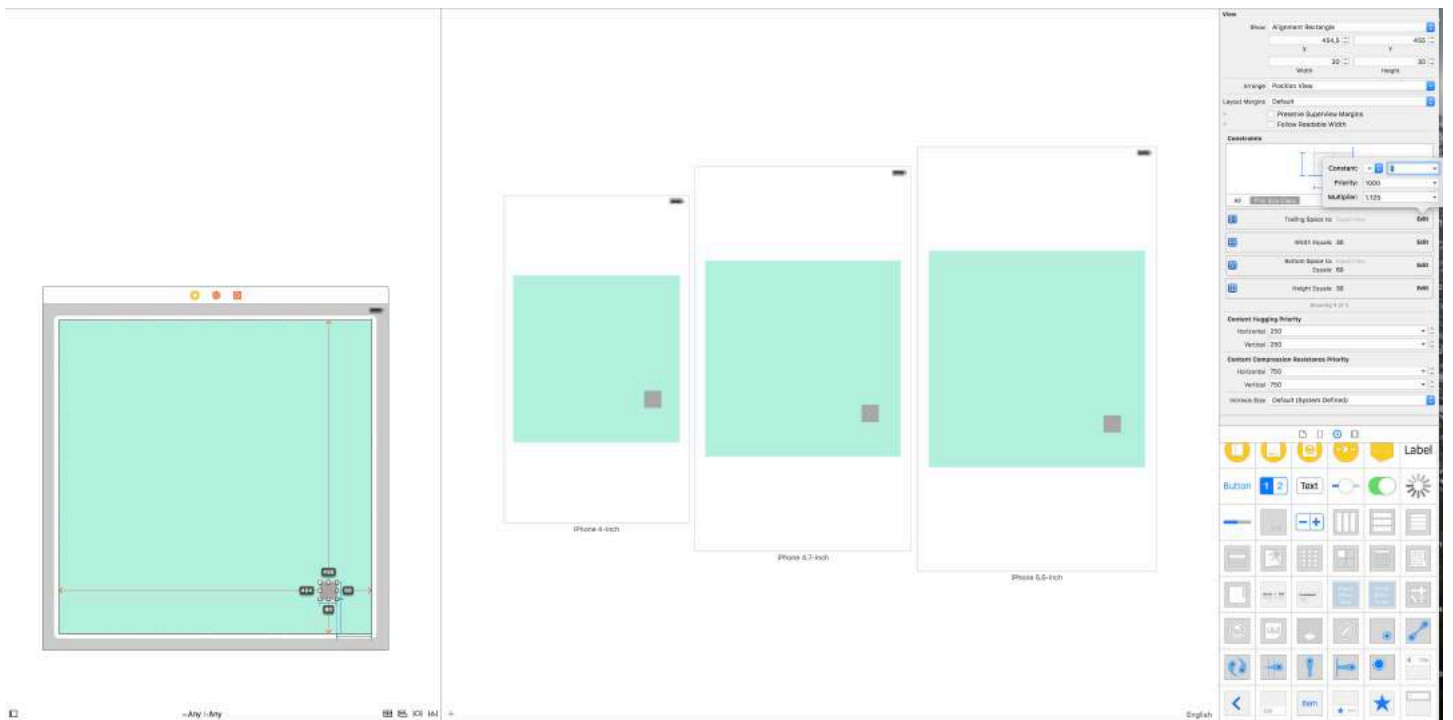
Note: if view.attribute is equal 0 (for example leading space), constraint formula (1), will be equal 0. You need to change second item of constraint or set constraint relative to margin, in order to view.attribute != 0.

# Section 67.11: Mixed usage of Auto Layout with non-Auto Layout

Sometimes you may want to perform some additional actions to **Auto Layout** calculations done by `UIKit` itself.

**Example:** when you have a `UIView` that has a `maskLayer`, you may need to update `maskLayer` as soon as **Auto Layout** changes `UIView`'s `frame`

```
// CustomView.m
- (void)layoutSubviews {
    [super layoutSubviews];
    // now you can assume Auto Layout did its job
    // you can use view's frame in your calculations
    CALayer maskLayer = self.maskLayer;
    maskLayer.bounds = self.bounds;
    ...
}
```

or if you want to take some additional action to **Auto Layout** in `ViewController`

```
- (void)viewDidLayoutSubviews {
    [super viewDidLayoutSubviews];
    // now you can assume all your subviews are positioned/resized correctly
    self.customView.frame = self.containerView.frame;
}
```

# Section 67.12: How to use Auto Layout

Auto layout is used to arrange views so that they look good on any device and orientation. Constraints are the rules that tell how everything should be laid down. They include pinning edges, centering, and setting sizes, among other things.

Auto layout is enabled by default, but you can double check this. If you click *Main.storyboard* in the Project Navigator and then show the File inspector. Make sure that Auto Layout and Size Classes are checked:
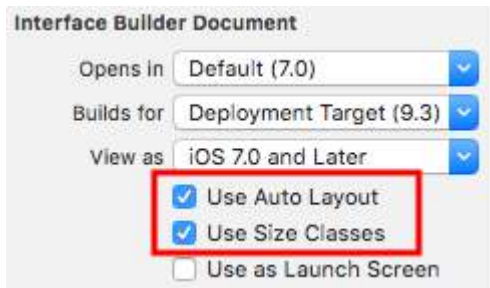


Auto layout constraints can be set in the Interface Builder or in code. In the Interface Builder you find the Auto Layout tools at the bottom right. Clicking them will reveal different options for setting the constraints on a view.
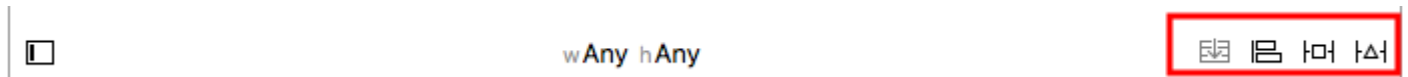


If you wish to have different constraints for different device sizes or orientations, you can set them in wAny hAny Size Class options found in the bottom middle.
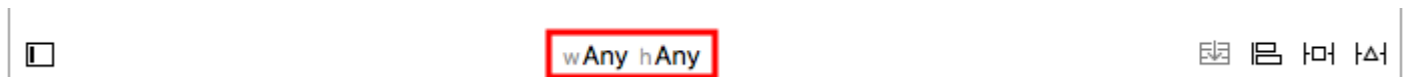
# Chapter 68: MKMapView

## Section 68.1: Change map-type

There are 5 different types (MKMapType), MKMapView can display.

Version ≥ iPhone OS 3
**.standard**

Displays a street map that shows the position of all roads and some road names.

**Swift 2**

```
mapView.mapType = .Standard
```

**Swift 3**

```
mapView.mapType = .standard
```

**Objective-C**

```
_mapView.mapType = MKMapTypeStandard;
```

**.satellite**

Displays satellite imagery of the area.

**Swift 2**

```
mapView.mapType = .Satellite
```

**Swift 3**

```
mapView.mapType = .satellite
```

**Objective-C**

```
_mapView.mapType = MKMapTypeSatellite;
```

**.satelliteFlyover**

Displays a satellite image of the area with flyover data where available.

**Swift 2**

```
mapView.mapType = .SatelliteFlyover
```

**Swift 3**

```
mapView.mapType = .satelliteFlyover
```

**Objective-C**

```
_mapView.mapType = MKMapTypeSatelliteFlyover;
```

**.hybrid**

Displays a satellite image of the area with road and road name information layered on top.

**Swift 2**

```
mapView.mapType = .Hybrid
```

**Swift 3**

```
mapView.mapType = .hybrid
```

**Objective-C**

```
_mapView.mapType = MKMapTypeHybrid;
```
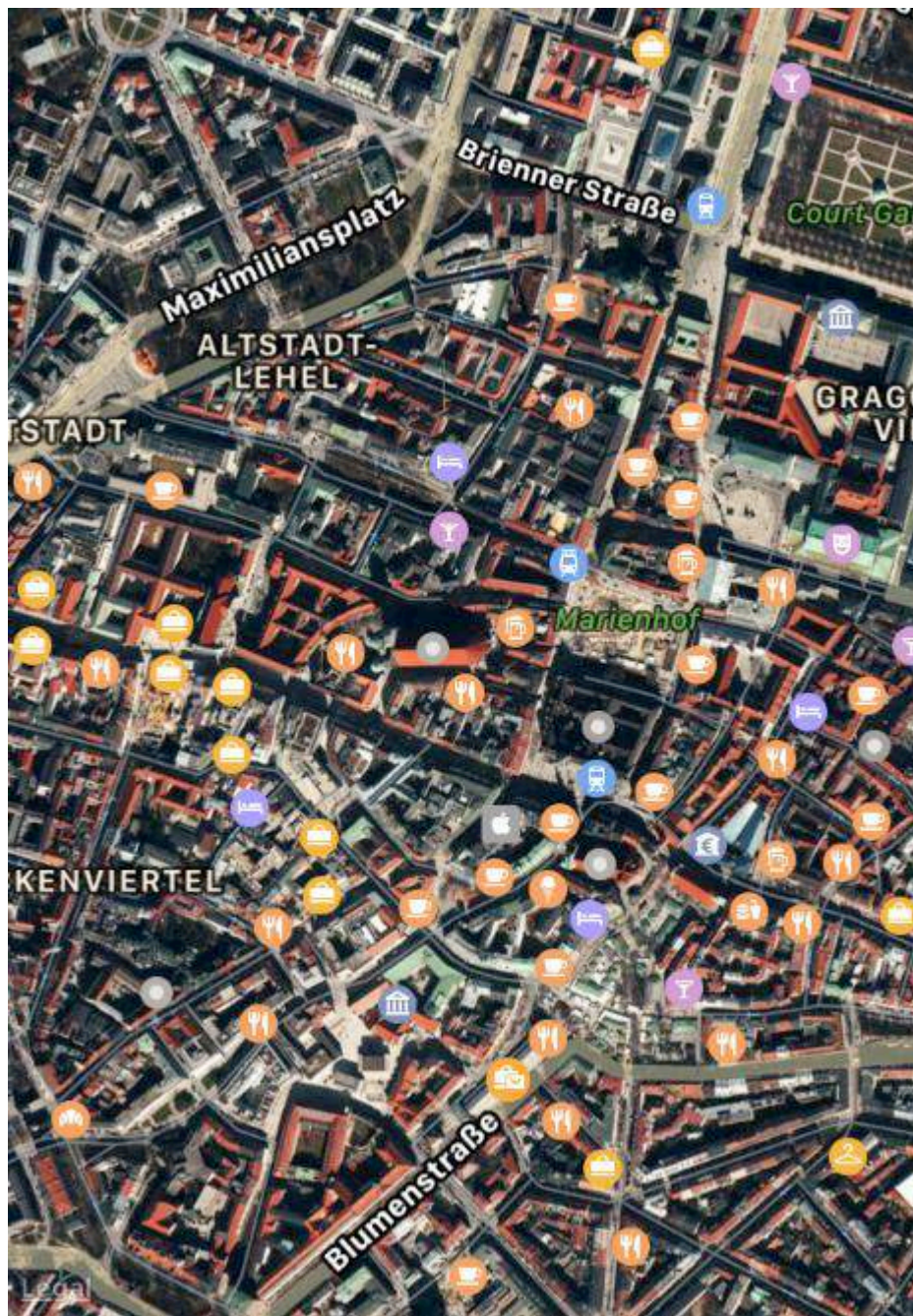


```
Version ≥ iOS 9
```
**.hybridFlyover**

Displays a hybrid satellite image with flyover data where available.

**Swift 2**

```
mapView.mapType = .HybridFlyover
```

**Swift 3**

```
mapView.mapType = .hybridFlyover
```

**Objective-C**

```
_mapView.mapType = MKMapTypeHybridFlyover;
```

# Section 68.2: Simulate a custom location

Step 1: In Xcode: File -> New -> File -> Resource -> GPX File -> Next -> Give the GPX file a name(It's `Taipei` in this example) -> Create

Step 2: Edit the GPX file

```
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
    <wpt lat="25.041865" lon="121.551361"> // Edit the latitude and longitude
        <name>Taipei</name> // Edit the name of the location
        <time>2014-09-24T14:55:37Z</time>
    </wpt>
</gpx>
```

Step 3: When the simulator is running:
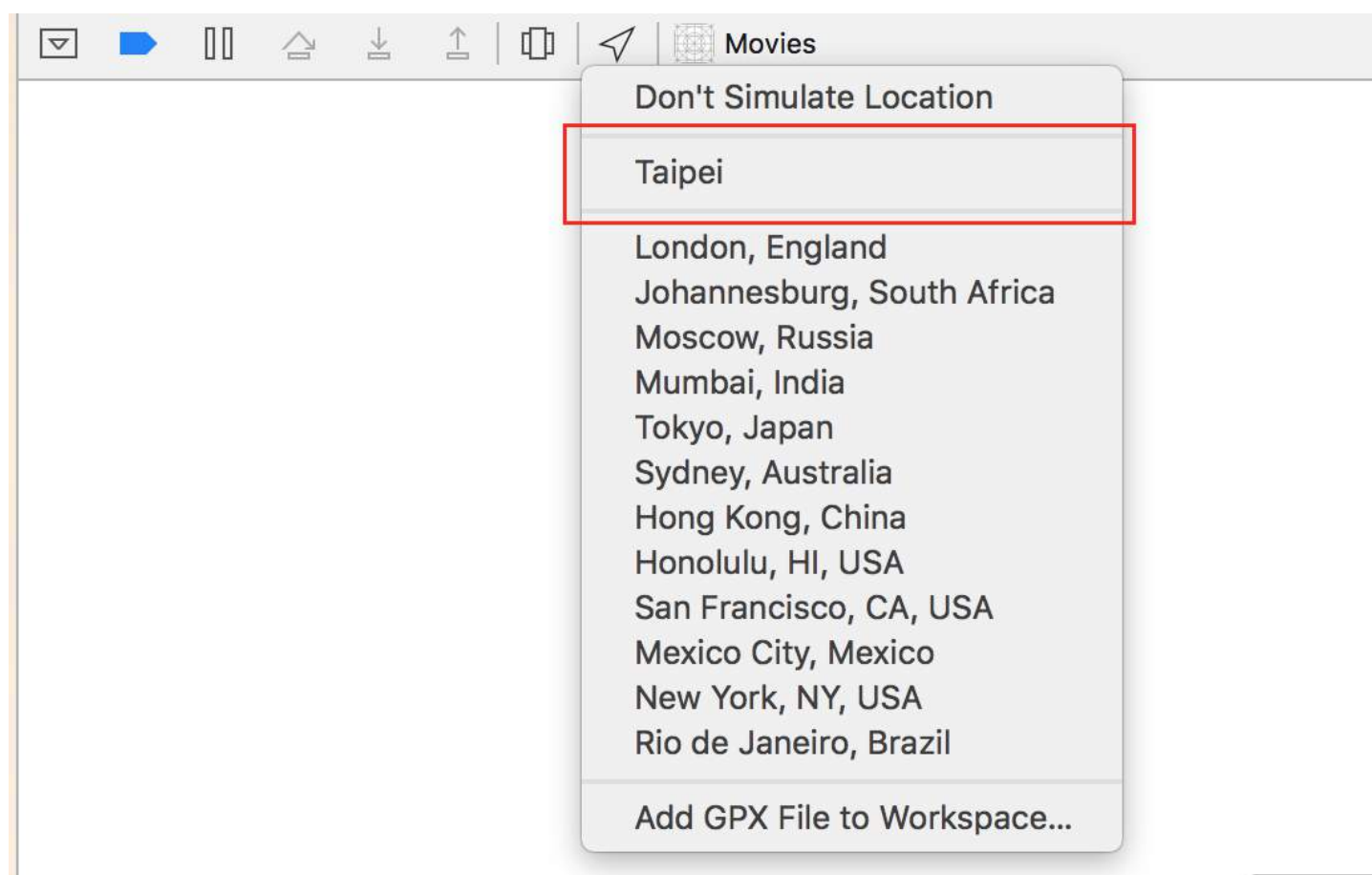


You can repeat this process to create multiple locations.

# Section 68.3: Set Zoom/Region for Map

For setting some zoom level, let say we want to zoom user's location with user location as center and 2km of area as radius. Then, we use following code

```
MKUserLocation *userLocation = _mapView.userLocation;
MKCoordinateRegion region = MKCoordinateRegionMakeWithDistance (userLocation.location.coordinate,
2000, 2000);
[_mapView setRegion:region animated:NO];
```

# Section 68.4: Local search implementation using MKLocalSearch

MKLocalSearch allows users to search for location using natural language strings like "gym". Once the search get completed, the class returns a list of locations within a specified region that match the search string.

Search results are in form of MKMapItem within MKLocalSearchResponse object.

lets try by example

```
MKLocalSearchRequest *request =
    [[MKLocalSearchRequest alloc] init];//initialising search request
request.naturalLanguageQuery = @"Gym"; // adding query
request.region = _mapView.region; //setting region
MKLocalSearch *search =
    [[MKLocalSearch alloc]initWithRequest:request];//initiate search

[search startWithCompletionHandler:^(MKLocalSearchResponse
    *response, NSError *error)
{
   if (response.mapItems.count == 0)
      NSLog(@"No Matches");
   else
      for (MKMapItem *item in response.mapItems)
      {
         NSLog(@"name = %@", item.name);
         NSLog(@"Phone = %@", item.phoneNumber);
       }
}];
```

# Section 68.5: OpenStreetMap Tile-Overlay

In some cases, you might not want to use the default maps, Apple provides.

You can add an overlay to your `mapView` that contains custom tiles for example from *OpenStreetMap*.

Let's assume, `self.mapView` is your `MKMapView` that you have already added to your `ViewController`.

At first, your `ViewController` needs to conform to the protocol `MKMapViewDelegate`.

```
class MyViewController: UIViewController, MKMapViewDelegate
```

Then you have to set the `ViewController` as delegate of `mapView`

```
mapView.delegate = self
```

Next, you configure the overlay for the map. You'll need an URL-template for this. The URL should be similar to this on all tile-servers and even if you would store the map-data offline:
`http://tile.openstreetmap.org/{z}/{x}/{y}.png`

```
let urlTeplate = "http://tile.openstreetmap.org/{z}/{x}/{y}.png"
let overlay = MKTileOverlay(urlTemplate: urlTeplate)
overlay.canReplaceMapContent = true
```

After you configured the overlay, you must add it to your `mapView`.

```
mapView.add(overlay, level: .aboveLabels)
```

To use custom maps, it is recommended to use `.aboveLabels` for `level`. Otherwise, the default labels would be visible on your custom map. If you want to see the default labels, you can choose `.aboveRoads` here.

If you would run your project now, you would recognize, that your map would still show the default map:
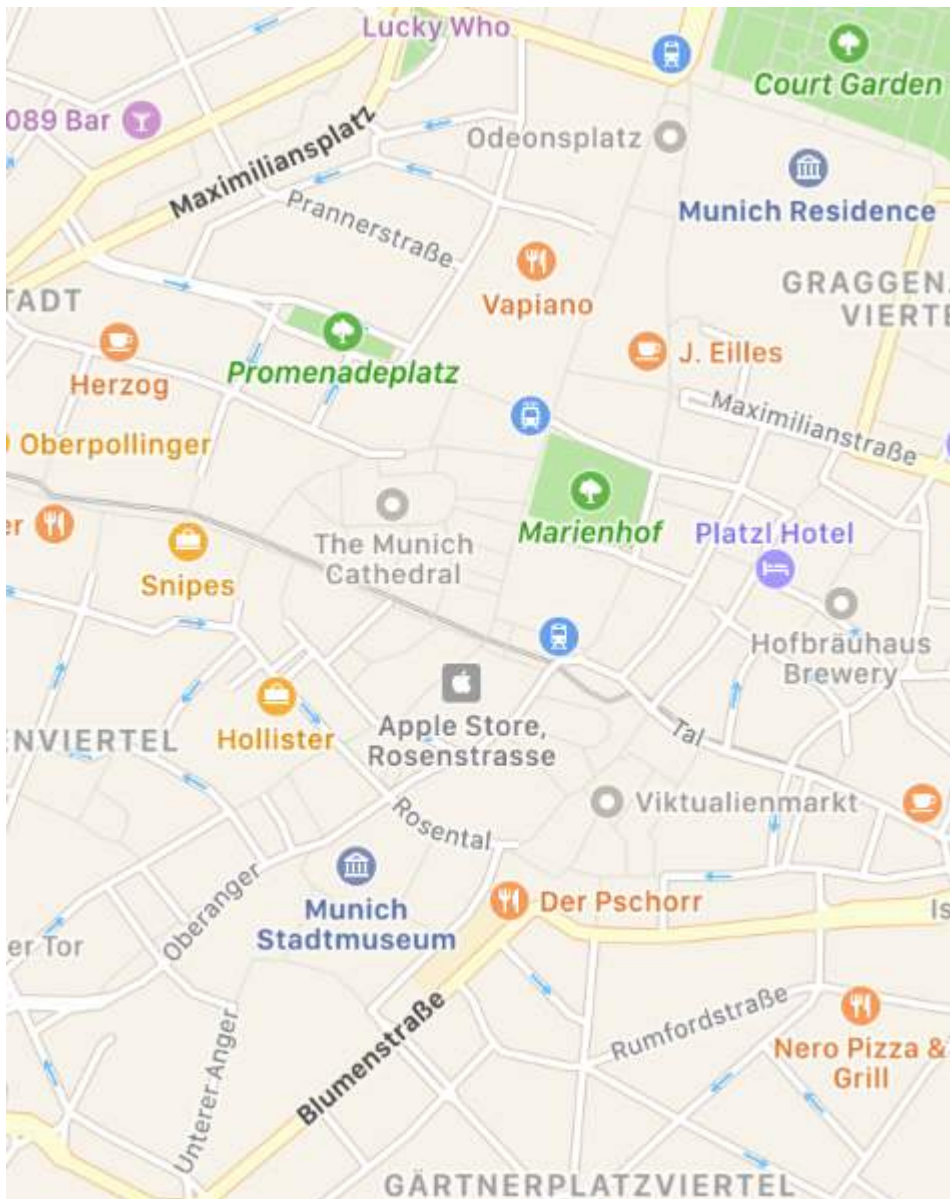


That's because we haven't told the `mapView` yet, how to render the overlay. This is the reason, why you had to set the delegate before. Now you can add `func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer` to your view controller:

```
func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
    if overlay is MKTileOverlay {
        let renderer = MKTileOverlayRenderer(overlay: overlay)
        return renderer
    } else {
        return MKTileOverlayRenderer()
    }
}
```

This will return the correct `MKOverlayRenderer` to your `mapView`. If you run your project now, you should see a map like this:

If you want to display another map, you just have to change the URL-template. There is a [list of tile-servers](#) in the OSM Wiki.

## Section 68.6: Scroll to coordinate and zoom-level

When you show a location to your users, you might want the `MKMapView` to display a coordinate at a zoom-level instead of setting a region to show. This functionality is not implemented by default, so you need to extend `MKMapView` with a methods that do the complex calculation from a *coordinate* and *zoom-level* to a `MKCoordinateRegion`.

```
let MERCATOR_OFFSET = 268435456.0
let MERCATOR_RADIUS = 85445659.44705395
let DEGREES = 180.0

public extension MKMapView {

    //MARK: Map Conversion Methods

    private func longitudeToPixelSpaceX(longitude:Double)->Double{
        return round(MERCATOR_OFFSET + MERCATOR_RADIUS * longitude * M_PI / DEGREES)
    }

    private func latitudeToPixelSpaceY(latitude:Double)->Double{
        return round(MERCATOR_OFFSET - MERCATOR_RADIUS * log((1 + sin(latitude * M_PI / DEGREES)) /
(1 - sin(latitude * M_PI / DEGREES))) / 2.0)
```

```
    }

    private func pixelSpaceXToLongitude(pixelX:Double)->Double{
        return ((round(pixelX) - MERCATOR_OFFSET) / MERCATOR_RADIUS) * DEGREES / M_PI
    }

    private func pixelSpaceYToLatitude(pixelY:Double)->Double{
        return (M_PI / 2.0 - 2.0 * atan(exp((round(pixelY) - MERCATOR_OFFSET) / MERCATOR_RADIUS)))
* DEGREES / M_PI
    }

    private func coordinateSpanWithCenterCoordinate(centerCoordinate:CLLocationCoordinate2D,
zoomLevel:Double)->MKCoordinateSpan{
        // convert center coordiate to pixel space
        let centerPixelX = longitudeToPixelSpaceX(longitude: centerCoordinate.longitude)
        let centerPixelY = latitudeToPixelSpaceY(latitude: centerCoordinate.latitude)
        print(centerCoordinate)
        // determine the scale value from the zoom level
        let zoomExponent:Double = 20.0 - zoomLevel
        let zoomScale:Double = pow(2.0, zoomExponent)
        // scale the map's size in pixel space
        let mapSizeInPixels = self.bounds.size
        let scaledMapWidth = Double(mapSizeInPixels.width) * zoomScale
        let scaledMapHeight = Double(mapSizeInPixels.height) * zoomScale
        // figure out the position of the top-left pixel
        let topLeftPixelX = centerPixelX - (scaledMapWidth / 2.0)
        let topLeftPixelY = centerPixelY - (scaledMapHeight / 2.0)
        // find delta between left and right longitudes
        let minLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX)
        let maxLng = pixelSpaceXToLongitude(pixelX: topLeftPixelX + scaledMapWidth)
        let longitudeDelta = maxLng - minLng
        let minLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY)
        let maxLat = pixelSpaceYToLatitude(pixelY: topLeftPixelY + scaledMapHeight)
        let latitudeDelta = -1.0 * (maxLat - minLat)
        return MKCoordinateSpan(latitudeDelta: latitudeDelta, longitudeDelta: longitudeDelta)
    }


    /**
     Sets the center of the `MKMapView` to a `CLLocationCoordinate2D` with a custom zoom-level.
There is no nee to set a region manually. :-)

     - author: Mylene Bayan (on GitHub)
     */
    public func setCenter(_ coordinate:CLLocationCoordinate2D, zoomLevel:Double, animated:Bool){
        // clamp large numbers to 28
        var zoomLevel = zoomLevel
        zoomLevel = min(zoomLevel, 28)
        // use the zoom level to compute the region
        print(coordinate)
        let span = self.coordinateSpanWithCenterCoordinate(centerCoordinate: coordinate, zoomLevel:
zoomLevel)
        let region = MKCoordinateRegionMake(coordinate, span)
        if region.center.longitude == -180.00000000{
            print("Invalid Region")
        }
        else{
            self.setRegion(region, animated: animated)
        }
    }
}
```

(The original Swift 2 version by Mylene Bayan can be found on GitHub)

After you implemented this `extension`, you can set the center coordinate as following:

```
let centerCoordinate = CLLocationCoordinate2DMake(48.136315, 11.5752901) //latitude, longitude
mapView?.setCenter(centerCoordinate, zoomLevel: 15, animated: true)
```

zoomLevel is a `Double` value, usually between 0 and 21 (which is a very high zoom-level), but values up to 28 are allowed.

## Section 68.7: Working With Annotation

**Get All Annotation**

```
//following method returns all annotations object added on map
NSArray *allAnnotations = mapView.annotations;
```

**Get Annotation View**

```
for (id<MKAnnotation> annotation in mapView.annotations)
{
    MKAnnotationView* annotationView = [mapView viewForAnnotation:annotation];
    if (annotationView)
    {
        // Do something with annotation view
        // for e.g change image of annotation view
        annotationView.image = [UIImage imageNamed:@"SelectedPin.png"];
    }
}
```

**Remove All Annotations**

```
[mapView removeAnnotations:mapView.annotations]
```

**Remove Single Annotation**

```
//getting all Annotation
NSArray *allAnnotations = self.myMapView.annotations;

if (allAnnotations.count > 0)
{
    //getting first annoation
    id <MKAnnotation> annotation=[allAnnotations firstObject];

    //removing annotation
    [mapView removeAnnotation:annotation];

}
```

## Section 68.8: Add MKMapView

**Swift**

```
let mapView = MKMapView(frame: CGRect(x: 0, y: 0, width: 320, height: 500))
```

It's recommended to store the mapView as a property of the containing `ViewController` since you might want to access it in more complex implementations.

**Objective C**

```
self.map = [[MKMapView alloc]initWithFrame:CGRectMake(0, 0, self.view.frame.size.width,
self.view.frame.size.height)];
[self.view addSubview:self.map];
```

# Section 68.9: Show UserLocation and UserTracking example

This will show the user location on the map

**Objective-C**

```
[self.map setShowsUserLocation:YES];
```

**Swift**

```
self.map?.showsUserLocation = true
```
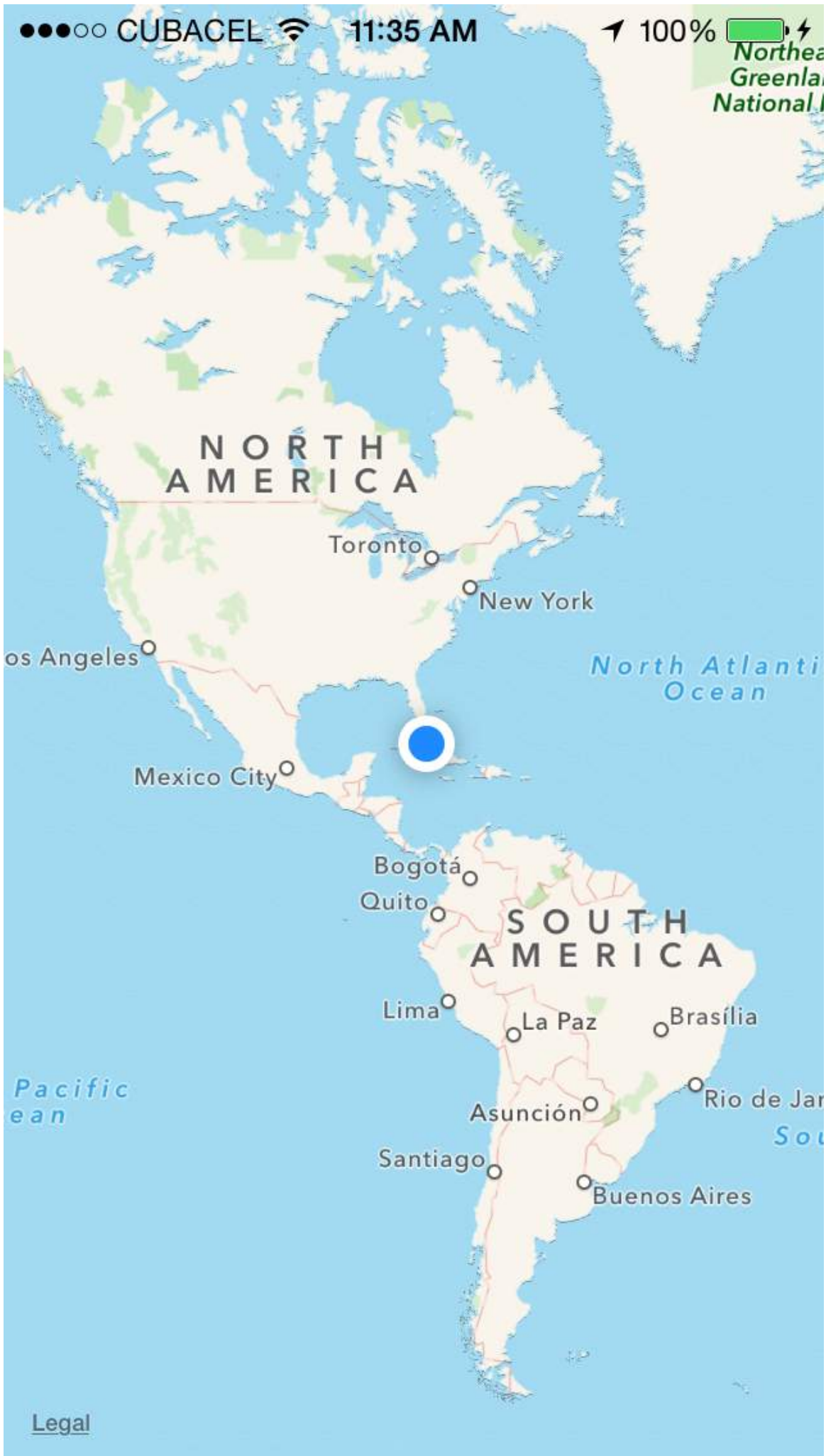
This will track the user location on the map, updating regions according

**Objective-C**

```
[self.map setUserTrackingMode:MKUserTrackingModeFollow];
```

**Swift**

```
self.map?.userTrackingMode = .follow
```

# Section 68.10: Adding Pin/Point Annotation on map

For annotating some point of interest on map, we use pin annotation. Now, start by creating annotation object first.

```
MKPointAnnotation *pointAnnotation = [[MKPointAnnotation alloc] init];
```

Now provide coordinate to pointAnnotation,as

```
CLLocationCoordinate2D coordinate = CLLocationCoordinate2DMake(23.054625,72.534562);
pointAnnotation.coordinate = coordinate;
```

Now, provide title and subtitle to annotation,

```
pointAnnotation.title = @"XYZ Point";
pointAnnotation.subtitle = @"Ahmedabad Area";
```

Now, add this annotation to map.

```
[self.mapView addAnnotation:pointAnnotation];
```

Yeaah.. Hurrah.. you have done the job. You can now see point annotation(red coloured pin) at given coordinate.

But now, what if you want to change color of the pin(3 available colors are - Purple,red and green). Then follow this step.

set mapview's delegate to self,

```
self.mapView.delegate = self;
```

Add MKMapViewDelegate implementation. Now add following method then,

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id <MKAnnotation>)annotation
{
    // If it's the user location, just return nil, because it have user location's own annotation,
if you want to change that, then use this object;
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;

    if ([annotation isKindOfClass:[MKPointAnnotation class]])
    {
        //Use dequed pin if available
        MKAnnotationView *pinView = [mapView
dequeueReusableAnnotationViewWithIdentifier:@"PinAnnotationView"];

        if (!pinView)
        {
            // If not dequed, then create new.
            pinView = [[MKAnnotationView alloc] initWithAnnotation:annotation
reuseIdentifier:@"PinAnnotationView"];
```

```
            pinView.canShowCallout = YES;
            pinView.image = [UIImage imageNamed:@"abc.png"];
            pinView.calloutOffset = CGPointMake(0, 32);
        } else {
            pinView.annotation = annotation;
        }
        return pinView;
    }
    return nil;
}
```

# Section 68.11: Adjust the map view's visible rect in order to display all annotations

Swift:

```
mapView.showAnnotations(mapView.annotations, animated: true)
```

Objective-C:

```
[mapView showAnnotations:mapView.annotations animated:YES];
```

Demo:
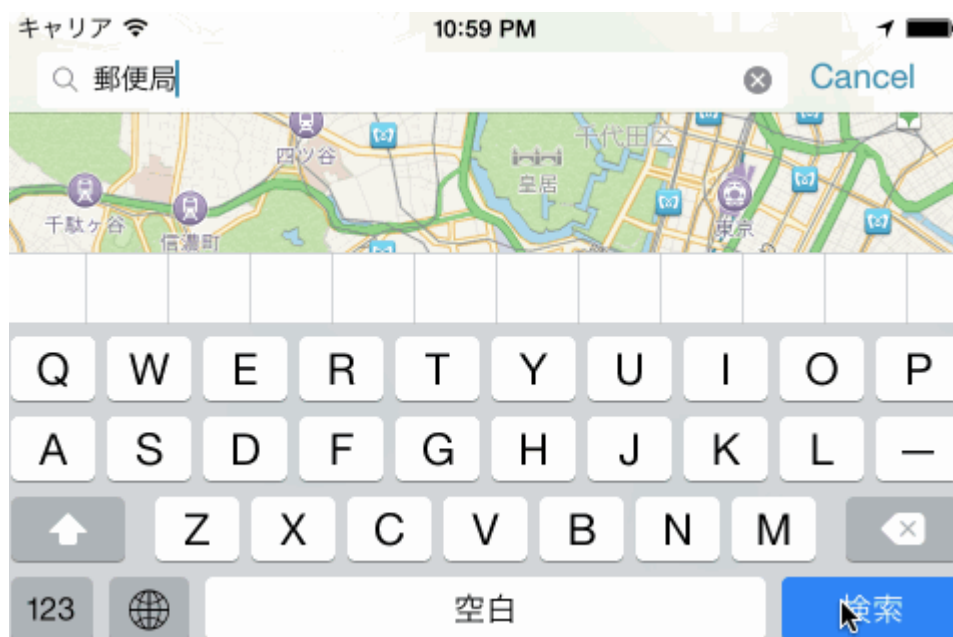
# Chapter 69: NSArray

Here are some useful utility functions/methods that can be used as with Array extension for ease of developer to perform certain critical operations on array with help of single line code.

## Section 69.1: Convert Array into json string

Call this function with parameter argument as array with type 'any'. It will return you json string. Json string is used to submit array in web service call as request input parameter in Swift.

```
//-----------------------
```

```swift
let array = [["one" : 1], ["two" : 2], ["three" : 3], ["four" : 4]]

let jsonString = convertIntoJSONString(arrayObject: array)
print("jsonString - \(jsonString)")
```

```
//-----------------------
```

```swift
func convertIntoJSONString(arrayObject: [Any]) -> String? {

        do {
            let jsonData: Data = try JSONSerialization.data(withJSONObject: arrayObject, options:
[])
            if  let jsonString = NSString(data: jsonData, encoding: String.Encoding.utf8.rawValue)
{
                return jsonString as String
            }

        } catch let error as NSError {
            print("Array convertIntoJSON - \(error.description)")
        }
        return nil
    }
```

# Chapter 70: NSAttributedString

## Section 70.1: Creating a string that has custom kerning (letter spacing)

`NSAttributedString` (and its mutable sibling `NSMutableAttributedString`) allows you to create strings that are complex in their appearance to the user.

A common application is to use this to display a string and adding custom kerning / letter-spacing.

This would be achieved as follows (where label is a `UILabel`), giving a different kerning for the word "kerning"

**Swift**

```swift
var attributedString = NSMutableAttributedString("Apply kerning")
attributedString.addAttribute(attribute: NSKernAttributeName, value: 5, range: NSMakeRange(6, 7))
label.attributedText = attributedString
```

**Objective-C**

```objc
NSMutableAttributedString *attributedString;
attributedString = [[NSMutableAttributedString alloc] initWithString:@"Apply kerning"];
[attributedString addAttribute:NSKernAttributeName value:@5 range:NSMakeRange(6, 7)];
[label setAttributedText:attributedString];
```

## Section 70.2: Change the color of a word or string

**Objective-C**

```objc
UIColor *color = [UIColor redColor];
NSString *textToFind = @"redword";

NSMutableAttributedString *attrsString =  [[NSMutableAttributedString alloc]
initWithAttributedString:yourLabel.attributedText];

// search for word occurrence
NSRange range = [yourLabel.text rangeOfString:textToFind];
if (range.location != NSNotFound) {
    [attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
}

// set attributed text
yourLabel.attributedText = attrsString;
```

**Swift**

```swift
let color = UIColor.red;
let textToFind = "redword"

let attrsString =  NSMutableAttributedString(string:yourlabel.text!);

// search for word occurrence
let range = (yourlabel.text! as NSString).range(of: textToFind)
if (range.length > 0) {
    attrsString.addAttribute(NSForegroundColorAttributeName,value:color,range:range)
}
```

```
// set attributed text
yourlabel.attributedText = attrsString
```

**Note**:

The main here is to use a `NSMutableAttributedString` and the selector `addAttribute:value:range` with the attribute `NSForegroundColorAttributeName` to change a color of a string range:

```
NSMutableAttributedString *attrsString = [[NSMutableAttributedString alloc]
initWithAttributedString:label.attributedText];
[attrsString addAttribute:NSForegroundColorAttributeName value:color range:range];
```

You could use another way to get the range, for example: NSRegularExpression.

# Section 70.3: Removing all attributes

**Objective-C**

```
NSMutableAttributedString *mutAttString = @"string goes here";
NSRange range = NSMakeRange(0, mutAttString.length);
[mutAttString setAttributes:@{} range:originalRange];
```

As per Apple Documentation we use, `setAttributes` and not `addAttribute`.

**Swift**

```
mutAttString.setAttributes([:], range: NSRange(0..<string.length))
```

# Section 70.4: Appending Attributed Strings and bold text in Swift

```
let someValue : String = "Something the user entered"
let text = NSMutableAttributedString(string: "The value is: ")
text.appendAttributedString(NSAttributedString(string: someValue, attributes:
[NSFontAttributeName:UIFont.boldSystemFontOfSize(UIFont.systemFontSize())]))
```

The result looks like:

The value is: **Something the user entered**

# Section 70.5: Create a string with strikethrough text

**Objective-C**

```
NSMutableAttributedString *attributeString = [[NSMutableAttributedString alloc]
initWithString:@"Your String here"];
[attributeString addAttribute:NSStrikethroughStyleAttributeName
                    value:@2
                    range:NSMakeRange(0, [attributeString length])];
```

**Swift**

```
let attributeString: NSMutableAttributedString = NSMutableAttributedString(string: "Your String
here")
```

```
attributeString.addAttribute(NSStrikethroughStyleAttributeName, value: 2, range: NSMakeRange(0,
attributeString.length))
```

Then you can add this to your UILabel:

```
yourLabel.attributedText = attributeString;
```

# Chapter 71: Convert HTML to NSAttributed string and vice verse

## Section 71.1: Objective C code to convert HTML string to NSAttributedString and Vice Versa

**HTML to NSAttributedString conversion Code:**

```
//HTML String
NSString *htmlString=[[NSString alloc]initWithFormat:@"<!DOCTYPE html><html><body><h1>My First
Heading</h1><p>My first paragraph.</p></body></html>"];
//Converting HTML string with UTF-8 encoding to NSAttributedString
NSAttributedString *attributedString = [[NSAttributedString alloc]
                                        initWithData: [htmlString
dataUsingEncoding:NSUnicodeStringEncoding]
                                        options: @{ NSDocumentTypeDocumentAttribute:
NSHTMLTextDocumentType }
                                        documentAttributes: nil
                                        error: nil ];
```

**NSAttributedString to HTML Conversion:**

```
//Dictionary to hold all the attributes of NSAttributed String
NSDictionary *documentAttributes = @{NSDocumentTypeDocumentAttribute: NSHTMLTextDocumentType};
//Saving the NSAttributedString with all its attributes as a NSData Entity
NSData *htmlData = [attributedString dataFromRange:NSMakeRange(0, attributedString.length)
documentAttributes:documentAttributes error:NULL];
//Convert the NSData into HTML String with UTF-8 Encoding
NSString *htmlString = [[NSString alloc] initWithData:htmlData encoding:NSUTF8StringEncoding];
```

# Chapter 72: NSTimer

| Parameter | Details |
|-----------|---------|
| `interval` | The time, in seconds, to wait beforing firing the timer; or, in repeating timers, the time between firings. |
| `target` | The object to call the `selector` on |
| `selector` | In Swift, a `Selector` object specifying the method to call on the `target` |
| `repeats` | If `false`, fire the timer only once. If `true`, fire the timer every `interval` seconds. |

## Section 72.1: Creating a Timer

This will create a timer to call the `doSomething` method on `self` in 5 seconds.

**Swift**

```swift
let timer = NSTimer.scheduledTimerWithTimeInterval(5,
                              target: self,
                            selector: Selector(doSomething()),
                            userInfo: nil,
                             repeats: false)
```

**Swift 3**

```swift
let timer = Timer.scheduledTimer(timeInterval: 1,
                                       target: self,
                                     selector: #selector(doSomething()),
                                     userInfo: nil,
                                      repeats: true)
```

**Objective-C**

```objc
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self
selector:@selector(doSomething) userInfo:nil repeats:NO];
```

Setting repeats to `false`/NO indicates that we want the timer to fire only once. If we set this to `true`/YES, it would fire every five seconds until manually invalidated.

## Section 72.2: Manually firing a timer

**Swift**

```swift
timer.fire()
```

**Objective-C**

```objc
[timer fire];
```

Calling the `fire` method causes an NSTimer to perform the task it would have usually performed on a schedule.

In a **non-repeating timer**, this will automatically invalidate the timer. That is, calling `fire` before the time interval is up will result in only one invocation.

In a **repeating timer**, this will simply invoke the action without interrupting the usual schedule.

# Section 72.3: Timer frequency options

**Repeated Timer event**

**Swift**

```
class ViewController: UIViewController {

    var timer = NSTimer()

    override func viewDidLoad() {
        NSTimer.scheduledTimerWithTimeInterval(1.0, target: self, selector:
Selector(self.timerMethod()), userInfo: nil, repeats: true)
    }

    func timerMethod() {
        print("Timer method called")
    }

    func endTimer() {
        timer.invalidate()
    }
}
```

**Swift 3**

```
class ViewController: UIViewController {

    var timer = Timer()

    override func viewDidLoad() {
        Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:
#selector(self.timerMethod()), userInfo: nil, repeats: true)
    }

    func timerMethod() {
        print("Timer method called")
    }

    func endTimer() {
        timer.invalidate()
    }
}
```

Must be invalidated manually if desired.

**Swift**

**Non-repeated delayed Timer event**

```
NSTimer.scheduledTimerWithTimeInterval(3.0, target: self, selector: Selector(self.timerMethod()),
userInfo: nil, repeats: false)
```

**Swift 3**

```
 Timer.scheduledTimer(timeInterval: 3.0, target: self, selector: #selector(self.timerMethod()),
userInfo: nil, repeats: false)
```

Timer will be fired once, 3 seconds after time of execution. Will be invalidated automatically, once fired.

# Section 72.4: Invalidating a timer

**Swift**

```
timer.invalidate()
```

**Objective-C**

```
[timer invalidate];
```

This will stop the timer from firing. **Must be called from the thread the timer was created in,** see [Apple's notes](#):

> You must send this message from the thread on which the timer was installed. If you send this message from another thread, the input source associated with the timer may not be removed from its run loop, which could prevent the thread from exiting properly.

**Notes: Once timer has been invalidated, its impossible to fire same invalidated timer.Instead you need to initialise the invalidated timer again and trigger fire method.**

# Section 72.5: Passing of data using Timer

If you you want to pass some data with the timer trigger you can do it with the `userInfo`parameter.

Here is the simple approach that gives brief idea about how you can pass the data to triggered method from the Timer.

[*Swift 3*]

```
Timer.scheduledTimer(timeInterval: 1.0, target: self, selector:#selector(iGotCall(sender:)),
userInfo: ["Name": "i am iOS guy"], repeats:true)
```

[*Objective - C*]

```
NSTimer* timer = [NSTimer scheduledTimerWithTimeInterval:1.0
                                        target:self
                                        selector:@selector(iGotCall:)
                                        userInfo:@"i am iOS guy" repeats:YES];
```

The above line of code passing `["Name": "i am iOS guy"]` into the `userInfo`. So now when the `iGotCall` get call you can get the passed value as below code snippet.

[*Swift 3*]

```
func iGotCall(sender: Timer) {
        print((sender.userInfo)!)
    }
```

[*Objective - C*]

```
- (void)iGotCall:(NSTimer*)theTimer {
    NSLog (@"%@", (NSString*)[theTimer userInfo]);
}
```

# Chapter 73: NSDate

## Section 73.1: NSDateFormatter

Converting an `NSDate` object to string is just 3 steps.

**1. Create an `NSDateFormatter` object**
**Swift**

```
let dateFormatter = NSDateFormatter()
```

**Swift 3**

```
let dateFormatter = DateFormatter()
```

**Objective-C**

```
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
```

**2. Set the date format in which you want your string**
**Swift**

```
dateFormatter.dateFormat = "yyyy-MM-dd 'at' HH:mm"
```

**Objective-C**

```
dateFormatter.dateFormat = @"yyyy-MM-dd 'at' HH:mm";
```

**3. Get the formatted string**
**Swift**

```
let date = NSDate() // your NSDate object
let dateString = dateFormatter.stringFromDate(date)
```

**Swift 3**

```
let date = Date() // your NSDate object
let dateString = dateFormatter.stringFromDate(date)
```

**Objective-C**

```
NSDate *date = [NSDate date]; // your NSDate object
NSString *dateString = [dateFormatter stringFromDate:date];
```

This will give output something like this: `2001-01-02 at 13:00`

> **Note**
>
> Creating an `NSDateFormatter` instance is an expensive operation, so it is recommended to create it once and reuse when possible.

**Useful extension for converting date to string.**

```
extension Date {
    func toString() -> String {
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "MMMM dd yyyy"
        return dateFormatter.string(from: self)
    }
}
```

Useful links for swift date-formation [swiftly-getting-human-readable-date-nsdateformatter](swiftly-getting-human-readable-date-nsdateformatter).

For constructing date formats see [date format patterns](date format patterns).

# Section 73.2: Date Comparison

There are 4 methods for comparing dates:

**Swift**

- isEqualToDate(anotherDate: NSDate) -> Bool
- earlierDate(anotherDate: NSDate) -> NSDate
- laterDate(anotherDate: NSDate) -> NSDate
- compare(anotherDate: NSDate) -> NSComparisonResult

**Objective-C**

- - (BOOL)isEqualToDate:(NSDate *)anotherDate
- - (NSDate *)earlierDate:(NSDate *)anotherDate
- - (NSDate *)laterDate:(NSDate *)anotherDate
- - (NSComparisonResult)compare:(NSDate *)anotherDate

Let's say we have 2 dates:

**Swift**

```
let date1: NSDate = ... // initialized as  July 7, 2016 00:00:00
let date2: NSDate = ... // initialized as  July 2, 2016 00:00:00
```

**Objective-C**

```
NSDate *date1 = ... // initialized as  July 7, 2016 00:00:00
NSDate *date2 = ... // initialized as  July 2, 2016 00:00:00
```

Then, to compare them, we try this code:

**Swift**

```
if date1.isEqualToDate(date2) {
    // returns false, as both dates aren't equal
}

earlierDate: NSDate = date1.earlierDate(date2) // returns the earlier date of the two (date 2)
laterDate: NSDate = date1.laterDate(date2) // returns the later date of the two (date1)

result: NSComparisonResult = date1.compare(date2)

if result == .OrderedAscending {
    // true if date1 is earlier than date2
} else if result == .OrderedSame {
    // true if the dates are the same
} else if result == .OrderedDescending {
    // true if date1 is later than date1
}
```

**Objective-C**

```
if ([date1 isEqualToDate:date2]) {
    // returns false, as both date are not equal
}

NSDate *earlierDate = [date1 earlierDate:date2]; // returns date which comes earlier from both
date, here it will return date2
NSDate *laterDate = [date1 laterDate:date2]; // returns date which comes later from both date, here
it will return date1

NSComparisonResult result = [date1 compare:date2];
```

```
if (result == NSOrderedAscending) {
    // fails
    // comes here if date1 is earlier then date2, in our case it will not come here
} else if (result == NSOrderedSame){
    // fails
    // comes here if date1 is same as date2, in our case it will not come here
} else{ // NSOrderedDescending
    // succeeds
    // comes here if date1 is later than date2, in our case it will come here
}
```

If you want to compare dates and handle seconds, weeks, months and years:

**Swift 3**

```
let dateStringUTC = "2016-10-22 12:37:48 +0000"
let dateFormatter = DateFormatter()
dateFormatter.locale = Locale(identifier: "en_US_POSIX")
dateFormatter.dateFormat = "yyyy-MM-dd HH:mm:ss X"
let date = dateFormatter.date(from: dateStringUTC)!

let now = Date()

let formatter = DateComponentsFormatter()
formatter.unitsStyle = .full
formatter.maximumUnitCount = 2
let string = formatter.string(from: date, to: Date())! + " " + NSLocalizedString("ago", comment:
"added after elapsed time to say how long before")
```

Or you can use this for each component:

```
// get the current date and time
let currentDateTime = Date()

// get the user's calendar
let userCalendar = Calendar.current

// choose which date and time components are needed
let requestedComponents: Set<Calendar.Component> = [
    .year,
    .month,
    .day,
    .hour,
    .minute,
    .second
]

// get the components
let dateTimeComponents = userCalendar.dateComponents(requestedComponents, from: currentDateTime)

// now the components are available
dateTimeComponents.year
dateTimeComponents.month
dateTimeComponents.day
dateTimeComponents.hour
dateTimeComponents.minute
dateTimeComponents.second
```

# Section 73.3: Get Historic Time from NSDate (eg: 5s ago, 2m ago, 3h ago)

This can be used in various chat applications, rss feeds, and social apps where you need to have latest feeds with timestamps:

**Objective-C**

```objc
- (NSString *)getHistoricTimeText:(NSDate *)since
{
    NSString *str;
    NSTimeInterval interval = [[NSDate date] timeIntervalSinceDate:since];
    if(interval < 60)
        str = [NSString stringWithFormat:@"%is ago",(int)interval];
    else if(interval < 3600)
    {
        int minutes = interval/60;
        str = [NSString stringWithFormat:@"%im ago",minutes];
    }
    else if(interval < 86400)
    {
        int hours =  interval/3600;

        str = [NSString stringWithFormat:@"%ih ago",hours];
    }
    else
    {
        NSDateFormatter *dateFormater=[[NSDateFormatter alloc]init];
        [dateFormater setLocale:[NSLocale currentLocale]];
        NSString *dateFormat = [NSDateFormatter dateFormatFromTemplate:@"MMM d, YYYY" options:0
locale:[NSLocale currentLocale]];
        [dateFormater setDateFormat:dateFormat];
        str = [dateFormater stringFromDate:since];

    }
    return str;
}
```

# Section 73.4: Get Unix Epoch time

To get Unix Epoch Time, use the constant `timeIntervalSince1970`:

**Swift**

```swift
let date = NSDate() // current date
let unixtime = date.timeIntervalSince1970
```

**Objective-C**

```objc
NSDate *date = [NSDate date]; // current date
int unixtime = [date timeIntervalSince1970];
```

# Section 73.5: Get NSDate from JSON Date format "/Date(126812328143)/"

Prior to Json.NET 4.5 dates were written using the Microsoft format: "/Date(1198908717056)/". If your server sends date in this format you can use the below code to serialize it to NSDate:

**Objective-C**

```objc
(NSDate*) getDateFromJSON:(NSString *)dateString
```

```
{
    // Expect date in this format "/Date(1268123281843)/"
    int startPos = [dateString rangeOfString:@"("].location+1;
    int endPos = [dateString rangeOfString:@")"].location;
    NSRange range = NSMakeRange(startPos,endPos-startPos);
    unsigned long long milliseconds = [[dateString substringWithRange:range] longLongValue];
    NSLog(@"%llu",milliseconds);
    NSTimeInterval interval = milliseconds/1000;
    NSDate *date = [NSDate dateWithTimeIntervalSince1970:interval];
    // add code for date formatter if need NSDate in specific format.
    return date;
}
```

# Section 73.6: Get time cycle type (12-hour or 24-hour)

**Checking whether the current date contains the symbol for AM or PM**
**Objective-C**
```
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
[formatter setLocale:[NSLocale currentLocale]];
[formatter setDateStyle:NSDateFormatterNoStyle];
[formatter setTimeStyle:NSDateFormatterShortStyle];
NSString *dateString = [formatter stringFromDate:[NSDate date]];
NSRange amRange = [dateString rangeOfString:[formatter AMSymbol]];
NSRange pmRange = [dateString rangeOfString:[formatter PMSymbol]];
BOOL is24h = (amRange.location == NSNotFound && pmRange.location == NSNotFound);
```
**Requesting the time cycle type from `NSDateFormatter`**
**Objective-C**
```
NSString *formatStringForHours = [NSDateFormatter dateFormatFromTemplate:@"j" options:0
locale:[NSLocale currentLocale]];
NSRange containsA = [formatStringForHours rangeOfString:@"a"];
BOOL is24h = containsA.location == NSNotFound;
```

This uses a special date template string called "j" which according to the [ICU Spec](#) ...

> [...] requests the preferred hour format for the locale (h, H, K, or k), as determined by the preferred
> attribute of the hours element in supplemental data. [...] Note that use of 'j' in a skeleton passed to an API
> is the only way to have a skeleton request a locale's preferred time cycle type (12-hour or 24-hour).

That last sentence is important. It "is the only way to have a skeleton request a locale's preferred time cycle type".
Since `NSDateFormatter` and `NSCalendar` are built on the ICU library, the same holds true here.

> **Reference**
>
> The second option was derived from [this answer](#).

# Section 73.7: Get Current Date

Getting current date is very easy. You get NSDate object of current date in just single line as follows:

**Swift**
```
var date = NSDate()
```
**Swift 3**
```
var date = Date()
```

**Objective-C**

```objc
NSDate *date = [NSDate date];
```

# Section 73.8: Get NSDate Object N seconds from current date

The number of seconds from the current date and time for the new date. Use a negative value to specify a date before the current date.

For doing this we have a method named `dateWithTimerIntervalSinceNow(seconds: NSTimeInterval) -> NSDate` (Swift) or `+ (NSDate*)dateWithTimeIntervalSinceNow:(NSTimeInterval)seconds` (Objective-C).

Now for example, if you require a date one week from current date and one week to current date, then we can do it as.

**Swift**

```swift
let totalSecondsInWeek:NSTimeInterval = 7 * 24 * 60 * 60;
//Using negative value for previous date from today
let nextWeek = NSDate().dateWithTimerIntervalSinceNow(totalSecondsInWeek)

//Using positive value for future date from today
let lastWeek = NSDate().dateWithTimerIntervalSinceNow(-totalSecondsInWeek)
```

**Swift 3**

```swift
let totalSecondsInWeek:TimeInterval = 7 * 24 * 60 * 60;

//Using positive value to add to the current date
let nextWeek = Date(timeIntervalSinceNow: totalSecondsInWeek)

//Using negative value to get date one week from current date
let lastWeek = Date(timeIntervalSinceNow: -totalSecondsInWeek)
```

**Objective-C**

```objc
NSTimeInterval totalSecondsInWeek = 7 * 24 * 60 * 60;
//Using negative value for previous date from today
NSDate *lastWeek = [NSDate dateWithTimeIntervalSinceNow:-totalSecondsInWeek];

//Using positive value for future date from today
NSDate *nextWeek = [NSDate dateWithTimeIntervalSinceNow:totalSecondsInWeek];

NSLog(@"Last Week: %@", lastWeek);
NSLog(@"Right Now: %@", now);
NSLog(@"Next Week: %@", nextWeek);
```

# Section 73.9: UTC Time offset from NSDate with TimeZone

Here this will calculate the `UTC` time offset from current data in desired timezone.

```objc
+(NSTimeInterval)getUTCOffSetIntervalWithCurrentTimeZone:(NSTimeZone *)current forDate:(NSDate
*)date  {
    NSTimeZone *utcTimeZone = [NSTimeZone timeZoneWithAbbreviation:@"UTC"];
    NSInteger currentGMTOffset = [current secondsFromGMTForDate:date];
    NSInteger gmtOffset = [utcTimeZone secondsFromGMTForDate:date];
    NSTimeInterval gmtInterval = currentGMTOffset - gmtOffset;
    return gmtInterval;
}
```

# Section 73.10: Convert NSDate that is composed from hour and minute (only) to a full NSDate

There are many cases when one has created an NSDate from only an hour and minute format, i.e: 08:12 that returns from a server as a String and you initiate an NSDate instance by these **values only.**

The downside for this situation is that your NSDate is almost completely "naked" and what you need to do is to create: day, month, year, second and time zone in order to this object to "play along" with other NSDate types.

For the sake of the example let's say that hourAndMinute is the NSDate type that is composed from hour and minute format:

**Objective-C**

```objc
NSDateComponents *hourAndMinuteComponents = [calendar components:NSCalendarUnitHour |
NSCalendarUnitMinute
                                              fromDate:hourAndMinute];
NSDateComponents *componentsOfDate = [[NSCalendar currentCalendar] components:NSCalendarUnitDay |
NSCalendarUnitMonth | NSCalendarUnitYear
                                              fromDate:[NSDate date]];

NSDateComponents *components = [[NSDateComponents alloc] init];
[components setDay: componentsOfDate.day];
[components setMonth: componentsOfDate.month];
[components setYear: componentsOfDate.year];
[components setHour: [hourAndMinuteComponents hour]];
[components setMinute: [hourAndMinuteComponents minute]];
[components setSecond: 0];
[calendar setTimeZone: [NSTimeZone defaultTimeZone]];

NSDate *yourFullNSDateObject = [calendar dateFromComponents:components];
```

Now your object is the total opposite of being "naked".

# Chapter 74: NSNotificationCenter

| Parameter | Details |
|-----------|---------|
| name | The name of the notification for which to register the observer; that is, only notifications with this name are used to add the block to the operation queue. If you pass nil, the notification center doesn't use a notification's name to decide whether to add the block to the operation queue. |
| obj | The object whose notifications the observer wants to receive; that is, only notifications sent by this sender are delivered to the observer. If you pass nil, the notification center doesn't use a notification's sender to decide whether to deliver it to the observer. |
| queue | The operation queue to which block should be added. If you pass nil, the block is run synchronously on the posting thread. |
| block | The block to be executed when the notification is received. The block is copied by the notification center and (the copy) held until the observer registration is removed. |

iOS notifications are a simple and powerful way to send data in a loosely coupled way. That is, the sender of a notification doesn't have to care about who (if anyone) receives the notification, it just posts it out there to the rest of the app and it could be picked up by lots of things or nothing depending on your app's state.

**Source** : - [HACKING with Swift](HACKING with Swift)

## Section 74.1: Removing Observers

**Swift 2.3**

```
//Remove observer for single notification
NSNotificationCenter.defaultCenter().removeObserver(self, name: "TestNotification", object: nil)

//Remove observer for all notifications
NotificationCenter.defaultCenter().removeObserver(self)
```

**Swift 3**

```
//Remove observer for single notification
NotificationCenter.default.removeObserver(self, name: NSNotification.Name(rawValue:
"TestNotification"), object: nil)

//Remove observer for all notifications
NotificationCenter.default.removeObserver(self)
```

**Objective-C**

```
//Remove observer for single notification
[[NSNotificationCenter defaultCenter] removeObserver:self name:@"TestNotification" object:nil];

//Remove observer for all notifications
[[NSNotificationCenter defaultCenter] removeObserver:self];
```

## Section 74.2: Adding an Observer

**Naming Convention**

Notifications are identified by global NSString objects whose names are composed in this way:

```
Name of associated class + Did | Will + UniquePartOfName + Notification
```

For example:

- NSApplicationDidBecomeActiveNotification
- NSWindowDidMiniaturizeNotification
- NSTextViewDidChangeSelectionNotification

- NSColorPanelColorDidChangeNotification

**Swift 2.3**

```swift
NSNotificationCenter.defaultCenter().addObserver(self,
                                        selector: #selector(self.testNotification(_:)),
                                        name: "TestNotification",
                                        object: nil)
```

**Swift 3**

```swift
NSNotificationCenter.default.addObserver(self,
                                selector: #selector(self.testNotification(_:)),
                                name: NSNotification.Name(rawValue: "TestNotification"),
                                object: nil)
```

**Objective-C**

```objc
[[NSNotificationCenter defaultCenter] addObserver:self
                                selector:@selector(testNotification:)
                                name:@"TestNotification"
                                object:nil];
```

PS: It is also worth noting that the number of times an observer has been added has to be exactly the number of times the observer is removed. A rookie mistake is to add the observer in the `viewWillAppear:` of a UIViewController, but removing the observer in `viewDidUnload:`, will cause an uneven number of pushes and thus leaking the observer and the notification selector getting called in a superfluous manner.

# Section 74.3: Posting a Notification with Data

**Swift**

```swift
let userInfo: [String: AnyObject] = ["someKey": myObject]
NSNotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self,
userInfo: userInfo)
```

**Objective-C**

```objc
NSDictionary *userInfo = [NSDictionary dictionaryWithObject:myObject forKey:@"someKey"];
[[NSNotificationCenter defaultCenter] postNotificationName: @"TestNotification" object:nil
userInfo:userInfo];
```

# Section 74.4: Add and remove observer for name

```swift
// Add observer
let observer = NSNotificationCenter.defaultCenter().addObserverForName("nameOfTheNotification",
object: nil, queue: nil) { (notification) in
    // Do operations with the notification in this block
}

// Remove observer
NSNotificationCenter.defaultCenter().removeObserver(observer)
```

# Section 74.5: Posting a Notification

**Swift**

```swift
NSNotificationCenter.defaultCenter().postNotificationName("TestNotification", object: self)
```

**Objective-C**

```objc
[[NSNotificationCenter defaultCenter] postNotificationName:@"TestNotification" object:nil];
```

# Section 74.6: Observing a Notification

**Swift**

```swift
func testNotification(notification: NSNotification) {
    let userInfo = notification.userInfo
    let myObject: MyObject = userInfo["someKey"]
}
```

**Objective-C**

```objc
- (void)testNotification:(NSNotification *)notification {
    NSDictionary *userInfo = notification.userInfo;
    MyObject *myObject = [userInfo objectForKey:@"someKey"];
}
```

# Section 74.7: Adding/Removing an Observer with a Block

Instead of adding an observer with a selector, a block can be used:

```objc
id testObserver = [[NSNotificationCenter defaultCenter] addObserverForName:@"TestNotification"
                                                    object:nil
                                                     queue:nil
                                                usingBlock:^(NSNotification*
notification) {
    NSDictionary *userInfo = notification.userInfo;
    MyObject *myObject = [userInfo objectForKey:@"someKey"];
}];
```

The observer can then be removed with:

```objc
[[NSNotificationCenter defaultCenter] removeObserver:testObserver
                                     name:@"TestNotification"
                                   object:nil];
```

# Chapter 75: NSURLSession

## Section 75.1: Objective-C Create a Session And Data Task

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/"];
NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration defaultSessionConfiguration];

// Configure the session here.

NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration];

[[session dataTaskWithURL:url
        completionHandler:^(NSData *data, NSURLResponse *response, NSError *error)
{
    // The response object contains the metadata (HTTP headers, status code)

    // The data object contains the response body

    // The error object contains any client-side errors (e.g. connection
    // failures) and, in some cases, may report server-side errors.
    // In general, however, you should detect server-side errors by
    // checking the HTTP status code in the response object.
}] resume];
```

## Section 75.2: Setting up background configuration

To create a background session

```
// Swift:
let mySessionID = "com.example.bgSession"
let bgSessionConfig =
NSURLSessionConfiguration.backgroundSessionConfigurationWithIdentifier(mySessionID)

let session = NSURLSession(configuration: bgSessionConfig)

// add tasks here


// Objective-C:
NSString *mySessionID = @"com.example.bgSession";
NSURLSessionConfiguration *configuration =
    [NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier: mySessionID];
NSURLSession *session = [NSURLSession sessionWithConfiguration:configuration
                                                delegate:self]
```

Additionally, in iOS, you must set up support for handling background app relaunch. When your app's `application:handleEventsForBackgroundURLSession:completionHandler:` method (Objective-C) or `application(_:handleEventsForBackgroundURLSession:completionHandler:)` method (Swift) gets called, it means your app has been relaunched in the background to handle activity on a session.

In that method, you should create a new session with the provided identifier and configure it with a delegate to handle events just like you normally would in the foreground. Additionally, you should store the provided completion handler in a dictionary, using the session as the key.

When the delegate's `URLSessionDidFinishEventsForBackgroundURLSession:` (Obj-C) / `URLSessionDidFinishEventsForBackgroundURLSession` (Swift) method gets called to tell you that there are no more events to handle, your app should look up the completion handler for that session, remove the session from

the dictionary, and call the completion handler, thus telling the operating system that you no longer have any outstanding processing related to the session. (If you are still doing something for some reason when you get that delegate call, wait until done.) As soon as you call that method, the background session immediately gets invalidated.

If your application then receives an `application:application:didFinishLaunchingWithOptions:` call (likely indicating that the user foregrounded your app while you were busy processing background events), it is safe to create a background session with that same identifier, because the old session with that identifier no longer exists.

If you're curious about the details, at a high level, when you create a background session, you're doing two things:

- Creating a session in an external daemon (nsurlsessiond) to handle the downloads
- Creating a session within your app that talks to that external daemon via NSXPC

Normally, it is dangerous to create two sessions with the same session ID in a single launch of the app, because they both are trying to talk to the same session in the background daemon. This is why the official documentation says to never create multiple sessions with the same identifier. However, if the first session was a temporary session created as part of a `handleEventsForBackgroundURLSession` call, the association between the now-invalidated in-app session and the session in the background daemon no longer exists.

## Section 75.3: Simple GET request

```swift
// define url
let url = NSURL(string: "https://urlToGet.com")

//create a task to get data from a url
let task = NSURLSession.sharedSession().dataTaskWithURL(url!)
{

    /*inside this block, we have access to NSData *data, NSURLResponse *response, and NSError
*error returned by the dataTaskWithURL() function*/
    (data, response, error) in

    if error == nil
    {
        // Data from the request can be manipulated here
    }
    else
    {
        // An error occurred
    }
}

//make the request
task.resume()
```

## Section 75.4: Sending a POST Request with arguments using NSURLSession in Objective-C

There are two common ways to encode a POST request body: URL encoding (application/x-www-form-urlencoded) and form data (multipart/form-data). Much of the code is similar, but the way you construct the body data is different.

**Sending a request using URL encoding**

Be it you have a server for your small application or your working in a team with a full out back-end engineer, you'll

want to talk to that server at one point with your iOS application.

In the following code we will be composing a string of arguments that the destination server script will use to do something that changes depending on your case. For example we may want to send the string:

> name=Brendon&password=abcde

To the server when a user signs up to your application, so the server can store this information in a database.

Let's get started. You'll want to create a NSURLSession POST request with the following code.

```
// Create the configuration, which is necessary so we can cancel cacheing amongst other things.
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration
defaultSessionConfiguration];
// Disables cacheing
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration:defaultConfigObject
delegate:self delegateQueue:[NSOperationQueue mainQueue]];

NSString * scriptURL = [NSString stringWithFormat:@"https://server.io/api/script.php"];
//Converts the URL string to a URL usable by NSURLSession
NSMutableURLRequest * urlRequest = [NSMutableURLRequest requestWithURL:[NSURL
URLWithString:scriptURL]];
NSString * postDataString = [NSString stringWithFormat:@"name=%@&password=%@", [self nameString],
[self URLEncode:passwordString]];
[urlRequest setHTTPMethod:@"POST"];
[urlRequest setHTTPBody:[postDataString dataUsingEncoding:NSUTF8StringEncoding]];

NSURLSessionDataTask * dataTask = [defaultSession dataTaskWithRequest:urlRequest];
// Fire the data task.
[dataTask resume];
```

The above code just created and fired the POST request to the server. Remember that the script URL and the POST data string changes depending on your situation. If you're reading this, you'll know what to fill those variables with.

You'll also need to add a small method that does the URL encoding:

```
- (NSString *)URLEncode:(NSString *)originalString encoding:(NSStringEncoding)encoding
{
    return (__bridge_transfer NSString *)CFURLCreateStringByAddingPercentEscapes(
        kCFAllocatorDefault,
        (__bridge CFStringRef)originalString,
        NULL,
        CFSTR(":/?#[]@!$&'()*+,;="),
        CFStringConvertNSStringEncodingToEncoding(encoding));
}
```

So, when the server is finished processing this data it will send a return to your iOS app. So we need to process this return, but how?

We use event-driven programming and use NSURLSession's delegate methods. This means as the server sends back a response these methods will start triggering. The following 5 methods are the ones that'll be triggered throughout the ENTIRE request, each time one is made:

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
 completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler;
```

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error;

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge
*)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential
*))completionHandler;

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler;
```

Below you'll see the above methods used in context. Each of their purposes are pretty self-explanatory thanks to Apple, but I've commented their uses anyway:

```
// Response handling delegates
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
 completionHandler:(void (^)(NSURLSessionResponseDisposition disposition))completionHandler{
    // Handler allows us to receive and parse responses from the server
    completionHandler(NSURLSessionResponseAllow);
}

- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask
didReceiveData:(NSData *)data{

    // Parse the JSON that came in into an NSDictionary
    NSError * err = nil;
    NSDictionary * jsonDict = [NSJSONSerialization JSONObjectWithData:data
options:NSJSONReadingAllowFragments error:&err];

    if (!err){ // if no error occurred, parse the array of objects as normal
        // Parse the JSON dictionary 'jsonDict' here
    }else{ // an error occurred so we need to let the user know
        // Handle your error here
    }
}

// Error handling delegate
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error{
    if(error == nil){
        // Download from API was successful
        NSLog(@"Data Network Request Did Complete Successfully.");
    }else{
        // Describes and logs the error preventing us from receiving a response
        NSLog(@"Error: %@", [error userInfo]);

        // Handle network error, letting the user know what happened.
    }
}


// When the session receives a challenge (because of iOS 9 App Transport Security blocking non-
valid SSL certificates) we use the following methods to tell NSURLSession "Chill out, I can trust
me".
// The following is not necessary unless your server is using HTTP, not HTTPS

- (void)URLSession:(NSURLSession *)session didReceiveChallenge:(NSURLAuthenticationChallenge
*)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition, NSURLCredential
```

```
*))completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLSessionAuthChallengeUseCredential,credential);
        }
    }
}

- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void
(^)(NSURLSessionAuthChallengeDisposition, NSURLCredential * _Nullable))completionHandler{
    if([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]){
        if([challenge.protectionSpace.host isEqualToString:@"DomainNameOfServer.io"]){
            NSURLCredential * credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            completionHandler(NSURLSessionAuthChallengeUseCredential,credential);
        }
    }
}
```

So that's it! That's all the code you need to send, receive and parse a request for an API in iOS 9! Okay...it was kind of a lot of code. But if implemented right like above, it'll be fail-safe! Make sure to always handle errors where suggested above.

**Sending a request using form encoding**

URL encoding is a broadly compatible way to encode arbitrary data. However, it is relatively inefficient for uploading binary data (such as photos) because every non-ASCII byte turns into a three-character code. It also does not support file attachments, so you would have to pass filenames and file data as separate fields.

Suppose we want to upload a photograph in a way that is efficient and actually looks like a file on the server side. One way to do that is to use form encoding instead. To do this, edit the code that creates the NSURLSession as follows:

```
UIImage * imgToSend;

// 2nd parameter of UIImageJPEGRepresentation represents compression quality. 0 being most
compressed, 1 being the least
// Using 0.4 likely stops us hitting the servers upload limit and costs us less server space
NSData * imageData = UIImageJPEGRepresentation(imgToSend, 0.4f);

// Alternatively, if the photo is on disk, you can retrieve it with
// [NSData dataWithContentsOfURL:...]

// Set up the body of the POST request.

// This boundary serves as a separator between one form field and the next.
// It must not appear anywhere within the actual data that you intend to
// upload.
NSString * boundary = @"---------------------------1473780983146649988274664449";

// Body of the POST method
NSMutableData * body = [NSMutableData data];

// The body must start with the boundary preceded by two hyphens, followed
// by a carriage return and newline pair.
```

```objc
//
// Notice that we prepend two additional hyphens to the boundary when
// we actually use it as part of the body data.
//
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n",boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

// This is followed by a series of headers for the first field and then
// TWO CR-LF pairs.
[body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data;
name=\"tag_name\"\r\n\r\n"] dataUsingEncoding:NSUTF8StringEncoding]];

// Next is the actual data for that field (called "tag_name") followed by
// a CR-LF pair, a boundary, and another CR-LF pair.
[body appendData:[strippedCompanyName dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

// Encode the filename and image data as the "userfile" CGI parameter.
// This is similar to the previous field, except that it is being sent
// as an actual file attachment rather than a blob of data, which means
// it has both a filename and the actual file contents.
//
// IMPORTANT: The filename MUST be plain ASCII (and if encoded like this,
//            must not include quotation marks in the filename).
//
NSString * picFileName = [NSString stringWithFormat:@"photoName"];
NSString * appendDataString = [NSString stringWithFormat:@"Content-Disposition: form-data;
name=\"userfile\"; filename=\"%@.jpg\"\r\n", picFileName];
[body appendData:[appendDataString dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[@"Content-Type: application/octet-stream\r\n\r\n"
dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[NSData dataWithData:imageData]];

// Close the request body with one last boundary with two
// additional hyphens prepended **and** two additional hyphens appended.
[body appendData:[[NSString stringWithFormat:@"\r\n--%@--\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

// Create the session
// We can use the delegate to track upload progress and disable cacheing
NSURLSessionConfiguration * defaultConfigObject = [NSURLSessionConfiguration
defaultSessionConfiguration];
defaultConfigObject.requestCachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
NSURLSession * defaultSession = [NSURLSession sessionWithConfiguration: defaultConfigObject
delegate: self delegateQueue: [NSOperationQueue mainQueue]];

// Data uploading task.
NSURL * url = [NSURL URLWithString:@"https://server.io/api/script.php"];
NSMutableURLRequest * request = [NSMutableURLRequest requestWithURL:url];
NSString * contentType = [NSString stringWithFormat:@"multipart/form-data; boundary=%@",boundary];
[request addValue:contentType forHTTPHeaderField:@"Content-Type"];
request.HTTPMethod = @"POST";
request.HTTPBody = body;
NSURLSessionDataTask * uploadTask = [defaultSession dataTaskWithRequest:request];
[uploadTask resume];
```

This creates and fires the NSURLSession request just as before, and as a result the delegate methods will behave exactly the same way. Make sure that the script the image is being sent to (located at the url in the variable url) is expecting an image and can parse it correctly.

# Chapter 76: NSUserDefaults

## Section 76.1: Setting values

To set a value in `NSUserDefaults`, you can use the following functions:

**Swift < 3**

```
setBool(_:forKey:)
setFloat(_:forKey:)
setInteger(_:forKey:)
setObject(_:forKey:)
setDouble(_:forKey:)
setURL(_:forKey:)
```

**Swift 3**

In Swift 3 the names of function is changed to `set` insted of `set` folloed by the type.

```
set(_:forKey:)
```

**Objective-C**

```
-(void)setBool:(BOOL)value forKey:(nonnull NSString *)defaultName;
-(void)setFloat:(float)value forKey:(nonnull NSString *)defaultName;
-(void)setInteger:(NSInteger)value forKey:(nonnull NSString *)defaultName;
-(void)setObject:(nullable id)value forKey:(nonnull NSString *)defaultName;
-(void)setDouble:(double)value forKey:(nonnull NSString *)defaultName;
-(void)setURL:(nullable NSURL *)value forKey:(nonnull NSString *)defaultName;
```

Example usage would be:

**Swift < 3**

```
NSUserDefaults.standardUserDefaults.setObject("Netherlands", forKey: "HomeCountry")
```

**Swift 3**

```
UserDefaults.standard.set("Netherlands", forKey: "HomeCountry")
```

**Objective-C**

```
[[NSUserDefaults standardUserDefaults] setObject:@"Netherlands" forKey:@"HomeCountry"];
```

**Custom objects**

To save custom objects into the `NSUserDefaults` you need to make your CustomClass confirm to protocol of `NSCoding`. You need to implement the following methods: **Swift**

```
public func encodeWithCoder(aCoder: NSCoder) {
    aCoder.encodeObject(name, forKey:"name")
    aCoder.encodeObject(unitId, forKey: "unitId")
}

required public init(coder aDecoder: NSCoder) {
    super.init()
    name = aDecoder.decodeObjectForKey("name") as? String
    unitId = aDecoder.decodeIntegerForKey("unitId") as? NSInteger
}
```

**Objective-C**

```
- (id)initWithCoder:(NSCoder *)coder {
    self = [super init];
    if (self) {
        name = [coder decodeObjectForKey:@"name"];
        unitId = [coder decodeIntegerForKey:@"unitId"];
```

```
    }
    return self;
}

- (void)encodeWithCoder:(NSCoder*)coder {
    [coder encodeObject:name forKey:@"name"];
    [coder encodeInteger:unitId forKey:@"unitId"];
}
```

# Section 76.2: UserDefaults uses in Swift 3

*Every application needed to store User Session or User related details inside application in UserDefaults.So we made whole logic inside a Class for managing UserDefaults better way.*

**Swift 3**

```
import Foundation

public struct Session {

    fileprivate static let defaults = UserDefaults.standard

    enum userValues: String {
        case auth_token
        case email
        case fname
        case mobile
        case title
        case userId
        case userType
        case OTP
        case isApproved
    }


    //MARK: - Getting here User Details
    static func getUserSessionDetails()->[String:AnyObject]? {
        let dictionary = defaults.object(forKey: "LoginSession") as? [String:AnyObject]
        return  dictionary
    }

    //MARK: - Saving Device Token
    static func saveDeviceToken(_ token:String){
        guard (gettingDeviceToken() ?? "").isEmpty else {
            return
        }
        defaults.removeObject(forKey: "deviceToken")
        defaults.set(token, forKey: "deviceToken")
        defaults.synchronize()
    }

    //MARK: - Getting Token here
    static func gettingDeviceToken()->String?{
        let token = defaults.object(forKey: "deviceToken") as? String
        if token == nil{
            return ""
        }else{ return token}
    }

    //MARK: - Setting here User Details
```

```
static func setUserSessionDetails(_ dic :[String : AnyObject]){
    defaults.removeObject(forKey: "LoginSession")
    defaults.set(dic, forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: Removing here all Default Values
static func userSessionLogout(){
    //Set Activity
    defaults.removeObject(forKey: "LoginSession")
    defaults.synchronize()
}

//MARK: - Get value from session here
static func getUserValues(value: userValues) -> String? {
    let dic = getUserSessionDetails() ?? [:]
    guard let value = dic[value.rawValue] else{
        return ""
    }
    return value as? String
}

}
```

**Use of UserDefaults Class**

```
//Saving user Details
Session.setUserSessionDetails(json ?? [:])

//Retriving user Details
let userId = Session.getUserValues(value: .userId) ?? ""
```

# Section 76.3: Use Managers to Save and Read Data

While you can use the NSUserDefaults methods anywhere, it can sometimes be better to define a manager that saves and reads from NSUserDefaults for you and then use that manager for reading or writing your data.

Suppose that we want to save a user's score into NSUserDefaults. We can create a class like the one below that has at two methods: setHighScore and highScore. Anywhere you want to access the high scores, create an instance of this class.

**Swift**

```
public class ScoreManager: NSObject {

    let highScoreDefaultKey = "HighScoreDefaultKey"

    var highScore = {
        set {
            // This method includes your implementation for saving the high score
            // You can use NSUserDefaults or any other data store like CoreData or
            // SQLite etc.

            NSUserDefaults.standardUserDefaults().setInteger(newValue, forKey: highScoreDefaultKey)
            NSUserDefaults.standardUserDefaults().synchronize()
        }
        get {
        //This method includes your implementation for reading the high score

            let score = NSUserDefaults.standardUserDefaults().objectForKey(highScoreDefaultKey)
```

```
            if (score != nil) {
                return score.integerValue;
            } else {
                //No high score available, so return -1
                return -1;
            }
        }
    }
}
```

**Objective-C**

```objc
#import "ScoreManager.h"

#define HIGHSCRORE_KEY @"highScore"

@implementation ScoreManager

- (void)setHighScore:(NSUInteger) highScore {
    // This method includes your implementation for saving the high score
    // You can use NSUserDefaults or any other data store like CoreData or
    // SQLite etc.

    [[NSUserDefaults standardUserDefaults] setInteger:highScore forKey:HIGHSCRORE_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (NSInteger)highScore
{

    //This method includes your implementation for reading the high score

    NSNumber *highScore = [[NSUserDefaults standardUserDefaults] objectForKey:HIGHSCRORE_KEY];
    if (highScore) {
        return highScore.integerValue;
    }else
    {
        //No high score available, so return -1

        return -1;
    }

}

@end
```

The advantages are that:

1. The implementation of your read and write process is only in one place and you can change it (for example switch from `NSUserDefaults` to Core Data) whenever you want and not worry about changing all places that you are working with the high score.

2. Simply call only one method when you want to access to score or write it.

3. Simply debug it when you see a bug or something like this.

> **Note**
>
> If you are worried about synchronization, it is better to use a singleton class that manages the synchronization.

---

# Section 76.4: Saving Values

`NSUserDefaults` are written to disk periodically by the system, but there are times when you want your changes saved immediately, such as when the app transitions into background state. This is done by calling `synchronize`.

**Swift**

```
NSUserDefaults.standardUserDefaults().synchronize()
```

**Objective-C**

```
[[NSUserDefaults standardUserDefaults] synchronize];
```

# Section 76.5: Clearing NSUserDefaults

**Swift**

```
let bundleIdentifier = NSBundle.mainBundle().bundleIdentifier()

NSUserDefaults.standardUserDefaults().removePersistentDomainForName(bundleIdentifier)
```

**Objective-C**

```
NSString *bundleIdentifier = [[NSBundle mainBundle] bundleIdentifier];

[[NSUserDefaults standardUserDefaults] removePersistentDomainForName: bundleIdentifier];
```

# Section 76.6: Getting Default Values

To get a value in NSUserDefaults you can use the following functions:

**Swift**

```
arrayForKey(_:)
boolForKey(_:)
dataForKey(_:)
dictionaryForKey(_:)
floatForKey(_:)
integerForKey(_:)
objectForKey(_:)
stringArrayForKey(_:)
stringForKey(_:)
doubleForKey(_:)
URLForKey(_:)
```

**Objective-C**

```
-(nullable NSArray *)arrayForKey:(nonnull NSString *)defaultName;
-(BOOL)boolForKey:(nonnull NSString *)defaultName;
-(nullable NSData *)dataForKey:(nonnull NSString *)defaultName;
-(nullable NSDictionary<NSString *, id> *)dictionaryForKey:(nonnull NSString *)defaultName;
-(float)floatForKey:(nonnull NSString *)defaultName;
-(NSInteger)integerForKey:(nonnull NSString *)defaultName;
-(nullable id)objectForKey:(nonnull NSString *)key;
-(nullable NSArray<NSString *> *)stringArrayForKey:(nonnull NSString *)defaultName;
-(nullable NSString *)stringForKey:(nonnull NSString *)defaultName;
-(double)doubleForKey:(nonnull NSString *)defaultName;
-(nullable NSURL *)URLForKey:(nonnull NSString *)defaultName;
```

Example usage would be:

**Swift**

```
let homeCountry = NSUserDefaults.standardUserDefaults().stringForKey("HomeCountry")
```

**Objective-C**

```objc
NSString *homeCountry = [[NSUserDefaults standardUserDefaults] stringForKey:@"HomeCountry"];
```

# Chapter 77: NSHTTPCookieStorage

## Section 77.1: Store and read the cookies from NSUserDefault

```swift
import Foundation


class CookiesSingleton {

static let instance : CookiesSingleton = CookiesSingleton()
static var enableDebug = true

func loadCookies() {
    if let cookiesDetails = NSUserDefaults.standardUserDefaults().objectForKey("customeWebsite")  {
        for (keys,_) in cookiesDetails as! NSDictionary{
                if let  cookieDict = NSUserDefaults.standardUserDefaults().objectForKey(keys as!
String){
                    if let cookie = NSHTTPCookie(properties:cookieDict as! [String:AnyObject]) {
                        NSHTTPCookieStorage.sharedHTTPCookieStorage().setCookie(cookie)
                        if(CookiesSingleton.enableDebug){
                            print("Each Cookies",cookieDict)
                        }
                    }
                }
        }
    }
}

func  removeCookies(){
    NSURLCache.sharedURLCache().removeAllCachedResponses()
    NSURLCache.sharedURLCache().diskCapacity = 0
    NSURLCache.sharedURLCache().memoryCapacity = 0

    let storage : NSHTTPCookieStorage = NSHTTPCookieStorage.sharedHTTPCookieStorage()
    for cookie in storage.cookies! {
        storage.deleteCookie(cookie as NSHTTPCookie)
    }

    NSUserDefaults.standardUserDefaults().setValue("", forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Removed")
    }
}

func saveCookies() {

    let cookieArray = NSMutableArray()
    let savedC = NSHTTPCookieStorage.sharedHTTPCookieStorage().cookies

    let allCookiesDic:NSMutableDictionary = NSMutableDictionary()

    for c : NSHTTPCookie in savedC! {

        let cookieProps = NSMutableDictionary()
        cookieArray.addObject(c.name)
        cookieProps.setValue(c.name, forKey: NSHTTPCookieName)
        cookieProps.setValue(c.value, forKey: NSHTTPCookieValue)
        cookieProps.setValue(c.domain, forKey: NSHTTPCookieDomain)
```

```
        cookieProps.setValue(c.path, forKey: NSHTTPCookiePath)
        cookieProps.setValue(c.version, forKey: NSHTTPCookieVersion)
        cookieProps.setValue(NSDate().dateByAddingTimeInterval(2629743), forKey:
NSHTTPCookieExpires)

        allCookiesDic.setValue(cookieProps, forKey: c.name)

    }
    NSUserDefaults.standardUserDefaults().setValue(allCookiesDic, forKey: "customeWebsite")
    NSUserDefaults.standardUserDefaults().synchronize()

    if(CookiesSingleton.enableDebug){
        print("Cookies Saved")
    }
}

}
```

# Chapter 78: NSURLConnection

## Section 78.1: Delegate methods

//conform the NSURLConnectionDelegate protocol.

```
@interface ViewController : UIViewController<NSURLConnectionDelegate>
{
    NSMutableData *_responseData;
}
```

//Implementation of the NSURLConnection protocol methods.

```
#pragma mark NSURLConnection Delegate Methods

- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response {
    // A response has been received, this is where we initialize the instance var you created
    // so that we can append data to it in the didReceiveData method
    // Furthermore, this method is called each time there is a redirect so reinitializing it
    // also serves to clear it
    _responseData = [[NSMutableData alloc] init];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {
    // Append the new data to the instance variable you declared
    [_responseData appendData:data];
}

- (NSCachedURLResponse *)connection:(NSURLConnection *)connection
                willCacheResponse:(NSCachedURLResponse*)cachedResponse {
    // Return nil to indicate not necessary to store a cached response for this connection
    return nil;
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    // The request is complete and data has been received
    // You can parse the stuff in your instance variable now

}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error {
    // The request has failed for some reason!
    // Check the error var
}
```

## Section 78.2: Synchronous Request

```
NSURLRequest * urlRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:@"http://google.com"]];
NSURLResponse * response = nil;
NSError * error = nil;
NSData * data = [NSURLConnection sendSynchronousRequest:urlRequest
                                    returningResponse:&response
                                                error:&error];

if (error == nil)
{
    // Parse data here
```

```
}
```

# Section 78.3: Asynchronous request

```
// Create the request instance.
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:@"http://google.com"]];

// Create url connection and fire request
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];
```

# Chapter 79: NSURL

## Section 79.1: How to get last string component from NSURL String

```
NSURL *url = [NSURL URLWithString:@"http://www.example.com/images/apple-tree.jpg"];
NSString *fileName = [url lastPathComponent];
// fileName = "apple-tree.jpg"
```

## Section 79.2: How to get last string component from URL (NSURL) in Swift

Swift 2.3

```
let url = NSURL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

Swift 3.0

```
let url = URL(string: "http://google.com/lastPath")
let lastPath = url?.lastPathComponent
```

# Chapter 80: NSData

## Section 80.1: Converting NSData to HEX string

NSData can be represented as hexadecimal string, similar to what it outputs in its `description` method.

**Swift**

```swift
extension NSData {

    func hexString() -> String {
        return UnsafeBufferPointer<UInt8>(start: UnsafePointer<UInt8>(bytes), count: length)
            .reduce("") { $0 + String(format: "%02x", $1) }
    }

}
```

**Objective-C**

```objective-c
@implementation NSData (HexRepresentation)

- (NSString *)hexString {
    const unsigned char *bytes = (const unsigned char *)self.bytes;
    NSMutableString *hex = [NSMutableString new];
    for (NSInteger i = 0; i < self.length; i++) {
        [hex appendFormat:@"%02x", bytes[i]];
    }
    return [hex copy];
}

@end
```

## Section 80.2: Creating NSData objects

**Using a file**
**Swift**

```swift
let data = NSData(contentsOfFile: filePath) //assuming filePath is a valid path
```

**Objective-C**

```objective-c
NSData *data = [NSData dataWithContentsOfFile:filePath]; //assuming filePath is a valid path
```

**Using a String object**
**Swift**

```swift
let data = (string as NSString).dataUsingEncoding(NSUTF8StringEncoding) //assuming string is a
String object
```

**Objective-C**

```objective-c
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding]; //assuming string is a String
object
```

## Section 80.3: Converting NSData to other types

**To String**
**Swift**

```swift
let string = String(NSString(data: data, encoding: NSUTF8StringEncoding)) //assuming data is a
valid NSData object
```

**Objective-C**

```objective-c
NSString *string = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding]; //assuming
```

```
data is a valid NSData object
[string release];
```

**To Array**

**Swift**

```swift
let array = data.bytes as! NSMutableArray //assuming data is a valid NSData object
```

**Objective-C**

```objc
NSMutableArray *array = (NSMutableArray *)[data bytes]; //assuming data is a valid NSData object
```

**To Bytes Array**

**Swift**

```swift
let bytesArray = data.bytes as! UInt8 //assuming data is a valid NSData object
```

**Objective-C**

```objc
UInt8 *bytesArray = (UInt8 *)data.bytes; //assuming data is a valid NSData object
```

# Chapter 81: NSInvocation

## Section 81.1: NSInvocation Objective-C

Refer to this original Post by e.James

According to Apple's NSInvocation class reference:

> An `NSInvocation` is an Objective-C message rendered static, that is, it is an action turned into an object.

And, in a *little* more detail:

The concept of messages is central to the objective-c philosophy. Any time you call a method, or access a variable of some object, you are sending it a message. `NSInvocation` comes in handy when you want to send a message to an object at a different point in time, or send the same message several times. `NSInvocation` allows you to *describe* the message you are going to send, and then *invoke* it (actually send it to the target object) later on.

For example, let's say you want to add a string to an array. You would normally send the `addObject:` message as follows:

```
[myArray addObject:myString];
```

Now, let's say you want to use `NSInvocation` to send this message at some other point in time:

First, you would prepare an `NSInvocation` object for use with `NSMutableArray`'s `addObject:` selector:

```
NSMethodSignature * mySignature = [NSMutableArray
    instanceMethodSignatureForSelector:@selector(addObject:)];
NSInvocation * myInvocation = [NSInvocation
    invocationWithMethodSignature:mySignature];
```

Next, you would specify which object to send the message to:

```
[myInvocation setTarget:myArray];
```

Specify the message you wish to send to that object:

```
[myInvocation setSelector:@selector(addObject:)];
```

And fill in any arguments for that method:

```
[myInvocation setArgument:&myString atIndex:2];
```

Note that object arguments must be passed by pointer. Thank you to Ryan McCuaig for pointing that out, and please see Apple's documentation for more details.

At this point, `myInvocation` is a complete object, describing a message that can be sent. To actually send the message, you would call:

```
[myInvocation invoke];
```

This final step will cause the message to be sent, essentially executing `[myArray addObject:myString];`.

Think of it like sending an email. You open up a new email (`NSInvocation` object), fill in the address of the person (object) who you want to send it to, type in a message for the recipient (specify a `selector` and arguments), and then click "send" (call `invoke`).

See Using NSInvocation for more information.

`NSUndoManager` uses `NSInvocation` objects so that it can *reverse* commands. Essentially, what you are doing is creating an `NSInvocation` object to say: "Hey, if you want to undo what I just did, send this message to that object, with these arguments". You give the `NSInvocation` object to the `NSUndoManager`, and it adds that object to an array of undoable actions. If the user calls "Undo", `NSUndoManager` simply looks up the most recent action in the array, and invokes the stored `NSInvocation` object to perform the necessary action.

See Registering Undo Operations for more details.

# Chapter 82: NSUserActivity

An NSUserActivity object can be used to coordinate significant events in an app with the system. It is the basis for [Handoff](Handoff) between different devices running iOS and macOS. Additionally, it may also be used to improve public-indexing and augment or create Spotlight Search results for an app. As of iOS 10, it may also be used to coordinate interactions between your app and Siri using SiriKit.

## Section 82.1: Creating a NSUserActivity

To create a `NSUserActivity` object, your app must declare the types of activities it supports in its `Info.plist` file. Supported activities are defined by your application and should be unique. An activity is defined using a reverse-domain style naming scheme (i.e. "com.companyName.productName.activityName"). Here is what an entry in your Info.plist might look like:

| **Key** | **Value** |
|---|---|
| NSUserActivityTypes [Array] | |
| - item0 | com.companyName.productName.activityName01 |
| - item1 | com.companyName.productName.activityName02 |

Once you have defined all supported activity types, you may begin to access and use them in your application's code.

To create a `NSUserActivity` object you must do the following

```
// Initialize the activity object and set its type from one of the ones specified in your app's
plist
NSUserActivity *currentActivity = [[NSUserActivity alloc]
initWithActivityType:@"com.companyName.productName.activityName01"];

// Set the title of the activity.
// This title may be displayed to the user, so make sure it is localized and human-readable
currentActivity.title = @"Current Activity";

// Configure additional properties like userInfo which will be included in the activity
currentActivity.userInfo = @{@"informationKey" : @"value"};

// Configure the activity so the system knows what may be done with it
// It is important that you only set YES to tasks that your application supports
// In this example, we will only enable the activity for use with Handoff
[currentActivity setEligibleForHandoff:YES];
[currentActivity setEligibleForSearch:NO]; // Defaults to NO
[currentActivity setEligibleForPublicIndexing:NO]; // Defaults to NO

// Set this activity as the current user activity
// Only one activity may be current at a time on a device. Calling this method invalidates any
other current activities.
[currentActivity becomeCurrent];
```

After this, the activity above should be available for Handoff (although more work is required to properly handle the "Handoff").

# Chapter 83: NSPredicate

## Section 83.1: Form validation using NSPredicate

```
NSString *emailRegex = @"[A-Z0-9a-z]([A-Z0-9a-z._-]{0,64})+[A-Z0-9a-z]+@[A-Z0-9a-z]+([A-Za-z0-9.-
]{0,64})+([A-Z0-9a-z])+\\.[A-Za-z]{2,4}";    NSString *firstNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *firstNameRegex = @"[ 0-9A-Za-z]{2,32}$";
NSString *lastNameRegex = @"[0-9A-Za-z\"'-]{2,32}$";
NSString *mobileNumberRegEx = @"^[0-9]{10}$";
NSString *zipcodeRegEx = @"^[0-9]{5}$";
NSString *SSNRegEx = @"^\\d{3}-?\\d{2}-?\\d{4}$";
NSString *addressRegEx = @"^[ A-Za-z0-9]{2,32}$";
NSString *cityRegEx = @"^[ A-Za-z0-9]{2,25}$";
NSString *PINRegEx = @"^[0-9]{4}$";
NSString *driversLiscRegEx = @"^[0-9a-zA-Z]{5,20}$";


-(BOOL)validateEmail {
    //Email address field should give an error when the email address begins with ".","-","_" .
    NSPredicate *emailPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegex];

    return ([emailPredicate evaluateWithObject:self.text] && self.text.length <= 64 && ([self.text
rangeOfString:@".."].location == NSNotFound));
}

- (BOOL)validateFirstName {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateLastName {
    NSPredicate *lastNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
lastNameRegex];
    return [lastNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateAlphaNumericMin2Max32 {
    NSPredicate *firstNamePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
firstNameRegex];
    return [firstNamePredicate evaluateWithObject:self.text];
}

- (BOOL)validateMobileNumber {
    NSString *strippedMobileNumber =  [[[[self.text stringByReplacingOccurrencesOfString:@"("
withString:@""]
                                        stringByReplacingOccurrencesOfString:@")" withString:@""]
                                        stringByReplacingOccurrencesOfString:@"-" withString:@""]
                                        stringByReplacingOccurrencesOfString:@" " withString:@""];

    NSPredicate *mobileNumberPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
mobileNumberRegEx];

    return [mobileNumberPredicate evaluateWithObject:strippedMobileNumber];
}

- (BOOL)validateZipcode {
    NSPredicate *zipcodePredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
zipcodeRegEx];

    return [zipcodePredicate evaluateWithObject:self.text];
```

```
}

- (BOOL)validateSSN {
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", SSNRegEx];

return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateAddress {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", addressRegEx];

    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validateCity {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", cityRegEx];
    return [predicate evaluateWithObject:self.text];
}

- (BOOL)validatePIN {
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", PINRegEx];
    return [predicate evaluateWithObject:self.text];
}
    - (BOOL)validateDriversLiscNumber {
    if([self.text length] > 20) {
        return NO;
    }
    NSPredicate *driversLiscPredicate = [NSPredicate predicateWithFormat:@"SELF MATCHES %@",
driversLiscRegEx];

    return [driversLiscPredicate evaluateWithObject:self.text];
}
```

## Section 83.2: Creating an NSPredicate Using predicateWithBlock

**Objective-C**

```
NSPredicate *predicate = [NSPredicate predicateWithBlock:^BOOL(id item,
                                                    NSDictionary *bindings) {
    return [item isKindOfClass:[UILabel class]];
}];
```

**Swift**

```
let predicate = NSPredicate { (item, bindings) -> Bool in
    return item.isKindOfClass(UILabel.self)
}
```

In this example, the predicate will match items that are of the class `UILabel`.

## Section 83.3: Creating an NSPredicate Using predicateWithFormat

**Objective-C**

```
NSPredicate *predicate = [NSPredicate predicateWithFormat: @"self[SIZE] = %d", 5)];
```

**Swift**

```
let predicate = NSPredicate(format: "self[SIZE] >= %d", 5)
```

In this example, the predicate will match items that are arrays with length of at least 5.

## Section 83.4: Creating an NSPredicate with Substitution Variables

An `NSPredicate` can use substitution variables to allow values to be bound on the fly.

**Objective-C**

```objc
NSPredicate *template = [NSPredicate predicateWithFormat: @"self BEGINSWITH $letter"];
NSDictionary *variables = @{ @"letter": @"r" };
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: variables];
```

**Swift**

```swift
let template = NSPredicate(format: "self BEGINSWITH $letter")
let variables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(variables)
```

The template predicate is not modified by `predicateWithSubstitutionVariables`. Instead, a copy is created, and that copy receives the substitution variables.

## Section 83.5: NSPredicate with `AND`, `OR` and `NOT` condition

Conditional predicate will be cleaner and safer by using the `NSCompoundPredicate` class which provides basic boolean operators for the given predicates.

**Objective-c**
**AND - Condition**

```objc
  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
  NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
  NSPredicate *combinedPredicate = [NSCompoundPredicate andPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

**OR - Condition**

```objc
  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
  NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
  NSPredicate *combinedPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:
@[predicate,anotherPredicate]];
```

**NOT - Condition**

```objc
  NSPredicate *predicate = [NSPredicate predicateWithFormat:@"samplePredicate"];
  NSPredicate *anotherPredicate = [NSPredicate predicateWithFormat:@"anotherPredicate"];
  NSPredicate *combinedPredicate = [NSCompoundPredicate notPredicateWithSubpredicate:
@[predicate,anotherPredicate]];
```

## Section 83.6: Using NSPredicate to Filter an Array

**Objective-C**

```objc
NSArray *heroes = @[@"tracer", @"bastion", @"reaper", @"junkrat", @"roadhog"];

NSPredicate *template = [NSPredicate predicateWithFormat:@"self BEGINSWITH $letter"];

NSDictionary *beginsWithRVariables = @{ @"letter": @"r"};
NSPredicate *beginsWithR = [template predicateWithSubstitutionVariables: beginsWithRVariables];

NSArray *beginsWithRHeroes = [heroes filteredArrayUsingPredicate: beginsWithR];
// ["reaper", "roadhog"]
```

```
NSDictionary *beginsWithTVariables = @{ @"letter": @"t"};
NSPredicate *beginsWithT = [template predicateWithSubstitutionVarables: beginsWithTVariables];

NSArray *beginsWithTHeroes = [heroes filteredArrayUsingPredicate: beginsWithT];
// ["tracer"]
```

**Swift**

```
let heroes = ["tracer", "bastion", "reaper", "junkrat", "roadhog"]

let template = NSPredicate(format: "self BEGINSWITH $letter")

let beginsWithRVariables = ["letter": "r"]
let beginsWithR = template.predicateWithSubstitutionVariables(beginsWithRVariables)

let beginsWithRHeroes = heroes.filter { beginsWithR.evaluateWithObject($0) }
// ["reaper", "roadhog"]

let beginsWithTVariables = ["letter": "t"]
let beginsWithT = template.predicateWithSubstitutionVariables(beginsWithTVariables)

let beginsWithTHeroes = heroes.filter { beginsWithT.evaluateWithObject($0) }
// ["tracer"]
```

# Chapter 84: NSBundle

## Section 84.1: Getting Bundle by Path

1. Locating a Cocoa bundle using its path

To obtain the bundle at a specific path using Cocoa, call the ***bundleWithPath:*** class method of the **NSBundle**

```
NSBundle *myBundle;
// obtain a reference to a loadable bundle
myBundle = [NSBundle bundleWithPath:@"/Library/MyBundle.bundle";
```

2. Locating a Cocoa Foundation bundle using its Path

To obtain the bundle at a specific path using Core Foundation, call the ***CFBundleCreate*** function and must use **CFURLRef** type.

```
CFURLRef bundleURL;
CFBundleRef myBundle;
// Make a CFURLRef from the CFString representation of the bundle's path.
bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault,
 CFSTR("/Library/MyBundle.bundle"), kCFURLPOSIXPathStyle, true);
// Make a bundle instance using the URLRef.
myBundle = CFBundleCreate(kCFAllocatorDefault, bundeURL);
// You can release the URL now.
CFRelease(bundleURL);
// Use the bundle ...
// Release the bundle when done.
CFRelease(myBundle);
```

## Section 84.2: Getting the Main Bundle

1. Getting a reference to the main bundle using Cocoa.

To get the main bundle in Cocoa application, call the ***mainBundle*** class method of the **NSBundle** class.

```
NSBundle *mainBundle;
// Get the main bundle for the app;
mainBundle = [NSBundle mainBundle];
```

2. Getting a reference to the main bundle using Core Foundation.

Use the ***CFBundleGetMainBundle*** function to retrieve the main bundle for your C-based application.

```
CFBundleRef mainBundle;
// Get the main bundle for the app
mainBundle = CFBundleGetMainBundle();
```

# Chapter 85: CAAnimation

## Section 85.1: Animate a view from one position to another

**Objective-C**

```objc
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"position.x"];
animation.fromValue = @0;
animation.toValue = @320;
animation.duration = 1;

[_label.layer addAnimation:animation forKey:@"basic"];
```

**Swift**

```swift
let animation = CABasicAnimation(keyPath: "position.x")
animation.fromValue = NSNumber(value: 0.0)
animation.toValue = NSNumber(value: 320.0)

_label.layer.addAnimation(animation, forKey: "basic")
```

The view will move from 0 to 320 horizontally. if you want to Move view to Vertically just replace keypath like this:

```
"position.y"
```

## Section 85.2: Animate View - Toss

**OBJECTIVE-C**

```objc
CATransition* transition = [CATransition animation];
transition.startProgress = 0;
transition.endProgress = 1.0;
transition.type = @"flip";
transition.subtype = @"fromLeft";
transition.duration = 0.8;
transition.repeatCount = 5;
[_label.layer addAnimation:transition forKey:@"transition"];
```

**SWIFT**

```swift
var transition = CATransition()
transition.startProgress = 0
transition.endProgress = 1.0
transition.type = "flip"
transition.subtype = "fromLeft"
transition.duration = 0.8
transition.repeatCount = 5
label.layer.addAnimation(transition, forKey: "transition")
```

## Section 85.3: Push View Animation

**Objective C**

```objc
CATransition *animation = [CATransition animation];
[animation setSubtype:kCATransitionFromRight];//kCATransitionFromLeft
[animation setDuration:0.5];
[animation setType:kCATransitionPush];
[animation setTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut]];
[[yourView layer] addAnimation:animation forKey:@"SwitchToView1"];
```

**Swift**

```
let animation = CATransition()
animation.subtype = kCATransitionFromRight//kCATransitionFromLeft
animation.duration = 0.5
animation.type = kCATransitionPush
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
yourView.layer.addAnimation(animation, forKey: "SwitchToView1")
```

## Section 85.4: Revolve View

```
 CGRect boundingRect = CGRectMake(-150, -150, 300, 300);

CAKeyframeAnimation *orbit = [CAKeyframeAnimation animation];
orbit.keyPath = @"position";
orbit.path = CFAutorelease(CGPathCreateWithEllipseInRect(boundingRect, NULL));
orbit.duration = 4;
orbit.additive = YES;
orbit.repeatCount = HUGE_VALF;
orbit.calculationMode = kCAAnimationPaced;
orbit.rotationMode = kCAAnimationRotateAuto;

[_label.layer addAnimation:orbit forKey:@"orbit"];
```

## Section 85.5: Shake View

**Objective-C**

```
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:@"position.x"];
animation.values = @[ @0, @10, @-10, @10, @0 ];
animation.keyTimes = @[ @0, @(1 / 6.0), @(3 / 6.0), @(5 / 6.0), @1 ];
animation.duration = 0.4;
animation.additive = YES;
[_label.layer addAnimation:animation forKey:@"shake"];
```

**Swift 3**

```
let animation = CAKeyframeAnimation(keyPath: "position.x")
animation.values = [ 0, 10, -10, 10, 0 ]
animation.keyTimes = [ 0, NSNumber(value: (1 / 6.0)), NSNumber(value: (3 / 6.0)), NSNumber(value:
(5 / 6.0)), 1 ]
animation.duration = 0.4
animation.isAdditive = true
label.layer.add(animation, forKey: "shake")
```

# Chapter 86: Concurrency

| | |
|---|---|
| queue | **The queue that the code in the dispatch block will run in. A *queue* is like (but not exactly the same as) a thread; code in different queues can run in parallel. Use `dispatch_get_main_queue` to get the queue for the main thread To create a new queue, which in turn creates a new thread, use `dispatch_queue_create("QUEUE_NAME", DISPATCH_QUEUE_CONCURRENT)`. The first parameter is the name of the queue, which is displayed in the debugger if you pause while the block is still running. The second parameter doesn't matter unless you want to use the same queue for multiple `dispatch_async` or `dispatch_sync` calls. It describes what happens when another block is put on the same queue; `DISPATCH_QUEUE_CONCURRENT` will cause both blocks to run at the same time, while `DISPATCH_QUEUE_SERIAL` will make the second block wait for the first block to finish** |
| block | Code in this block will run in the queue queue; put code you want to run on the separate queue here. A helpful tip: If you're writing this in Xcode and the block argument has the blue outline around it, double click on the argument and Xcode will automatically make an empty block (this applies to all block arguments in any function or method) |

Related topic: Grand Central Dispatch

## Section 86.1: Dispatch group - waiting for other threads completed

```
dispatch_group_t preapreWaitingGroup = dispatch_group_create();

dispatch_group_enter(preapreWaitingGroup);
[self doAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    // Notify that this task has been completed.
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_enter(preapreWaitingGroup);
[self doOtherAsynchronousTaskWithComplete:^(id someResults, NSError *error) {
    dispatch_group_leave(preapreWaitingGroup);
}]

dispatch_group_notify(preapreWaitingGroup, dispatch_get_main_queue(), ^{
    // This block will be executed once all above threads completed and call dispatch_group_leave
    NSLog(@"Prepare completed. I'm readyyyy");
});
```

**Update 1.** Swift 3 version.

```
let prepareGroup = DispatchGroup()
prepareGroup.enter()
doAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.enter()
doOtherAsynchronousTaskWithComplete() { (someResults, error) in
    // Notify that this task has been completed.
    prepareGroup.leave()
}

prepareGroup.notify(queue: DispatchQueue.main) {
    // This block will be executed once all above threads completed and call dispatch_group_leave
    print("Prepare completed. I'm readyyyy")
}
```

# Section 86.2: Executing on the main thread

When performing tasks asynchronously there typically becomes a need to ensure a piece of code is run on the main thread. For example you may want to hit a REST API asynchronously, but put the result in a UILabel on the screen. Before updating the UILabel you must ensure that your code is run on the main thread:

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    //Perform expensive tasks
    //...

    //Now before updating the UI, ensure we are back on the main thread
    dispatch_async(dispatch_get_main_queue(), ^{
        label.text = //....
    });
}
```

Whenever you update views on the screen, always ensure you are doing so on the main thread, otherwise undefined behavior could occur.

# Section 86.3: Running code concurrently -- Running code while running other code

Say you want to perform in action (in this case, logging "Foo"), while doing something else (logging "Bar"). Normally, if you don't use concurrency, one of these actions is going to be fully executed, and the other run will run only after it's completely finished. But with concurrency, you can make both actions run at the same time:

```
dispatch_async(dispatch_queue_create("Foo", DISPATCH_QUEUE_CONCURRENT), ^{
    for (int i = 0; i < 100; i++) {
        NSLog(@"Foo");
        usleep(100000);
    }
});

for (int i = 0; i < 100; i++) {
    NSLog(@"Bar");
    usleep(50000);
}
```

This will log "Foo" 100 times, pausing for 100ms each time it logs, but it will do all this on a separate thread. While Foo is being logged, "Bar" will also be logged in 50ms intervals, at the same time. You should ideally see an output with "Foo"s and "Bars" mixed together

# Chapter 87: CAGradientLayer

| Parameter | Details |
| --- | --- |
| color | An array of `CGColorRef` objects defining the color of each gradient stop. Animatable. |
| locations | An optional array of `NSNumber` objects defining the location of each gradient stop. Animatable. |
| endPoint | The end point of the gradient when drawn in the layer's coordinate space. Animatable. |
| startPoint | The start point of the gradient when drawn in the layer's coordinate space. Animatable. |
| type | Style of gradient drawn by the layer. Defaults to `kCAGradientLayerAxial`. |

## Section 87.1: Creating a CAGradientLayer

```swift
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.red.cgColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.yellow.cgColor

// Set colors.
gradientLayer.colors = [topColor, bottomColor]

// Set locations of the colors.
gradientLayer.locations = [0.0, 1.0]

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result :

# Section 87.2: Animating a color change in CAGradientLayer

```swift
// Get the current colors of the gradient.
let oldColors = self.gradientLayer.colors

// Define the new colors for the gradient.
let newColors = [UIColor.red.cgColor, UIColor.yellow.cgColor]

// Set the new colors of the gradient.
self.gradientLayer.colors = newColors

// Initialize new animation for changing the colors of the gradient.
let animation: CABasicAnimation = CABasicAnimation(keyPath: "colors")

// Set current color value.
animation.fromValue = oldColors

// Set new color value.
animation.toValue = newColors

// Set duration of animation.
animation.duration = 0.3

// Set animation to remove once its completed.
animation.isRemovedOnCompletion = true

// Set receiver to remain visible in its final state when the animation is completed.
animation.fillMode = kCAFillModeForwards

// Set linear pacing, which causes an animation to occur evenly over its duration.
animation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionLinear)

// Set delegate of animation.
animation.delegate = self

// Add the animation.
self.gradientLayer.addAnimation(animation, forKey: "animateGradientColorChange")
```

Result :
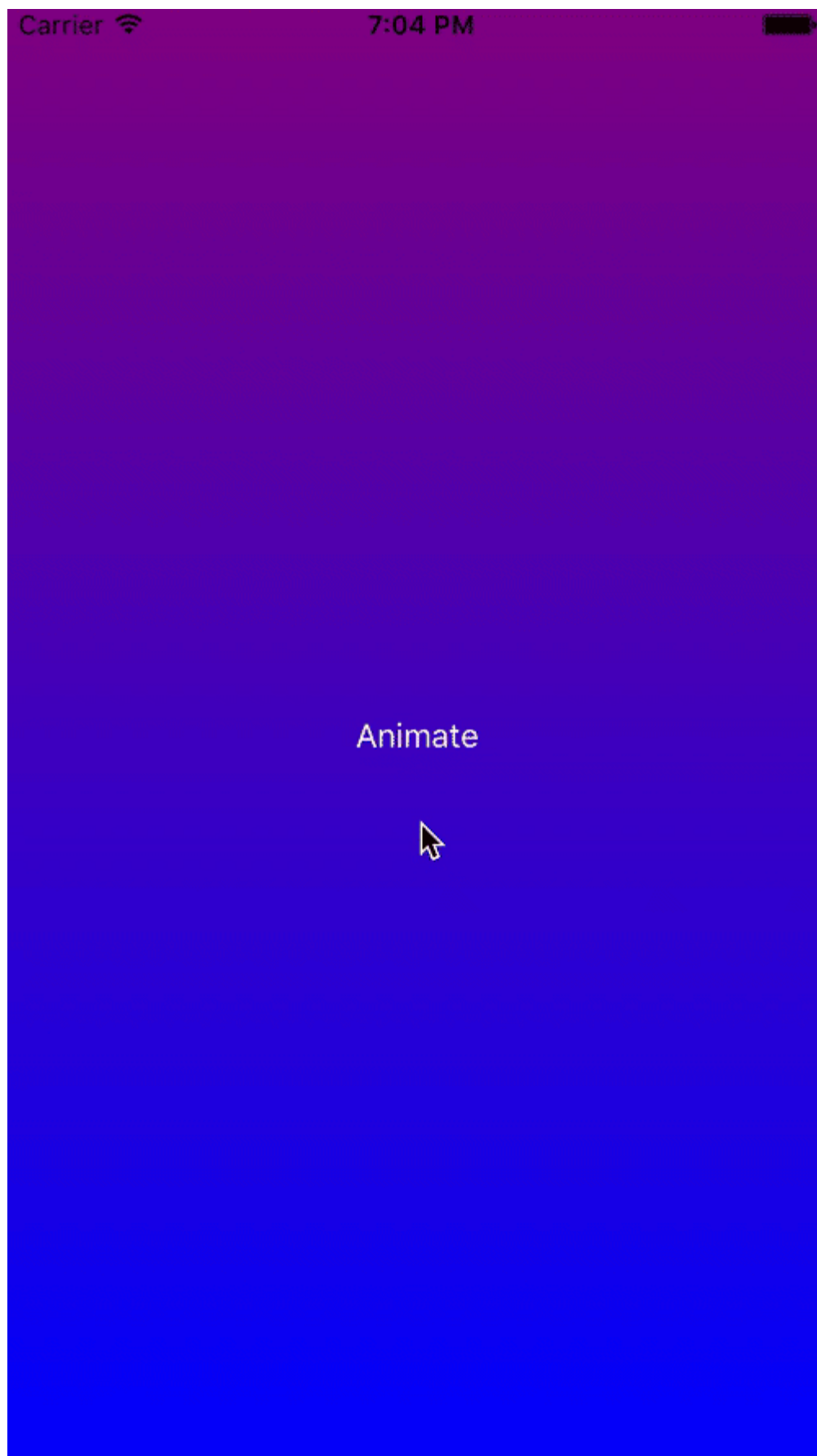
## Section 87.3: Creating a horizontal CAGradientLayer

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
```

```
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.redColor().CGColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.yellowColor().CGColor

// Set colors.
gradientLayer.colors = [topColor, bottomColor]

// Set start point.
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// Set end point.
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

Result:



# Section 87.4: Creating a horizontal CAGradientLayer with multiple colors

```
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.greenColor().CGColor

// Color at the middle of the gradient.
```

```swift
let middleColor: CGColor = UIColor.blueColor().CGColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.blackColor().CGColor

// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]

// Set start point.
gradientLayer.startPoint = CGPoint(x: 0.0, y: 0.5)

// Set end point.
gradientLayer.endPoint = CGPoint(x: 1.0, y: 0.5)

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, atIndex: 0)
```

Result:
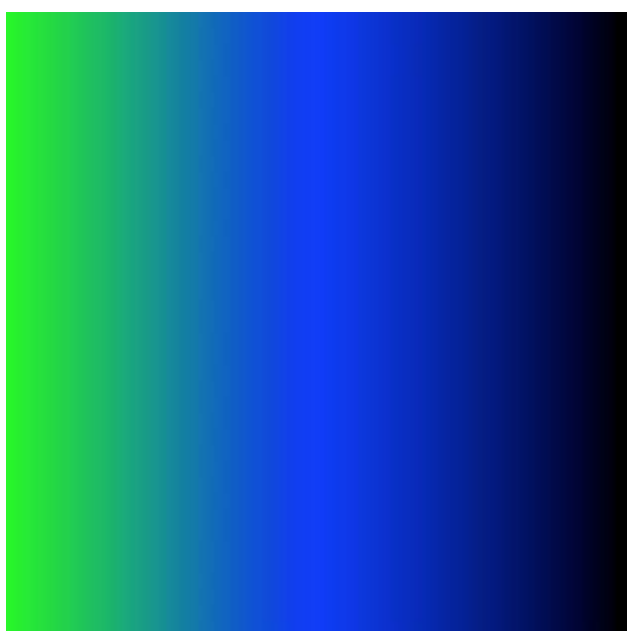


# Section 87.5: Creating a CGGradientLayer with multiple colors

```swift
// View to hold the CAGradientLayer.
let view: UIView = UIView(frame: CGRect(x: 0, y: 0, width: 320, height: 320))

// Initialize gradient layer.
let gradientLayer: CAGradientLayer = CAGradientLayer()

// Set frame of gradient layer.
gradientLayer.frame = view.bounds

// Color at the top of the gradient.
let topColor: CGColor = UIColor.blue.cgColor

// Color at the middle of the gradient.
let middleColor: CGColor = UIColor.yellow.cgColor

// Color at the bottom of the gradient.
let bottomColor: CGColor = UIColor.green.cgColor

// Set colors.
gradientLayer.colors = [topColor, middleColor, bottomColor]
```

```
// Set locations of the colors.
gradientLayer.locations = [0.0, 0.5, 1.0]

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, at: 0)
```

Result:



// Set locations of the colors.
gradientLayer.locations = [0.0, 0.5, 1.0]

// Insert gradient layer into view's layer heirarchy.
view.layer.insertSublayer(gradientLayer, at: 0)
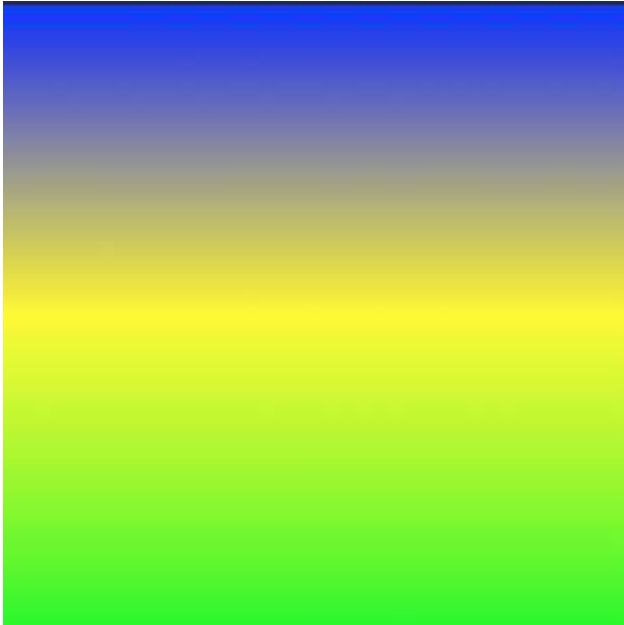
# Chapter 88: Safari Services

## Section 88.1: Open a URL with SafariViewController

Don't forget to import the necessary framework first.

```
import SafariServices
//Objective-C
@import SafariServices;
```

Instantiate a `SafariViewController` instance.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!)
//Objective-C
@import SafariServices;
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL];
```

Optionally you can also tell SafariViewController to enter reading mode if possible once it's done loading.

```
let safariVC = SFSafariViewController(URL: URL(string: "your_url")!, entersReaderIfAvailable: true)
//Objective-C
NSURL *URL = [NSURL URLWithString:[NSString stringWithFormat:@"http://www.google.com"]];
SFSafariViewController *sfvc = [[SFSafariViewController alloc] initWithURL:URL
entersReaderIfAvailable:YES];
```

Present the view controller.

```
present(safariVC, animated: true, completion: nil)
//Objective-C
[self presentViewController:sfvc animated:YES completion:nil];
```

## Section 88.2: Implement SFSafariViewControllerDelegate

You should implement `SFSafariViewControllerDelegate` so that your class is notified when the user hits the Done button on the SafariViewController and you can dismiss it as well.

First declare your class to implement the protocol.

```
class MyClass: SFSafariViewControllerDelegate {

}
```

Implement the delegate method to be notified on dismissal.

```
func safariViewControllerDidFinish(controller: SFSafariViewController) {
    // Dismiss the SafariViewController when done
    controller.dismissViewControllerAnimated(true, completion: nil)
}
```

Don't forget to set your class as the SafariViewController's delegate.

```
let safariVC = SFSafariViewController(URL: yourURL)
safariVC.delegate = self
```

Additional delegate methods you can implement are:

```swift
// Called when the initial URL load is complete.
safariViewController(_ controller: SFSafariViewController, didCompleteInitialLoad
didLoadSuccessfully: Bool) { }

// Called when the user taps an Action button.
safariViewController(_ controller: SFSafariViewController, activityItemsFor URL: URL, title:
String?) -> [UIActivity] { }
```

# Section 88.3: Add Items to Safari Reading List

You can add items to a user's Reading List in Safari by calling the `addItem` method on the `SSReadingList` singleton.

```swift
let readingList = SSReadingList.default()
readingList?.addItem(with: yourURL, title: "optional title", previewText: "optional preview text")
```

The default Reading List can be `nil` if access to the Reading List is not permitted.

Additionally you can check if the Reading List supports a URL by calling `supportsURL`.

```swift
SSReadingList.default().supportsURL(URL(string: "https://example.com")!)
```

This will return either `true` or `false` indicating if the given URL is supported by Safari Reading List. Use this for example to determine whether to show a button to add a URL to the Reading List.

# Chapter 89: CALayer

## Section 89.1: Adding Transforms to a CALayer (translate, rotate, scale)

**Basics**

There are a number of different transforms you can do on a layer, but the basic ones are

- translate (move)
- scale
- rotate



To do transforms on a `CALayer`, you set the layer's `transform` property to a `CATransform3D` type. For example, to translate a layer, you would do something like this:

```
myLayer.transform = CATransform3DMakeTranslation(20, 30, 0)
```

The word `Make` is used in the name for creating the initial transform: CATransform3D**Make**Translation. Subsequent transforms that are applied omit the `Make`. See, for example, this rotation followed by a translation:

```
let rotation = CATransform3DMakeRotation(CGFloat(30.0 * M_PI / 180.0), 20, 20, 0)
myLayer.transform = CATransform3DTranslate(rotation, 20, 30, 0)
```

Now that we have the basis of how to make a transform, let's look at some examples of how to do each one. First, though, I'll show how I set up the project in case you want to play around with it, too.

**Setup**

For the following examples I set up a Single View Application and added a `UIView` with a light blue background to the storyboard. I hooked up the view to the view controller with the following code:

```
import UIKit

class ViewController: UIViewController {

    var myLayer = CATextLayer()
    @IBOutlet weak var myView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // setup the sublayer
        addSubLayer()
```

```
        // do the transform
        transformExample()
    }

    func addSubLayer() {
        myLayer.frame = CGRect(x: 0, y: 0, width: 100, height: 40)
        myLayer.backgroundColor = UIColor.blueColor().CGColor
        myLayer.string = "Hello"
        myView.layer.addSublayer(myLayer)
    }

    //******** Replace this function with the examples below ********

    func transformExample() {

        // add transform code here ...



    }

}
```

There are many different kinds of CALayer, but I chose to use CATextLayer so that the transforms will be more clear visually.

**Translate**

The translation transform moves the layer. The basic syntax is

```
CATransform3DMakeTranslation(tx: CGFloat, ty: CGFloat, tz: CGFloat)
```

where `tx` is the change in the x coordinates, `ty` is the change in y, and `tz` is the change in z.

**Example**



In iOS the origin of the coordinate system is in the top left, so if we wanted to move the layer 90 points to the right and 50 points down, we would do the following:

```
myLayer.transform = CATransform3DMakeTranslation(90, 50, 0)
```

Notes

- Remember that you can paste this into the `transformExample()` method in the project code above.
- Since we are just going to deal with two dimensions here, `tz` is set to 0.
- The red line in the image above goes from the center of the original location to the center of the new location. That's because transforms are done in relation to the anchor point and the anchor point by default is in the center of the layer.

**Scale**

The scale transform stretches or squishes the layer. The basic syntax is

```
CATransform3DMakeScale(sx: CGFloat, sy: CGFloat, sz: CGFloat)
```

where sx, sy, and sz are the numbers by which to scale (multiply) the x, y, and z coordinates respectively.

**Example**



If we wanted to half the width and triple the height, we would do the following

```
myLayer.transform = CATransform3DMakeScale(0.5, 3.0, 1.0)
```

Notes

- Since we are only working in two dimensions, we just multiply the z coordinates by 1.0 to leave them unaffected.
- The red dot in the image above represents the anchor point. Notice how the scaling is done in relation to the anchor point. That is, everything is either stretched toward or away from the anchor point.

**Rotate**

The rotation transform rotates the layer around the anchor point (the center of the layer by default). The basic syntax is

```
CATransform3DMakeRotation(angle: CGFloat, x: CGFloat, y: CGFloat, z: CGFloat)
```

where angle is the angle in radians that the layer should be rotated and x, y, and z are the axes about which to rotate. Setting an axis to 0 cancels a rotation around that particular axis.

**Example**

If we wanted to rotate a layer clockwise 30 degrees, we would do the following:

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)
myLayer.transform = CATransform3DMakeRotation(radians, 0.0, 0.0, 1.0)
```

Notes

- Since we are working in two dimentions, we only want the xy plane to be rotated around the z axis. Thus we set x and y to `0.0` and set z to `1.0`.
- This rotated the layer in a clockwise direction. We could have rotated counterclockwise by setting z to `-1.0`.
- The red dot shows where the anchor point is. The rotation is done around the anchor point.
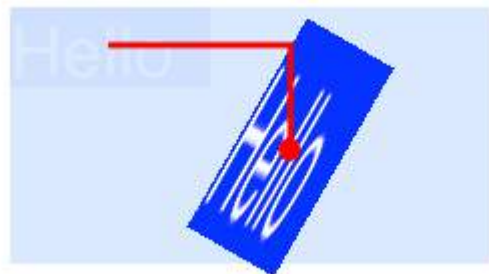
**Multiple transforms**

In order to combine multiple transforms we could use concatination like this

```
CATransform3DConcat(a: CATransform3D, b: CATransform3D)
```

However, we will just do one after another. The first transform will use the `Make` in its name. The following transforms will not use `Make`, but they will take the previous transform as a parameter.

**Example**



This time we combine all three of the previous transforms.

```
let degrees = 30.0
let radians = CGFloat(degrees * M_PI / 180)

// translate
var transform = CATransform3DMakeTranslation(90, 50, 0)

// rotate
transform = CATransform3DRotate(transform, radians, 0.0, 0.0, 1.0)
```

```
// scale
transform = CATransform3DScale(transform, 0.5, 3.0, 1.0)

// apply the transforms
myLayer.transform = transform
```

Notes

- The order that the transforms are done in matters.
- Everything was done in relation to the anchor point (red dot).

**A Note about Anchor Point and Position**

We did all our transforms above without changing the anchor point. Sometimes it is necessary to change it, though, like if you want to rotate around some other point besides the center. However, this can be a little tricky.

The anchor point and position are both at the same place. The anchor point is expressed as a unit of the layer's coordinate system (default is 0.5, 0.5) and the position is expressed in the superlayer's coordinate system. They can be set like this

```
myLayer.anchorPoint = CGPoint(x: 0.0, y: 1.0)
myLayer.position = CGPoint(x: 50, y: 50)
```

If you only set the anchor point without changing the position, then the frame changes so that the position will be in the right spot. Or more precisely, the frame is recalculated based on the new anchor point and old position. This usually gives unexpected results. The following two articles have an excellent discussion of this.

- [About the anchorPoint](#)
- [Translate rotate translate?](#)

**See also**

- [Border, rounded corners, and shadow on a CALayer](#)
- [Using a border with a Bezier path for a layer](#)

*This example originally comes from [this Stack Overflow example](#).*

# Section 89.2: Shadows

You can use 5 properties on each layer to configure your shadows:

- `shadowOffset` - this property moves your shadow left/right or up/down

```
self.layer.shadowOffset = CGSizeMake(-1, -1); // 1px left and up

self.layer.shadowOffset = CGSizeMake(1, 1); // 1px down and right
```

- `shadowColor` - this sets the color of your shadow

```
self.layer.shadowColor = [UIColor blackColor].CGColor;
```

- `shadowOpacity` - this is the opacity of the shadow, from 0 to 1

```
self.layer.shadowOpacity = 0.2;
```

- `shadowRadius` - this is the blur radius (equivalent of the blur property in Sketch or Photoshop)

```
self.layer.shadowRadius = 6;
```

- shadowPath - this is an important property for performance, when unset iOS bases the shadow on the alpha
  channel of the view, which can be performance intensive with a complex PNG with alpha. This property lets
  you force a shape for your shadow and be more performant because of it.

**Objective-C**

```
self.layer.shadowPath = [UIBezierPath bezierPathWithOvalInRect:CGRectMake(0,0,100,100)]; //this
does a circular shadow
```

**Swift 3**

```
self.layer.shadowPath = UIBezierPath(ovalIn: CGRect(x: 0, y: 0, width: 100, height: 100)).cgPath
```

# Section 89.3: Emitter View with custom image

For example we will create view that contains emitter layer and animates particles.

```swift
import QuartzCore

class ConfettiView: UIView {
    // main emitter layer
    var emitter: CAEmitterLayer!

    // array of color to emit
    var colors: [UIColor]!

    // intensity of appearance
    var intensity: Float!

    private var active :Bool!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setup()
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        setup()
    }

    func setup() {
        // initialization
        colors = [UIColor.redColor(),
                  UIColor.greenColor(),
                  UIColor.blueColor()
                  ]
        intensity = 0.2

        active = false
    }

    func startConfetti() {
        emitter = CAEmitterLayer()

        emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
        emitter.emitterShape = kCAEmitterLayerLine
```

```
        emitter.emitterSize = CGSize(width: frame.size.width, height: 1)

        var cells = [CAEmitterCell]()
        for color in colors {
            cells.append(confettiWithColor(color))
        }

        emitter.emitterCells = cells
        layer.addSublayer(emitter)
        active = true
    }

    func stopConfetti() {
        emitter?.birthRate = 0
        active = false
    }

    func confettiWithColor(color: UIColor) -> CAEmitterCell {
        let confetti = CAEmitterCell()

        confetti.birthRate = 10.0 * intensity
        confetti.lifetime = 180.0 * intensity
        confetti.lifetimeRange = 0
        confetti.color = color.CGColor
        confetti.velocity = CGFloat(350.0 * intensity)
        confetti.velocityRange = CGFloat(40.0 * intensity)
        confetti.emissionLongitude = CGFloat(M_PI)
        confetti.emissionRange = CGFloat(M_PI_4)
        confetti.spin = CGFloat(3.5 * intensity)
        confetti.spinRange = CGFloat(4.0 * intensity)

        // WARNING: A layer can set this property to a CGImageRef to display the image as its
contents.
        confetti.contents = UIImage(named: "confetti")?.CGImage
        return confetti
    }

    internal func isActive() -> Bool {
        return self.active
    }
}
}
```

You need to add "confetti" image or define rect with **confetti.contentsRect**

# Section 89.4: Rounded corners

```
layer.masksToBounds = true;
layer.cornerRadius = 8;
```

# Section 89.5: Creating particles with CAEmitterLayer

The **CAEmitterLayer** class provides a particle emitter system for Core Animation. The particles are defined by instances of **CAEmitterCell**.

The particles are drawn above the layer's background color and border.

```
        var emitter = CAEmitterLayer()

        emitter.emitterPosition = CGPoint(x: frame.size.width / 2.0, y: -20)
```

```
        emitter.emitterShape = kCAEmitterLayerLine
        emitter.emitterSize = CGSize(width: frame.size.width, height: 1)

        emitter.emitterCells = cells
        layer.addSublayer(emitter)
```

# Section 89.6: How to add a UIImage to a CALayer

You can add an image to a view's `layer` simply by using its `contents` property:

```
myView.layer.contents = UIImage(named: "star")?.CGImage
```

- Note that the `UIImage` needs to be converted to a `CGImage`.

If you wish to add the image in its own layer, you can do it like this:

```
let myLayer = CALayer()
let myImage = UIImage(named: "star")?.CGImage
myLayer.frame = myView.bounds
myLayer.contents = myImage
myView.layer.addSublayer(myLayer)
```

**Modifying the appearance**

The above code produces a view like this. The light blue is the `UIView` and the dark blue star is the `UIImage`.



As you can see, though, it looks pixelated. This is because the `UIImage` is smaller than the `UIView` so it is being scaled to fill the view, which is the default it you don't specify anything else.

The examples below show variations on the layer's `contentsGravity` property. The code looks like this:

```
myView.layer.contents = UIImage(named: "star")?.CGImage
myView.layer.contentsGravity = kCAGravityTop
myView.layer.geometryFlipped = true
```

In iOS, you may want to set the [geometryFlipped property](#) to `true` if you are doing anything with top or bottom gravity, otherwise it will be the opposite of what you expect. (Only the gravity is flipped vertically, not the content rendering. If you are having trouble with the content being flipped, see [this Stack Overflow answer](#).)

There are two `UIView` examples below for every `contentsGravity` setting, one view is larger than the `UIImage` and the other is smaller. This way you can see the effects of the scaling and gravity.

**kCAGravityResize**

This is the default.

**kCAGravityResizeAspect**



**kCAGravityResizeAspectFill**



**kCAGravityCenter**



**kCAGravityTop**

**kCAGravityBottom**



**kCAGravityLeft**



**kCAGravityRight**



**kCAGravityTopLeft**

**kCAGravityTopRight**

**kCAGravityBottomLeft**

**kCAGravityBottomRight**

**Related**

- [Content mode property of a view](#)
- [Drawing a `UIImage` in `drawRect` with `CGContextDrawImage`](#)
- [CALayer Tutorial: Getting Started](#)

**Notes**

- This example comes originally from [this Stack Overflow answer](#).

# Section 89.7: Disable Animations

`CALayer` property animations are enabled by default. When this is undesirable, they can be disabled as follows.

**Swift**

```swift
CATransaction.begin()
CATransaction.setDisableActions(true)

// change layer properties that you don't want to animate
```

```
CATransaction.commit()
```

**Objective-C**

```
[CATransaction begin];
[CATransaction setDisableActions:YES];

// change layer properties that you don't want to animate

[CATransaction commit];
```

# Chapter 90: iOS - Implementation of XMPP with Robbie Hanson framework

## Section 90.1: iOS XMPP Robbie Hanson Example with Openfire

**SRXMPPDemo**

**Download the example and all the classes here - https://github.com/SahebRoy92/SRXMPPDemo**

A demo on XMPP in Objective C, with various simple and complex features implemented in it. All the features of XMPP is done by **"in band"** xmpp functions. Few features this project contains are --

**SRXMPP** - A wrapper Singleton class that almost has all features needed for one-to-one chat application.

- one to one chat
- Core data implementation of chat (text message) thus having saving of previous messages, offline messages.
- implementation of vCard(profile information of user, own and others too) from XML and Core Data provided by Robbie Hanson's own framework.
- availability of friends status (online/offline/typing)

**Steps to follow**

You want to use this project as a reference then you can do the following--

**1. Installed Openfire in a live server** - Rent a server, install openfire.

**2. Want to try it out without a hassle in your own computer** - You need to download, install and setup 3 things to start

**a. Java -**

- Download and install Java for Mac.

**b. XAMPP** -

- Install XAMPP is relatively easy.
- After installation just start the XAMPP and start **Database(SQL)** and **Apache Server**.
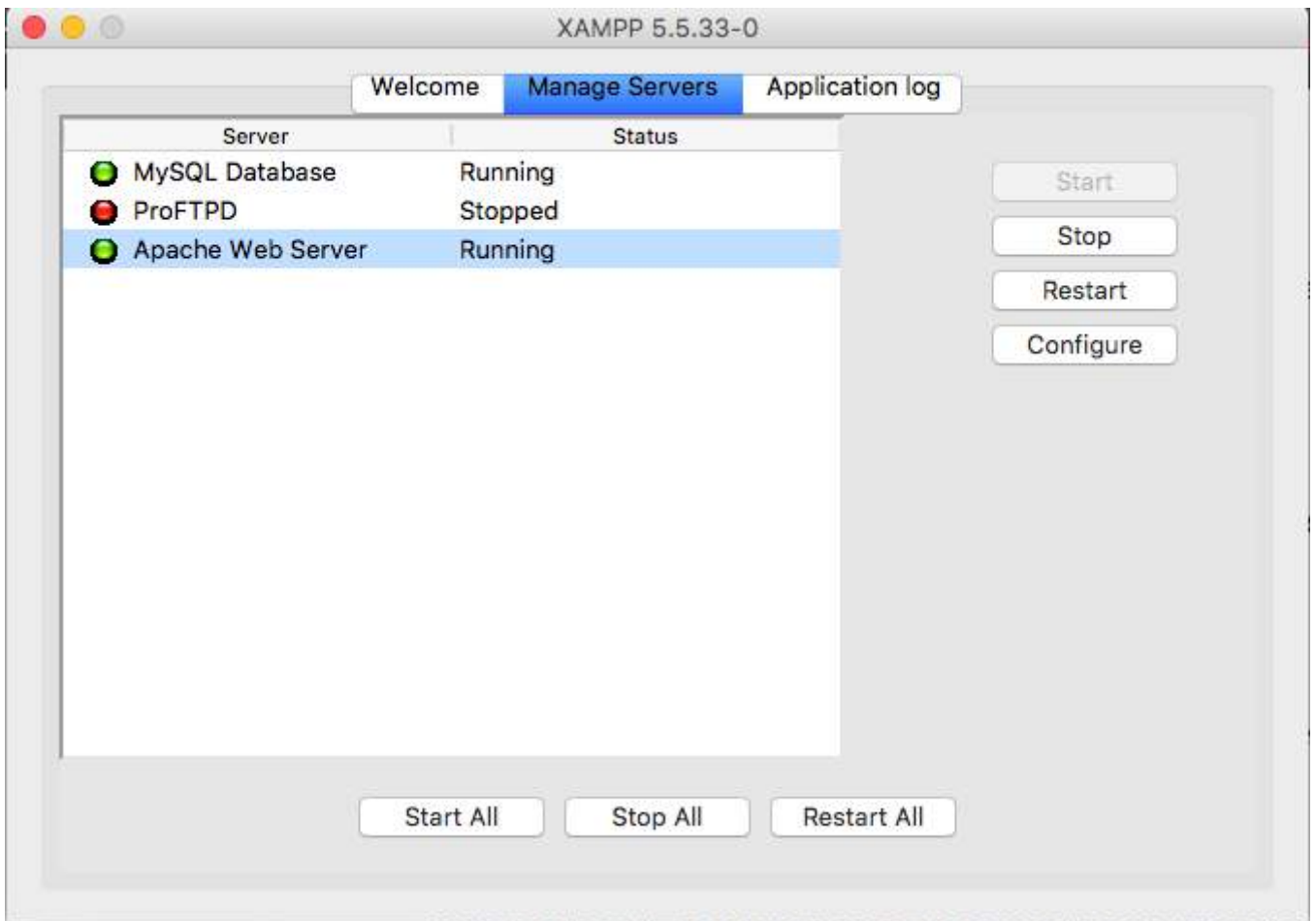
- Then open browser and paste this URL [http://localhost/phpmyadmin/]
- . Create a new DB from the left hand side panel.
- **Name the DB anything but remember this name, suppose we name it ChatDB**

**c. Openfire** -

- Install Openfire and run the application and "Start Openfire"

- Open Browser and Paste this URL - [http://localhost:9090/setup/index.jsp](http://localhost:9090/setup/index.jsp)
- Do normal setup
  - Select Language >
  - Server settings, leave as it is, just do continue >
  - Database Settings, leave as it is as "Standard Database Connection as selected >
  - Database Settings - Standard Connection". Now remember the name of the DB you set was **ChatDB**.
  - Select Database Driver Presets as *"*MySQL*"*. Leave JDBC Driver Class as it is. Now in the Database URL you can see, brackets mentioning hostname and Database Name. Just change Hostname to **"localhost"**, and database name to **"ChatDB"**, or any other name of DB you have set earlier, while seting up XAMPP. Leave the Username and password as blank.Fill up details like the image here



  .
  - Next complete setup by giving a username and password and reconfirming it. Thats it your done Setting up Openfire.

Now the part comes when you have to change a tiny detail in the code.

#**Important** We need to go to the class - **SRXMPP.m**, locate the NSString extern **SRXMPP_Hostname** (in the top) and overwrite the value of it to the

- IP of the server where OpenFire is installed , **OR**
- if you have installed it locally, overwrite the value to - **"localhost"**.

Thats it, you are ready to use this example project and start coding and making it into a better project of your own.

This starter pack will help you in understanding XMPP structure better as well as getting a grasp into XMPP protocols.

You can find other XMPP protocols here in this site - [https://xmpp.org/rfcs/rfc3920.html](https://xmpp.org/rfcs/rfc3920.html)

Development is still left and parts where I hope to include them later on

1. Group Chat
2. Image sending support

In short this example project along with the singleton has almost all features that are needed for a One-to-One chat application to have.

# Chapter 91: Swift and Objective-C interoperability

## Section 91.1: Using Objective-C Classes in Swift

> If you have an existing class that you'd like to use, perform **Step 2** and then skip to **Step 5**. (For some cases, I had to add an explicit `#import <Foundation/Foundation.h>` to an older ObjC File)

**Step 1: Add Objective-C Implementation -- .m**

Add a `.m` file to your class, and name it `CustomObject.m`

**Step 2: Add Bridging Header**

When adding your `.m` file, you'll likely be hit with a prompt that looks like this:
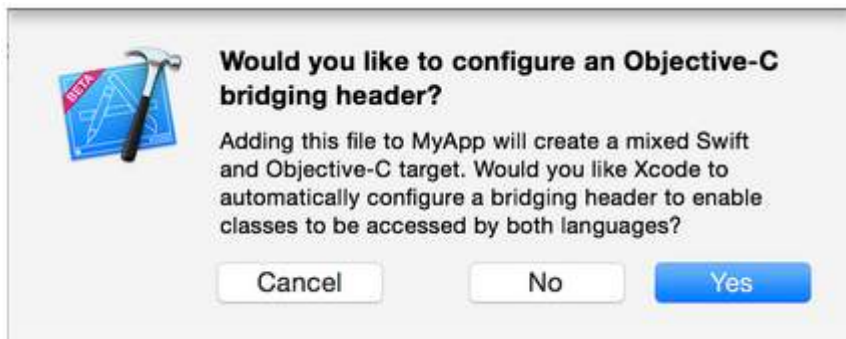


Click **YES** !

If you did not see the prompt, or accidentally deleted your bridging header, add a new `.h` file to your project and name it `<#YourProjectName#>-Bridging-Header.h`

In some situations, particularly when working with ObjC frameworks, you don't add an Objective-C class explicitly and Xcode can't find the linker. In this case, create your `.h` file named as mentioned above, then make sure you link its path in your target's project settings like so:
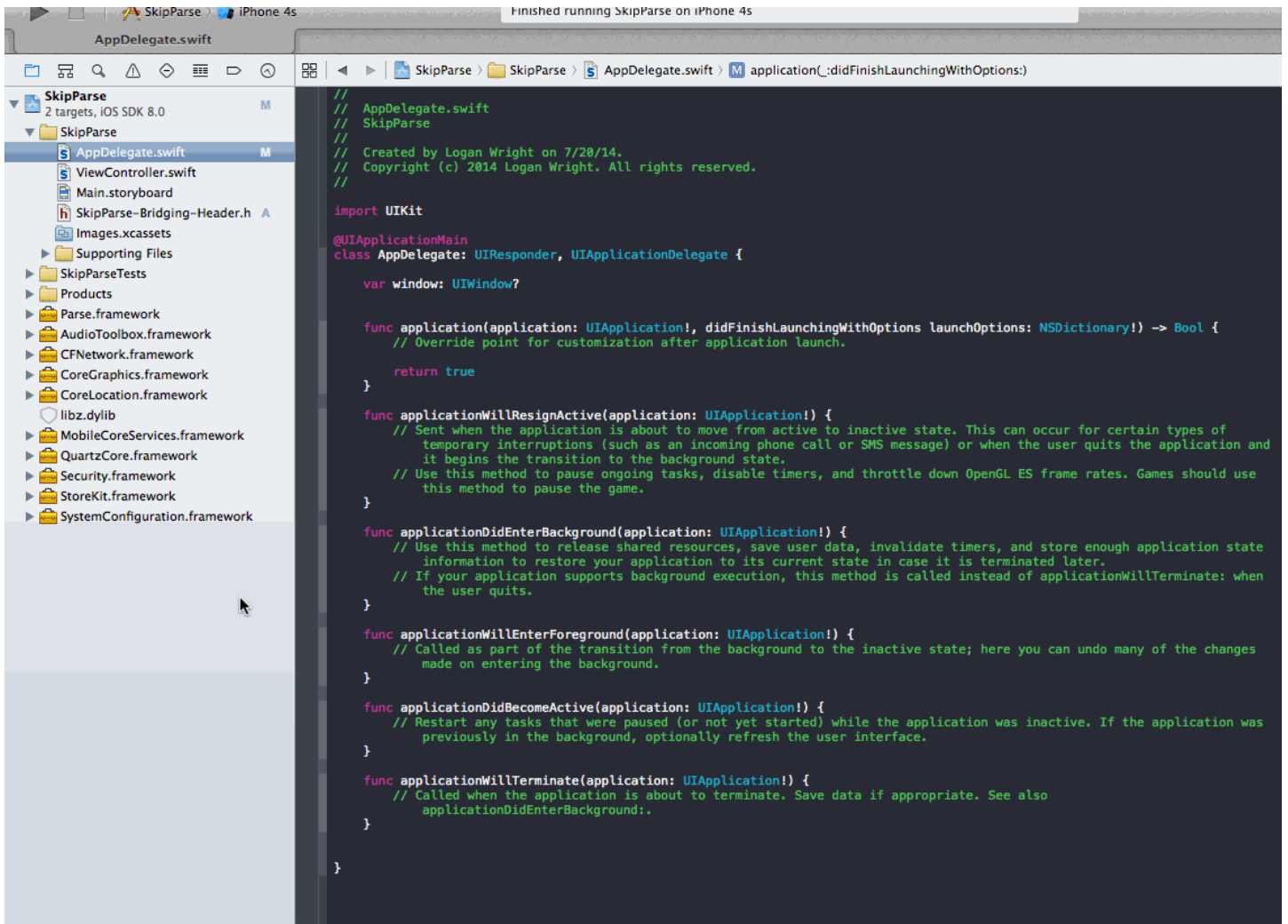
**Note**

It's best practice to link your project using the `$(SRCROOT)` macro so that if you move your project, or work on it with others using a remote repo, it will still work. `$(SRCROOT)` can be thought of as the directory that contains your .xcodeproj file. It might look like this:

```
$(SRCROOT)/Folder/Folder/<#YourProjectName#>-Bridging-Header.h
```

**Step 3: Add Objective-C Header -- .h**

Add another `.h` file and name it `CustomObject.h`

**Step 4: Build your Objective-C Class**

In `CustomObject.h`

```objc
#import <Foundation/Foundation.h>

@interface CustomObject : NSObject

@property (strong, nonatomic) id someProperty;

- (void) someMethod;

@end
```

In `CustomObject.m`

```
#import "CustomObject.h"

@implementation CustomObject

- (void) someMethod {
    NSLog(@"SomeMethod Ran");
}

@end
```

**Step 5: Add Class to Bridging-Header**

In YourProject-Bridging-Header.h:

```
#import "CustomObject.h"
```

**Step 6: Use your Object**

In SomeSwiftFile.swift:

```
var instanceOfCustomObject: CustomObject = CustomObject()
instanceOfCustomObject.someProperty = "Hello World"
println(instanceOfCustomObject.someProperty)
instanceOfCustomObject.someMethod()
```

No need to import explicitly, that's what the bridging header is for.

# Section 91.2: Using Swift Classes in Objective-C

**Step 1: Create New Swift Class**

Add a .swift file to your project, and name it MySwiftObject.swift

In MySwiftObject.swift:

```
import Foundation

class MySwiftObject : NSObject {

    var someProperty: AnyObject = "Some Initializer Val"

    init() {}

    func someFunction(someArg:AnyObject) -> String {
        var returnVal = "You sent me \(someArg)"
        return returnVal
    }

}
```

**Step 2: Import Swift Files to ObjC Class**

In SomeRandomClass.m:

```
#import "<#YourProjectName#>-Swift.h"
```

The file:<#YourProjectName#>-Swift.h should already be created automatically in your project, even if you can not

---

see it.

**Step 3: Use your class**

```
MySwiftObject * myOb = [MySwiftObject new];
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
myOb.someProperty = @"Hello World";
NSLog(@"MyOb.someProperty: %@", myOb.someProperty);
NSString * retString = [myOb someFunction:@"Arg"];
NSLog(@"RetString: %@", retString);
```

**Note:**

**1.** CodeCompletion wasn't behaving as accurately as I'd like it to. On my system, running a quick build w/ "cmd + r" seemed to help Swift find some of the Objc code and vice versa.

**2.** If you add `.swift` file to an older project and get error: `dyld: Library not loaded: @rpath/libswift_stdlib_core.dylib`, try completely [restarting Xcode.](#)

**3.** While it was originally possible to use pure Swift classes in Objective-C by using the `@objc` prefix, after Swift 2.0, this is no longer possible. See edit history for original explanation. If this functionality is reenabled in future Swift versions, the answer will be updated accordingly.

# Chapter 92: Custom fonts

## Section 92.1: Embedding custom fonts

> **Custom Font Support**
>
> Applications that want to use custom fonts can now include those fonts in their application bundle and register those fonts with the system by including the UIAppFonts key in their Info.plist file. The value of this key is an array of strings identifying the font files in the application's bundle. When the system sees the key, it loads the specified fonts and makes them available to the application.

Once the fonts have been set in the `Info.plist`, you can use your custom fonts as any other font in IB or programatically.

1. Drag and drop your font to Xcode Supporting Files folder. Don't forget to mark your app at "Add to targets" section. From this moment you can use this font in IB and choose it from font pallet.



2. To make this font available on the device, open `Info.plist` and add `Fonts provided by application key` (UIAppFonts). Add font name as the value to the Item 0 key. Note: Font name can vary from your font file name.



3. Get the custom added font name using below snippet

[*Swift 3*]

```swift
for family in UIFont.familyNames {
        print("\(family)")

        for name in UIFont.fontNames(forFamilyName: family) {
            print("    \(name)")
        }
    }
```

[*Objective - C*]

```objectivec
for (NSString *familyName in [UIFont familyNames]){
        NSLog(@"Family name: %@", familyName);
        for (NSString *fontName in [UIFont fontNamesForFamilyName:familyName]) {
            NSLog(@"--Font name: %@", fontName);
        }
    }
```

## Section 92.2: Applying custom fonts to controls within a Storyboard

The following example shows how to apply custom fonts to a Navigation Bar and includes fixes for some quirky behaviors found in Xcode. One also may apply the custom fonts to **any other UIControls** such as **UILabels**,

**UIButtons**, and more by using the attributes inspector after the custom font is added to the project. Please note the external links to working samples and videos near the bottom.

1. Select Your Navigation Bar within your Navigation Controller
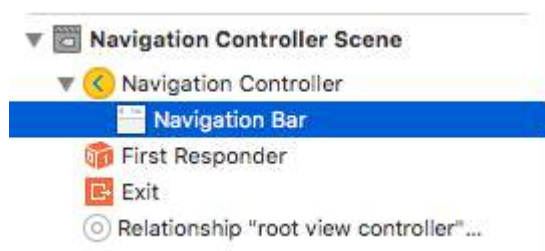


2. Change the Title Font in the Attributes Inspector
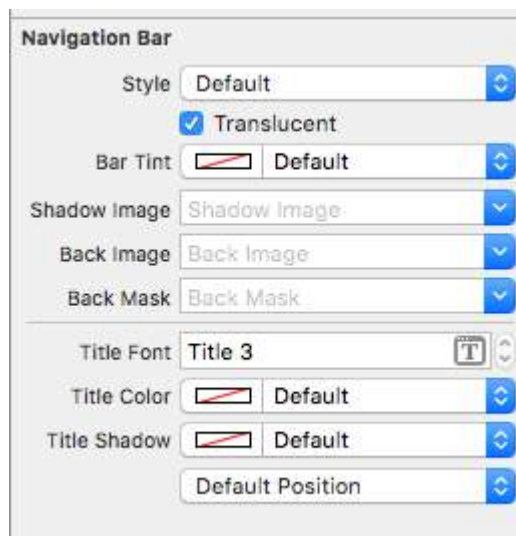


*(You will likely need to toggle the Bar Tint for the Navigation Bar before Xcode picks up the new font)*

**Notes (Caveats)**

Verified that this does work on Xcode 7.1.1+. (**See the Samples below**)

1. You do need to toggle the nav bar tint before the font takes effect (seems like a bug in Xcode; you can switch it back to default and font will stick)
2. If you choose a system font ~ Be sure to make sure the size is not 0.0 (Otherwise the new font will be ignored)



3. Seems like this works with no problem when only one NavBar is in the view hierarchy. It appears that secondary NavBars in the same stack are ignored. (Note that if you show the master navigation controller's navBar all the other custom navBar settings are ignored).

**Gotchas (deux)**

*Some of these are repeated which means they are very likely worth noting.*

1. Sometimes the storyboard xml gets corrupt. This requires you to review the structure in Storyboard as Source Code mode (right click the storyboard file > Open As ...)
2. In some cases the navigationItem tag associated with user defined runtime attribute was set as an xml child of the view tag instead of the view controller tag. If so remove it from between the tags for proper operation.
3. Toggle the NavBar Tint to ensure the custom font is used.
4. Verify the size parameter of the font unless using a dynamic font style
5. View hierarchy will override the settings. It appears that one font per stack is possible.

**Result**



**Samples**

- [Video Showing Multiple Fonts In Advanced Project](#)
- [Simple Source Download](#)
- [Advanced Project Download ~ Shows Multiple NavBar Fonts & Custom Font Workaround](#)
- [Video Showing Multiple Fonts & Custom Fonts](#)

**Handling Custom Fonts**

*Note ~ A [nice checklist](#) can be found from the Code With Chris website and you can see the sample download project.*

If you have your own font and want to use that in your storyboard, then there is a decent set of answers on the following [SO Question](#). One answer identifies these steps.

1. Get you custom font file(.ttf,.ttc)
2. Import the font files to your Xcode project
3. In the app-info.plist,add a key named Fonts provided by application.It's an array type , add all your font file names to the array,note:including the file extension.
4. In the storyboard , on the NavigationBar go to the Attribute Inspector,click the right icon button of the Font select area.In the popup panel , choose Font to Custom, and choose the Family of you embeded font name.

**Custom Font Workaround**

So Xcode naturally looks like it can handle custom fonts on UINavigationItem but that feature is just not updating properly (The font selected is ignored).
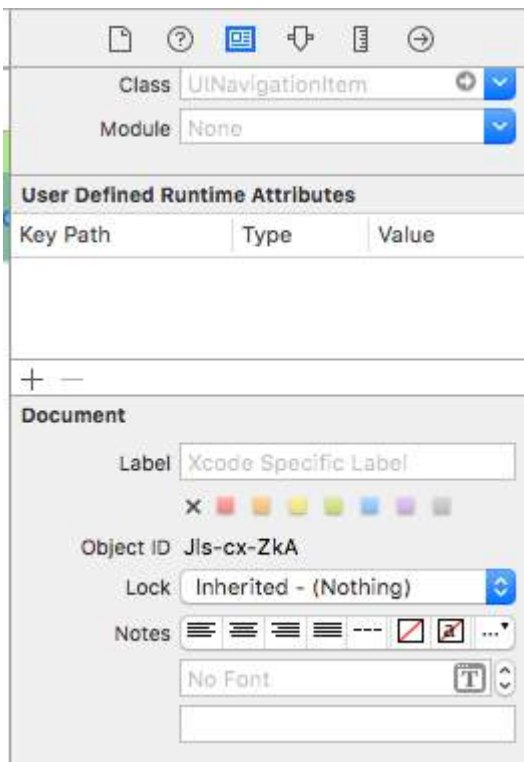
**To workaround this:**

One way is to fix using the storyboard and adding a line of code: First add a UIView (UIButton, UILabel, or some other UIView subclass) to the View Controller (Not the Navigation Item...Xcode is not currently allowing one to do that). After you add the control you can modify the font in the storyboard and add a reference as an outlet to your View Controller. Just assign that view to the UINavigationItem.titleView. You could also set the text name in code if necessary. Reported Bug (23600285).

```
@IBOutlet var customFontTitleView: UIButton!

//Sometime later...
self.navigationItem.titleView = customFontTitleView
```

Note - This example is derived from an answer I posted on SO (here).

# Section 92.3: Custom Fonts with Storyboard

Custom Fonts for UI components from storyboard can be easily achieved with User Defined Runtime Attributes in storyboard and Categories.

The advantages are like,

- No need to define outlets for the ui element
- No need to set font for elements programatically.

**Steps to follow**

1. **Font File:** Add the Font file (.ttf) to the application bundle and add the entry for the font in Info.plist under ***Font provided by application*** as in this documentation of custom fonts.

2. **Define Categories:** Add a file like ***UIKit+IBExtensions*** and add the categories for UI elements like

> UILabel, UIButton etc. for which you want to set custom font. All the categories will be having a custom property say **fontName**. This will be using from the storyboard later for setting custom font (as in step 4).

**UIKit+IBExtensions.h**

```
#import <UIKit/UIKit.h>

//Category extension for UILabel
@interface UILabel (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UITextField
@interface UITextField (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end

// Category extension for UIButton
@interface UIButton (IBExtensions)

@property (nonatomic, copy) NSString *fontName;
@end
```

3. **Getters and Setters:** Define getters and setters for the fontName property towards each category added.

**UIKit+IBExtensions.m**

```
#import "UIKit+IBExtensions.h"

@implementation UILabel (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UITextField (IBExtensions)

- (NSString *)fontName {
    return self.font.fontName;
}

- (void)setFontName:(NSString *)fontName {
    self.font = [UIFont fontWithName:fontName size:self.font.pointSize];
}
@end

@implementation UIButton (IBExtensions)

- (NSString *)fontName {
    return self.titleLabel.font.fontName;
```

```
}

- (void)setFontName:(NSString *)fontName{
    self.titleLabel.font = [UIFont fontWithName:fontName size:self.titleLabel.font.pointSize];
}
@end
```

> 4. **Setting font in storyboard:** Add an entry in User Defined Runtime Attributes with *fontName* as keyPath and your *Custom Font's Name* as value with type as String as shown.
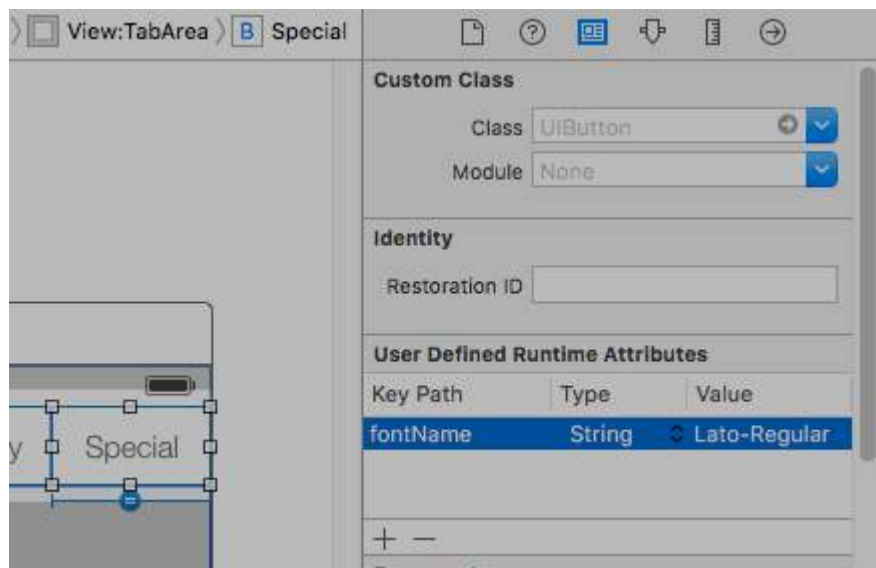


*This will set your custom font while running the app.*

Notes:

- Lato-Regular is the custom font I have used.
- Same name in the *.ttf* file added in bundle should be used without extension in storyboard.
- Font size will be same as it is defined in the UI element's attribute inspector.

# Chapter 93: AVSpeechSynthesizer

| Parameter | Details |
|-----------|---------|
| speaker | AVSpeechSynthesizer object |
| speech | AVSpeechUtterance object |

## Section 93.1: Creating a basic text to speech

Use the `speakUtterance:` method of `AVSpeechSynthesizer` to convert text to speech. You need to pass an `AVSpeechUtterance` object to this method, which contains the text that you want to be spoken.

**Objective C**

```objectivec
AVSpeechSynthesizer *speaker = [[AVSpeechSynthesizer alloc] init];
AVSpeechUtterance *speech   = [AVSpeechUtterance speechUtteranceWithString:@"Hello World"];
[speaker speakUtterance:speech];
```

**Swift**

```swift
let speaker = AVSpeechSynthesizer()
let speech = AVSpeechUtterance(string: "Hello World")
speaker.speakUtterance(speech)
```

# Chapter 94: Localization

**Localization** is feature provided by iOS which translates your app into multiple language.For **Localisation**,**Internationalization** is necessary.**Internationalization** is process of making iOS app able to adapt different culture,language and regions.

## Section 94.1: Localization in iOS

Create an individual `Localizable.strings` file for each language. The right side would be different for each language. Think of it as a key-value pair:

```
"str" = "str-language";
```

Access str in Objective-C:

```
//Try to provide description on the localized string to be able to create a proper documentation if needed
NSString *str = NSLocalizedString(@"string", @"description of the string");
```

Access str in Swift:

```
let str = NSLocalizedString("string", comment: "language");
```

# Chapter 95: Alamofire

| Parameter | Details |
|---|---|
| Method | `.OPTIONS`, `.GET`, `.HEAD`, `.POST`, `.PUT`, `.PATCH`, `.DELETE`, `.TRACE`, `.CONNECT` |
| URLString | `URLStringConvertible` |
| parameters | `[String: AnyObject]?` |
| encoding | `ParameterEncoding` |
| headers | `[String: String]?` |

## Section 95.1: Manual Validation

```swift
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .validate(statusCode: 200..<300)
    .validate(contentType: ["application/json"])
    .response { response in
        print(response)
    }
```

## Section 95.2: Automatic Validation

```swift
Alamofire.request("https://httpbin.org/get").validate().responseJSON { response in
switch response.result {
case .success:
    print("Validation Successful")
case .failure(let error):
    print(error)
 }
}
```

## Section 95.3: Chained Response Handlers

```swift
Alamofire.request(.GET, "https://httpbin.org/get")
    .validate()
    .responseString { response in
        print("Response String: \(response.result.value)")
    }
    .responseJSON { response in
        print("Response JSON: \(response.result.value)")
    }
```

## Section 95.4: Making a Request

```swift
import Alamofire

Alamofire.request(.GET, "https://httpbin.org/get")
```

## Section 95.5: Response Handling

```swift
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .responseJSON { response in
        print(response.request)   // original URL request
        print(response.response)  // URL response
        print(response.data)      // server data
        print(response.result)    // result of response serialization
```

```
        if let JSON = response.result.value {
            print("JSON: \(JSON)")
        }
    }
```

# Section 95.6: Response Handler

```
Alamofire.request(.GET, "https://httpbin.org/get", parameters: ["foo": "bar"])
    .validate()
    .response { request, response, data, error in
        print(request)
        print(response)
        print(data)
        print(error)
    }
```

# Chapter 96: iBeacon

| Parameters | Details |
|------------|---------|
| manager | CLLocationManager reference |
| region | CLRegion could be circular region (geofence or beacon region) |
| beacons | Array of CLBeacon contains all ranged beacons |

## Section 96.1: iBeacon Basic Operation

1. Setup monitoring beacons

```swift
func initiateRegion(ref:BeaconHandler){
    let uuid: NSUUID = NSUUID(UUIDString: "<UUID>")
    let beacon = CLBeaconRegion(proximityUUID: uuid, identifier: "")
    locationManager?.requestAlwaysAuthorization()    //cllocation manager obj.
    beacon?.notifyOnEntry = true
    beacon?.notifyOnExit = true
    beacon?.notifyEntryStateOnDisplay = true
    locationManager?.startMonitoringForRegion(beacon!)
    locationManager?.delegate = self;
    // Check if beacon monitoring is available for this device
    if (!CLLocationManager.isMonitoringAvailableForClass(CLBeaconRegion)) {
        print("error")
    }
    locationManager!.startRangingBeaconsInRegion(self.beacon!)
}
```

2. Location manager enter and exit region

```swift
func locationManager(manager: CLLocationManager, didEnterRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
        locationManager!.startRangingBeaconsInRegion(self.beacon!)
    }
}

func locationManager(manager: CLLocationManager, didExitRegion region: CLRegion) {
    if(region.isKindOfClass(CLBeaconRegion)) {
        locationManager!.stopRangingBeaconsInRegion(self.beacon!)
    }
}
```

3. Location manager range beacon

```swift
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    print(beacons.first.major)
}
```

## Section 96.2: Ranging iBeacons

First, you have to request authorization of location services

```swift
let locationManager = CLLocationManager()
locationManager.delegate = self
locationManager.requestWhenInUseAuthorization()
// OR locationManager.requestAlwaysAuthorization()
```

Then you can get all iBeacons' information inside `didRangeBeacons`

```swift
func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion
region: CLBeaconRegion) {
    for beacon in beacons {
        print(beacon.major)
        print(beacon.minor)
    }
}
```

## Section 96.3: Scanning specific Beacons

```swift
beacon = CLBeaconRegion(proximityUUID: <#NSUUID#>, major: <#CLBeaconMajorValue#>, identifier:
<#String#>) // listening to all beacons with given UUID and major value
beacon =     CLBeaconRegion(proximityUUID: <##NSUUID#>, major: <##CLBeaconMajorValue#>, minor:
<##CLBeaconMinorValue#>, identifier: <##String#>) // listening to all beacons with given UUID and
major and minor value
```

# Chapter 97: CLLocation

## Section 97.1: Distance Filter using

Example :

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.delegate = self;
locationManager.desiredAccuracy = kCLLocationAccuracyBest;
locationManager.distanceFilter = 5;
```

E.g. In the above example code above, location changes of less than 5 metres won't be sent to the callback, but instead be ignored.

## Section 97.2: Get User Location Using CLLocationManager

1 - Include the CoreLocation.framework in your project; this is accomplished by clicking on:

```
root directory -> build phases -> Link Binary With Libraries
```

Click on the (+) button, look for CoreLocation.framework and click add.

2- Modify the info.plist file to ask for permission to use user location by opening it as source code. Add either of the following key:value pair under the tag to ask for usage of user's location while the application is in use:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>message to display when asking for permission</string>
```

3- import CoreLocation to the ViewController that will be using it.

```
import CoreLocation
```

4- Make sure your ViewController conforms to the CLLocationManagerDelagate protocol

```
class ViewController: UIViewController,CLLocationManagerDelegate {}
```

After these steps, we can create a CLLocationManager object as instance variable and use it in the ViewController.

```
var manager:CLLocationManager!
```

We do not use 'let' here because we will modify the manager to specify its delegate, minimum distance before update event, and its accuracy

```
//initialize the manager
manager = CLLocationManager()

//specify delegate
manager.delegate = self

//set the minimum distance the phone needs to move before an update event is triggered (for
example:  100 meters)
manager.distanceFilter = 100

//set Accuracy to any of the following depending on your use case
```

```
//let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
//let kCLLocationAccuracyBest: CLLocationAccuracy
//let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
//let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
//let kCLLocationAccuracyKilometer: CLLocationAccuracy
//let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy

manager.desiredAccuracy = kCLLocationAccuracyBest

//ask the user for permission
manager.requestWhenInUseAuthorization()

//Start collecting location information
if #available(iOS 9.0, *) {

    manager.requestLocation()

 } else {

    manager.startUpdatingLocation()

  }
```

Now to get access to the location updates, we can implement the function below which is called overtime the distanceFilter is reached.

```
func locationManager(manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {}
```

The locations parameter is an array of CLLocation objects that represent the actual location of the device.From these objects, one can get access to the following attributes: `coordinate`, `altitude`, `floor`, `horizontalAccuracy`, `verticalAccuracy`, `timestamp`, `description`, `course`, `speed`, and a function `distance`(`from:`) that measures the distance between two locations.

Note: While requesting permission for location, there are two different types of authorization.

> "When In Use" authorization only gives the app permission to receive your location when the app is in use or foreground.
>
> "Always" authorization, gives the app background permissions which may lead to decrease battery life in case your app is closed.
>
> Plist file should be adjusted as necessary.

# Chapter 98: Checking iOS version

## Section 98.1: iOS 8 and later

**Swift 3:**

```swift
let minimumVersion = OperatingSystemVersion(majorVersion: 8, minorVersion: 1, patchVersion: 2)
if ProcessInfo().isOperatingSystemAtLeast(minimumVersion) {
    //current version is >= (8.1.2)
} else {
    //current version is < (8.1.2)
}
```

## Section 98.2: Swift 2.0 and later

```swift
if #available(iOS 9, *) {
  // iOS 9
} else {
  // iOS 8 or earlier
}
```

## Section 98.3: Compare versions

```swift
let minimumVersionString = "3.1.3"
let versionComparison = UIDevice.current.systemVersion.compare(minimumVersionString, options: .numeric)
switch versionComparison {
    case .orderedSame, .orderedDescending:
        //current version is >= (3.1.3)
        break
    case .orderedAscending:
        //current version is < (3.1.3)
        fallthrough
    default:
        break;
}
```

**Objective-C**

```objc
NSString *version = @"3.1.3";
NSString *currentVersion = @"3.1.1";
NSComparisonResult result = [currentVersion compare:version options:NSNumericSearch];
switch(result){
  case: NSOrderedAscending:
        //less than the current version
  break;
  case: NSOrderedDescending:
  case: NSOrderedSame:
        // equal or greater than the current version
  break;
}
```

## Section 98.4: Device iOS Version

This will give current system version.

**Objective-C**

```
NSString *version = [[UIDevice currentDevice] systemVersion]
```

**Swift**

```swift
let version = UIDevice.currentDevice().systemVersion
```

**Swift 3**

```swift
let version = UIDevice.current.systemVersion
```

# Chapter 99: Universal Links

## Section 99.1: Setup iOS Application (Enabling Universal Links)

**The setup on the app side requires two things:**

1. Configuring the app's entitlement, and enabling the universal links by turning on the Associated Domains capability in the project.
2. Handling Incoming Links in your `AppDelegate`.

**1. Configuring the app's entitlement, and enabling universal links.**

The first step in configuring your app's entitlements is to enable it for your App ID. Do this in the Apple Developer Member Center. Click on Certificates, Identifiers & Profiles and then Identifiers. Select your App ID (create it first if required), click Edit and enable the Associated Domains entitlement.
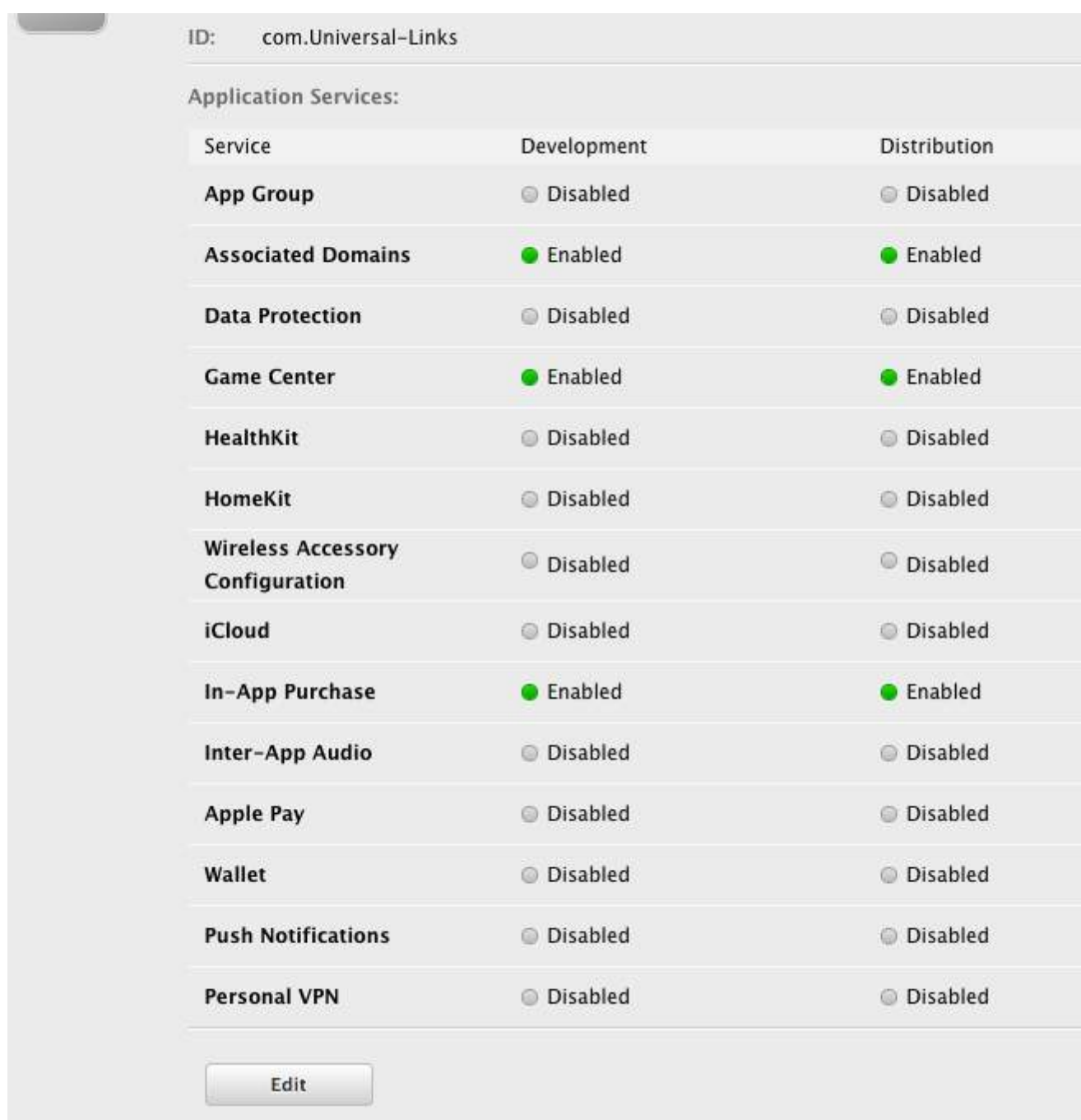
Next, get the App ID prefix and suffix by clicking on the respective App ID.

The App ID prefix and suffix should match the one in the apple-app-site-association file.

Next in Xcode, select your App's target, click Capabilities and toggle Associated Domains to On. Add an entry for each domain that your app supports, prefixed with **app links:**

For example **applinks:YourCustomDomainName.com**

Which looks like this for the sample app:
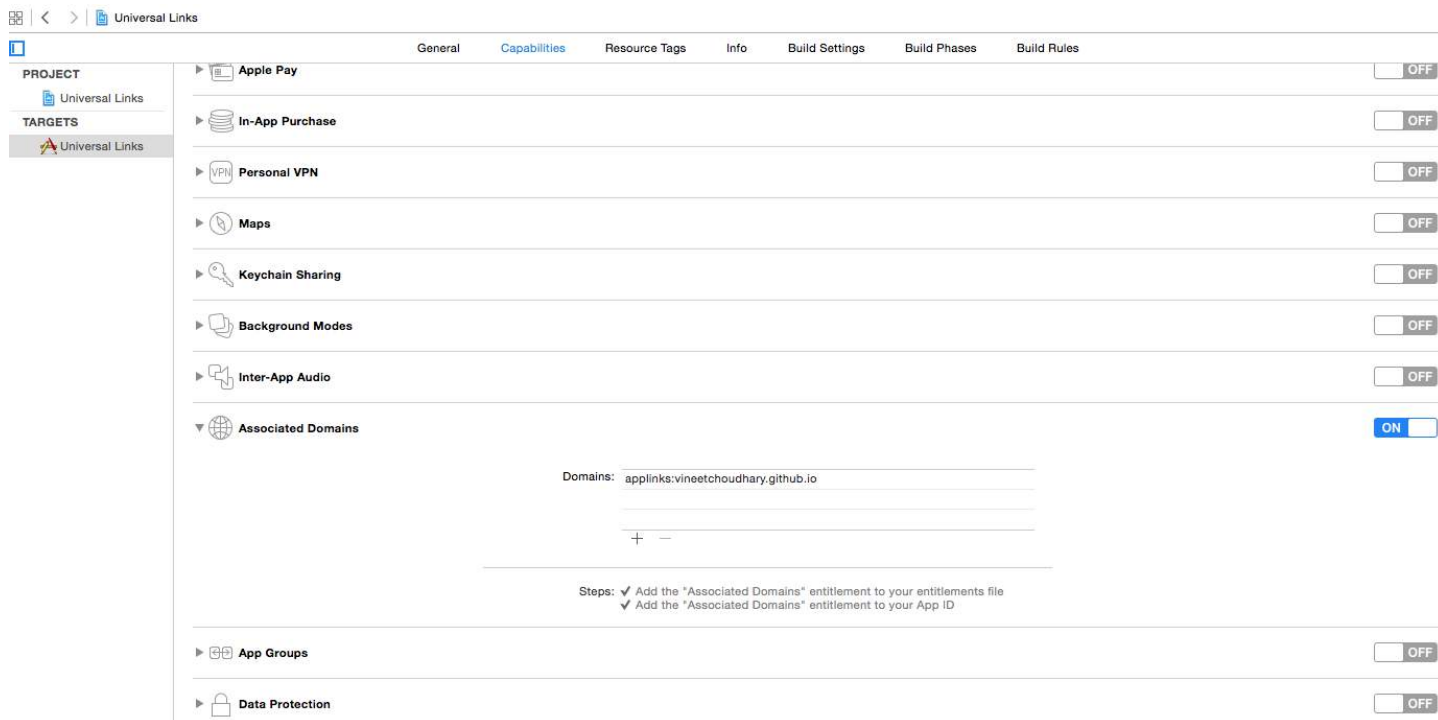


**Note**: Ensure you have selected the same team and entered the same Bundle ID as the registered App ID on the Member Center. Also ensure that the entitlements file is included by Xcode by selecting the file and in the File Inspector, ensure that your target is checked.

## Identity and Type

Name: Universal Links.entitlements

Type: Default - Entitlements Plist

Location: Relative to Group

Universal Links.entitlements

Full Path: /Users/emp195/Projects Backup/GitHub/Universal Links /Universal Links/ Universal Links/Universal Links.entitlements

## On Demand Resource Tags

Tags

## Localization

Localize...

## Target Membership

☑ ⚡ Universal Links

## Source Control

Repository: Universal Links

Type: Git

Current Branch: master

## 2. Handling Incoming Links in your AppDelegate

All redirects from Safari to the app for universal links go via the below method in the Application's AppDelegate class. You parse this URL to determine the right action in the app.

```
[UIApplicationDelegate application: continueUserActivity: restorationHandler:]
```

**Objective-C**

```objc
-(BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity *)userActivity
restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler{
    ///Checking whether the activity was from a web page redirect to the app.
    if ([userActivity.activityType isEqualToString: NSUserActivityTypeBrowsingWeb]) {
        ///Getting the URL from the UserActivty Object.
        NSURL *url = userActivity.webpageURL;
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:nil];
        UINavigationController *navigationController = (UINavigationController
*)_window.rootViewController;
        if ([url.pathComponents containsObject:@"home"]) {
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"HomeScreenId"] animated:YES];
        }else if ([url.pathComponents containsObject:@"about"]){
            [navigationController pushViewController:[storyBoard
instantiateViewControllerWithIdentifier:@"AboutScreenId"] animated:YES];
        }
    }
    return YES;
}
```

**Swift :**

```swift
func application(application: UIApplication, continueUserActivity userActivity: NSUserActivity,
```

```
restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
        let url = userActivity.webpageURL!
        //handle url
    }
    return true
}
```

**iOS Application Code**

The app code can be found master branch [here](here).

# Section 99.2: Supporting Multiple Domains

Each domain supported in the app needs to make available its own apple-app-site-association file. If the content served by each domain is different, then the contents of the file will also change to support the respective paths. Otherwise, the same file can be used, but it needs to be accessible at every supported domain.

# Section 99.3: Signing the App-Site-Association File

**Note**: *You can skip this part if your server uses `HTTPS` to serve content and jump to Application Setup guide.*

If your app targets iOS 9 and your server uses `HTTPS` to serve content, you don't need to sign the file. If not (e.g. when supporting Handoff on iOS 8), it has to be signed using a `SSL` certificate from a recognized certificate authority.

**Note**: This is not the certificate provided by Apple to submit your app to the App Store. It should be provided by a third-party, and it's recommended to use the same certificate you use for your `HTTPS` server (although it's not required).

To sign the file, first create and save a simple .txt version of it. Next, in the terminal, run the following command:

```
cat <unsigned_file>.txt | openssl smime -sign -inkey example.com.key -signer example.com.pem -
certfile intermediate.pem -noattr -nodetach -outform DER > apple-app-site-association
```

This will output the signed file in the current directory. The `example.com.key`, `example.com.pem`, and `intermediate.pem` are the files that would made available to you by your Certifying Authority.

**Note**: If the file is unsigned, it should have a `Content-Type` of `application/json`. Otherwise, it should be `application/pkcs7-mime`.

**Validate your Server with Apple App search validation tool**

Test your webpage for iOS 9 Search APIs. Enter a URL and Applebot will crawl your webpage and show how you can optimize for best results [https://search.developer.apple.com/appsearch-validation-tool/](https://search.developer.apple.com/appsearch-validation-tool/)

# Section 99.4: Setup Server

You need to having a server running online. To securely associate your iOS app with a server, Apple requires that you make available a configuration file, called `apple-app-site-association`. This is a `JSON` file which describes the domain and supported routes.

The `apple-app-site-association` file needs to be accessible via `HTTPS`, without any redirects, at *https://{domain}/apple-app-site-association*.

The file looks like this:

```
{
"applinks": {
    "apps": [ ],
    "details": [
        {
            "appID": "{app_prefix}.{app_identifier}",
            "paths": [ "/path/to/content", "/path/to/other/*", "NOT /path/to/exclude" ]
        },
        {
            "appID": "TeamID.BundleID2",
            "paths": [ "*" ]
        }
    ]
}
}
```

**NOTE** - *Don't append* `.json` *to the* `apple-app-site-association` *filename.*

**The keys are as follows:**

`apps`: Should have an empty array as its value, and it must be present. This is how Apple wants it.
`details`: Is an array of dictionaries, one for each iOS app supported by the website. Each dictionary contains information about the app, the team and bundle IDs.

**There are 3 ways to define paths:**

`Static`: The entire supported path is hardcoded to identify a specific link, e.g. /static/terms
`Wildcards`: A * can be used to match dynamic paths, e.g. /books/* can matches the path to any author's page. ? inside specific path components, e.g. books/1? can be used to match any books whose ID starts with 1.
`Exclusions`: Prepending a path with NOT excludes that path from being matched.

The order in which the paths are mentioned in the array is important. Earlier indices have higher priority. Once a path matches, the evaluation stops, and other paths ignored. Each path is case-sensitive.

#Website Code

The website code can be found gh-pages branch on
https://github.com/vineetchoudhary/iOS-Universal-Links/tree/gh-pages

# Chapter 100: PDF Creation in iOS

## Section 100.1: Create PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 612, 792), nil);

[self drawText];

UIGraphicsEndPDFContext();
```

**fileName** is the document file where You are going to append or attach

```
NSString* temporaryFile = @"firstIOS.PDF";
    NSArray *arrayPaths =
    NSSearchPathForDirectoriesInDomains(
                                  NSDocumentDirectory,
                                  NSUserDomainMask,
                                  YES);

    NSString *path = [arrayPaths objectAtIndex:0];

    NSString* fileName = [path stringByAppendingPathComponent:fileName];
```

Where **drawText** is

```
    (void)drawText
{
    NSString* textToDraw = @"Lorem Ipsum is simply dummy text of the printing and typesetting
industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an
unknown printer took a galley of type and scrambled it to make a type specimen book.";

    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);

    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);

    CGRect frameRect = CGRectMake(0, 0, 300, 100);

    CGMutablePathRef framePath = CGPathCreateMutable();

    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);

    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);

    CGContextRef currentContext = UIGraphicsGetCurrentContext();


    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);


    CGContextTranslateCTM(currentContext, 0, 450);

    CGContextScaleCTM(currentContext, 2, -2);
```

```
    CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);

    CFRelease(stringRef);

    CFRelease(framesetter);
}
```
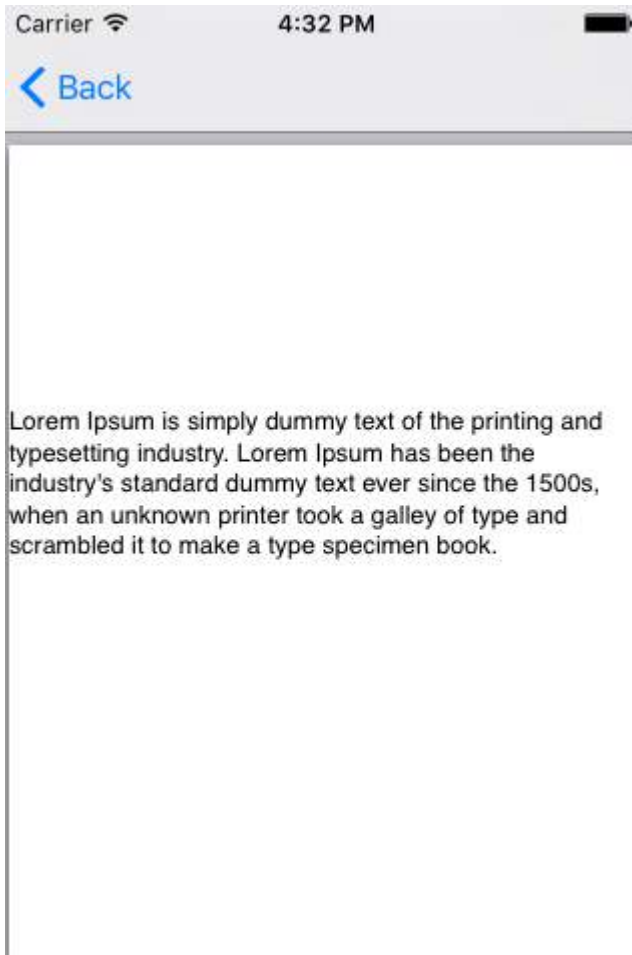


## Section 100.2: Show PDF

```
  NSString* fileName = @"firstIOS.PDF";


NSArray *arrayPaths =
NSSearchPathForDirectoriesInDomains(
                                    NSDocumentDirectory,
                                    NSUserDomainMask,
                                    YES);

NSString *path = [arrayPaths objectAtIndex:0];

NSString* pdfFileName = [path stringByAppendingPathComponent:fileName];

UIWebView* webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

NSURL *url = [NSURL fileURLWithPath:pdfFileName];

NSURLRequest *request = [NSURLRequest requestWithURL:url];
```

```
[webView setScalesPageToFit:YES];

[webView loadRequest:request];

[self.view addSubview:webView];
```

## Section 100.3: Multiple page PDF

```
UIGraphicsBeginPDFContextToFile(fileName, CGRectZero, nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsBeginPDFPageWithInfo(CGRectMake(0, 0, 600, 792), nil);

    UIGraphicsEndPDFContext();
```

## Section 100.4: Create PDF from any Microsoft Document loaded in UIWebview

```
#define kPaperSizeA4 CGSizeMake(595.2,841.8)
```

First of all implement UIPrintPageRenderer protocol

```
@interface UIPrintPageRenderer (PDF)

- (NSData*) printToPDF;

@end

@implementation UIPrintPageRenderer (PDF)

- (NSData*) printToPDF
{
    NSMutableData *pdfData = [NSMutableData data];
    UIGraphicsBeginPDFContextToData( pdfData, self.paperRect, nil );
    [self prepareForDrawingPages: NSMakeRange(0, self.numberOfPages)];
    CGRect bounds = UIGraphicsGetPDFContextBounds();
    for ( int i = 0 ; i < self.numberOfPages ; i++ )
    {
        UIGraphicsBeginPDFPage();
        [self drawPageAtIndex: i inRect: bounds];
    }
    UIGraphicsEndPDFContext();
    return pdfData;
}
@end
```

Then, call below method after document finished loading in UIWebView

```
-(void)createPDF:(UIWebView *)webView {

UIPrintPageRenderer *render = [[UIPrintPageRenderer alloc] init];
[render addPrintFormatter:webView.viewPrintFormatter startingAtPageAtIndex:0];

float padding = 10.0f;
CGRect paperRect = CGRectMake(0, 0, kPaperSizeA4.width, kPaperSizeA4.height);
```

```
CGRect printableRect = CGRectMake(padding, padding, kPaperSizeA4.width-(padding * 2),
kPaperSizeA4.height-(padding * 2));

[render setValue:[NSValue valueWithCGRect:paperRect] forKey:@"paperRect"];
[render setValue:[NSValue valueWithCGRect:printableRect] forKey:@"printableRect"];

NSData *pdfData = [render printToPDF];

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{

    if (pdfData) {
        [pdfData writeToFile:directoryPath atomically: YES];
    }
    else
    {
        NSLog(@"PDF couldnot be created");
    }
});;}
```

# Chapter 101: In-App Purchase

## Section 101.1: Single IAP in Swift 2

After creating an IAP in iTunesConnect:

In the view controller that you want to buy in

```
import StoreKit
```

and add the relevant delegates

```
class ViewController: UIViewController, SKProductsRequestDelegate, SKPaymentTransactionObserver {
```

declare a variable with the product id from iTunesConnect

```swift
var product_id: NSString?

override func viewDidLoad() {

    product_id = "YOUR_PRODUCT_ID"
    super.viewDidLoad()
    SKPaymentQueue.defaultQueue().addTransactionObserver(self)

    //Check if product is purchased
    if (NSUserDefaults.standardUserDefaults().boolForKey("purchased")){

        // Hide ads
        adView.hidden = true

    } else {
        print("Should show ads...")

    }

}
```

wire a button to a function to purchase the IAP

```swift
@IBAction func unlockAction(sender: AnyObject) {

    print("About to fetch the product...")

// Can make payments
if (SKPaymentQueue.canMakePayments())
    {
        let productID:NSSet = NSSet(object: self.product_id!);
        let productsRequest:SKProductsRequest = SKProductsRequest(productIdentifiers: productID as!
Set<NSString>);
        productsRequest.delegate = self;
        productsRequest.start();
        println("Fetching Products");
    }else{
        print("Can't make purchases");
    }
```

```
}
```

And here are some helper methods

```
func buyProduct(product: SKProduct){
    println("Sending the Payment Request to Apple");
    let payment = SKPayment(product: product)
    SKPaymentQueue.defaultQueue().addPayment(payment);

  }
```

the delegate methods that must be declared

```
func productsRequest (request: SKProductsRequest, didReceiveResponse response: SKProductsResponse)
{

    let count : Int = response.products.count
    if (count>0) {
        var validProduct: SKProduct = response.products[0] as SKProduct
        if (validProduct.productIdentifier == self.product_id) {
            print(validProduct.localizedTitle)
            print(validProduct.localizedDescription)
            print(validProduct.price)
            buyProduct(validProduct);
        } else {
            print(validProduct.productIdentifier)
        }
    } else {
        print("nothing")
    }
}


func request(request: SKRequest!, didFailWithError error: NSError!) {
    print("Error Fetching product information");
}

    func paymentQueue(_ queue: SKPaymentQueue,
updatedTransactions transactions: [SKPaymentTransaction])

{
    print("Received Payment Transaction Response from Apple");

    for transaction:AnyObject in transactions {
        if let trans:SKPaymentTransaction = transaction as? SKPaymentTransaction{
            switch trans.transactionState {
            case .Purchased:
                print("Product Purchased");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                // Handle the purchase
                NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                adView.hidden = true
                break;
            case .Failed:
                print("Purchased Failed");
                SKPaymentQueue.defaultQueue().finishTransaction(transaction as!
SKPaymentTransaction)
                break;
```

```
            case .Restored:
                print("Already Purchased");
                SKPaymentQueue.defaultQueue().restoreCompletedTransactions()


                // Handle the purchase
                    NSUserDefaults.standardUserDefaults().setBool(true , forKey: "purchased")
                    adView.hidden = true
                    break;
            default:
                break;
        }
    }
}

}
```

And then the code to restore a non-consumable in app purchase

```
if (SKPaymentQueue.canMakePayments()) {
  SKPaymentQueue.defaultQueue().restoreCompletedTransactions()
}
```

## Section 101.2: Most basic steps for purchasing/subscribing a user to an IAP

Assuming you know the `productID`:

First

```
import StoreKit
```

Then in your code

```
let productID: Set = ["premium"]
let request = SKProductsRequest(productIdentifiers: productID)
request.delegate = self
request.start()
```

and in the SKProductsRequestDelegate:

```
func productsRequest(request: SKProductsRequest, didReceiveResponse response: SKProductsResponse) {
    if response.products.count > 0 {
        let product = response.products[0]
        let payment = SKPayment(product: product)
        SKPaymentQueue.defaultQueue().addPayment(payment)
    }
}
```

## Section 101.3: Set Up in iTunesConnect

In iTunesConnect, select the app which you want to add an IAP to.

Click on features and you will see this:

Click the plus. You will then need to select which type of IAP you want to make.

Then you will need to fill out all of the information for your IAP.



If you have any trouble you can consult the IAP Set Up Guide.

# Chapter 102: CGContext Reference

## Section 102.1: Draw line

```
CGContextRef context = UIGraphicsGetCurrentContext();

CGContextSetLineWidth(context, 5.0);
CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();
CGContextMoveToPoint(context, 200, 400);
CGContextAddLineToPoint(context, 100, 100);
CGContextStrokePath(context);
CGColorSpaceRelease(colorspace);
```



## Section 102.2: Draw Text

Draw To requires **Core Text framework** to be added in the Build Phase

```
[NSString* textToDraw = @"Welcome to the world Of IOS";

    CFStringRef stringRef = (__bridge CFStringRef)textToDraw;

    CFAttributedStringRef currentText = CFAttributedStringCreate(NULL, stringRef, NULL);
    CTFramesetterRef framesetter = CTFramesetterCreateWithAttributedString(currentText);
    CGRect frameRect = CGRectMake(0, 0, 300, 100);
    CGMutablePathRef framePath = CGPathCreateMutable();
    CGPathAddRect(framePath, NULL, frameRect);

    CFRange currentRange = CFRangeMake(0, 0);
    CTFrameRef frameRef = CTFramesetterCreateFrame(framesetter, currentRange, framePath, NULL);
    CGPathRelease(framePath);
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    CGContextSetTextMatrix(currentContext, CGAffineTransformIdentity);
    CGContextTranslateCTM(currentContext, 200, 300);
    CGContextScaleCTM(currentContext, 2, -2);
    CTFrameDraw(frameRef, currentContext);

    CFRelease(frameRef);
    CFRelease(stringRef);
    CFRelease(framesetter);
```
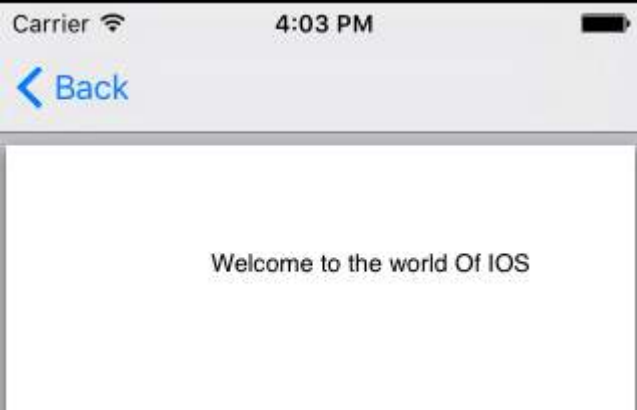
# Chapter 103: Core Location

## Section 103.1: Request Permission to Use Location Services

Check the app's authorization status with:

```swift
//Swift
let status: CLAuthorizationStatus = CLLocationManager.authorizationStatus()

//Objective-C
CLAuthorizationStatus status = [CLLocationManager authorizationStatus];
```

Test the status against the follow constants:

```swift
//Swift
switch status {
case .NotDetermined:
    // Do stuff
case .AuthorizedAlways:
    // Do stuff
case .AuthorizedWhenInUse:
    // Do stuff
case .Restricted:
    // Do stuff
case .Denied:
    // Do stuff
}

//Objective-C
switch (status) {
    case kCLAuthorizationStatusNotDetermined:

        //The user hasn't yet chosen whether your app can use location services or not.

        break;

    case kCLAuthorizationStatusAuthorizedAlways:

        //The user has let your app use location services all the time, even if the app is in the
background.

        break;

    case kCLAuthorizationStatusAuthorizedWhenInUse:

        //The user has let your app use location services only when the app is in the foreground.

        break;

    case kCLAuthorizationStatusRestricted:

        //The user can't choose whether or not your app can use location services or not, this
could be due to parental controls for example.

        break;

    case kCLAuthorizationStatusDenied:

        //The user has chosen to not let your app use location services.
```

```
            break;

    default:
        break;
}
```

**Getting Location Service Permission While App is in Use**
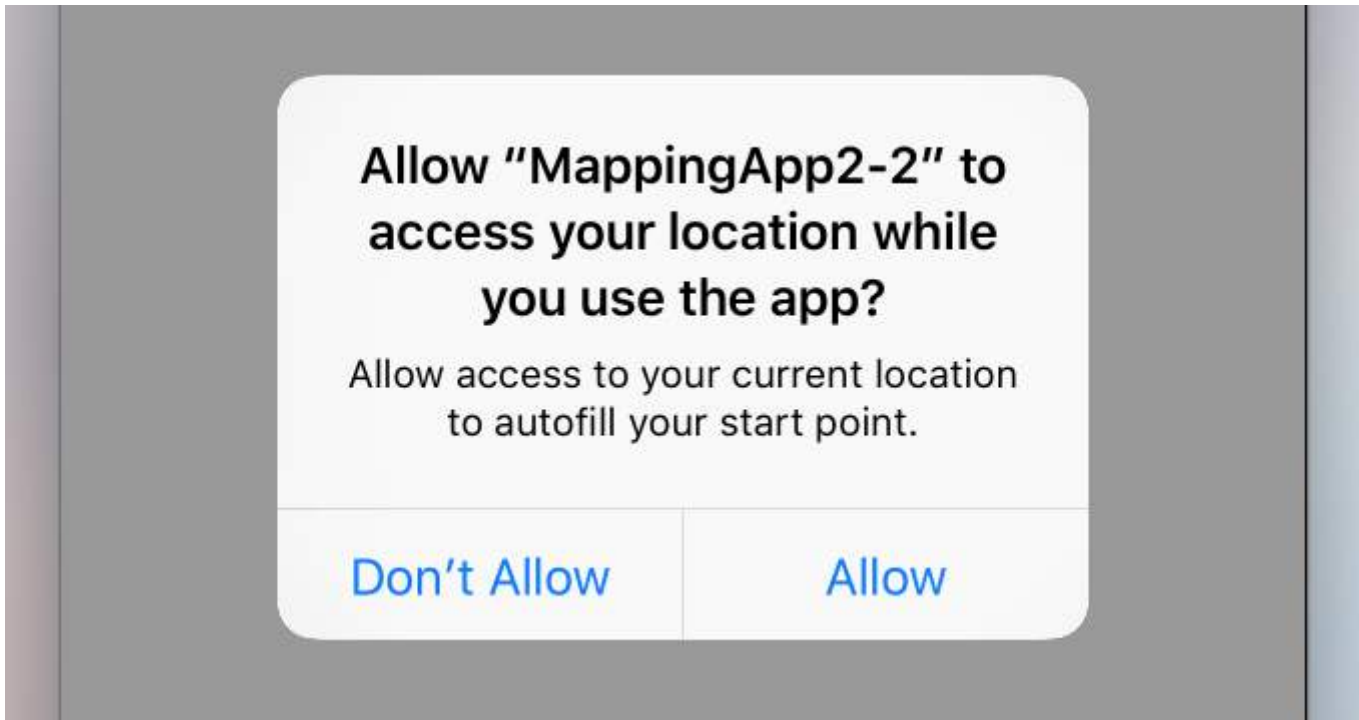


Simplest method is to initialize the location manager as a property of your root view controller and place the permission request in its `viewDidLoad`.

This brings up the alert controller that asks for permission:

```swift
//Swift
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

```objc
//Objective-C
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
[locationManager requestWhenInUseAuthorization];
```

Add the **NSLocationWhenInUseUsageDescription** key to your *Info.plist*. The value will be used in the alert controller's `message` label.

**Getting Location Service Permission Always**



To ask for permission to use location services even when the app is not active, use the following call instead:

```swift
//Swift
locationManager.requestAlwaysAuthorization()

//Objective-C
[locationManager requestAlwaysAuthorization];
```

Then add the **NSLocationAlwaysUsageDescription** key to your *Info.plist*. Again, the value will be used in the alert controller's `message` label.
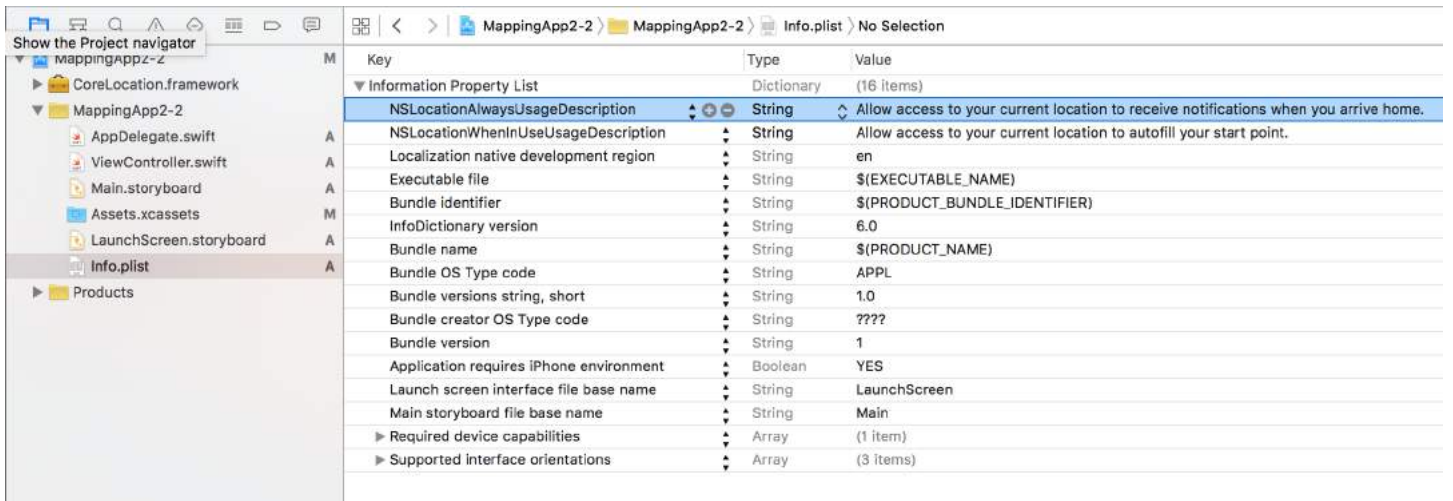
# Section 103.2: Add own custom location using GPX file

To check for location services we need real device but for testing purpose we can also use simulator and add our own location by following below steps:

- add new GPX file into your project.
- in GPX file add waypoints like

```xml
<?xml version="1.0"?>
<gpx version="1.1" creator="Xcode">
<!--
    Provide one or more waypoints containing a latitude/longitude pair. If you provide one
    waypoint, Xcode will simulate that specific location. If you provide multiple waypoints,
    Xcode will simulate a route visitng each waypoint.
-->
<wpt lat="52.599878" lon="4.702029">
    <name>location name (eg. Florida)</name>
</wpt>
```

- then go to product-->Scheme-->Edit Scheme and into RUN set default location as your GPX file name.

# Section 103.3: Link CoreLocation Framework
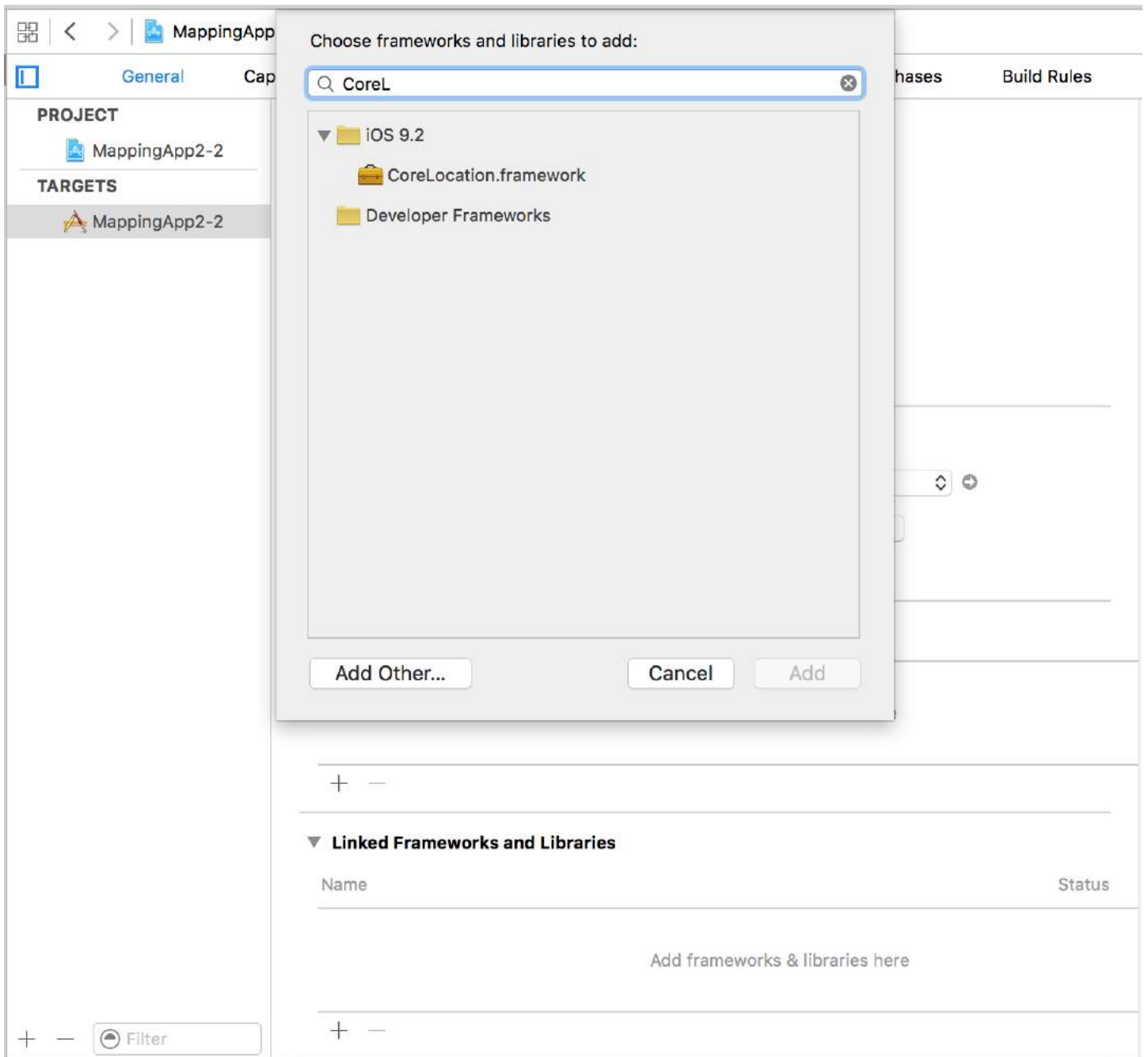
Import the CoreLocation module in your classes that use CoreLocation functionality.

```Swift
//Swift
import CoreLocation
```

```Objective-C
//Objective-C
#import <CoreLocation/CoreLocation.h>
```

# Section 103.4: Location Services in the Background

To use standard location services while the application is in the background you need first turn on `Background Modes` in the Capabilities tab of the Target settings, and select `Location updates`.


Or, add it directly to the Info.plist.

```
<key>NSLocationAlwaysUsageDescription</key>
<string>I want to get your location Information in background</string>
```

```
<key>UIBackgroundModes</key>
<array>
    <string>location</string>
</array>
```

Then you need to setup the CLLocationManager

**Objective C**

```
//The Location Manager must have a strong reference to it.
_locationManager = [[CLLocationManager alloc] init];
_locationManager.delegate = self;

//Request Always authorization (iOS8+)
if ([_locationManager respondsToSelector:@selector(requestAlwaysAuthorization)]) {
    [_locationManager requestAlwaysAuthorization];
}

//Allow location updates in the background (iOS9+)
if ([_locationManager respondsToSelector:@selector(allowsBackgroundLocationUpdates)]) {
    _locationManager.allowsBackgroundLocationUpdates = YES;
}

[_locationManager startUpdatingLocation];
```

**Swift**

```
self.locationManager.delegate = self

if #available (iOS 8.0,*) {
    self.locationManager.requestAlwaysAuthorization()
}

if #available (iOS 9.0,*) {
    self.locationManager.allowsBackgroundLocationUpdates = true
}

self.locationManager.startUpdatingLocation()
```

# Chapter 104: FacebookSDK

## Section 104.1: Creating your own custom "Sign In With Facebook" button

Sometimes we want to design our own UI for "Sign In With Facebook" button instead of the original button that comes with FacebookSDK.

1. In your storyboard, drag your UIButton and set it however you want it to be.
2. Ctrl + drag your button to your view controller as IBAction.
3. **Inside** the IBAction method you will have simulate a tap on the actual Facebook button as follow:

**Swift:**

```swift
let loginButton = FBSDKLoginButton()
loginButton.delegate = self
// Your Custom Permissions Array
loginButton.readPermissions =
[
                    "public_profile",
                    "email",
                    "user_about_me",
                    "user_photos"
]
// Hiding the button
loginButton.hidden = true
self.view.addSubview(loginButton)
// Simulating a tap for the actual Facebook SDK button
loginButton.sendActionsForControlEvents(UIControlEvents.TouchUpInside)
```

**Objective-C:**

```objc
FBSDKLoginButton *FBButton = [FBSDKLoginButton new];

// Your Custom Permissions Array
FBButton.readPermissions = @[@"public_profile",
                            @"email",
                            @"user_about_me",
                            @"user_photos"
                            ];
FBButton.loginBehavior = FBSDKLoginBehaviorNative;
[FBButton setDelegate:self];
[FBButton setHidden:true];
[loginButton addSubview:FBButton];

[FBButton sendActionsForControlEvents:UIControlEventTouchUpInside];
```

You're done.

## Section 104.2: FacebookSDK Integration

**Step 1: Install the SDK**

You can install the SDK manually or via `CocoaPods`. The latter option is highly recommended.

Put these lines in `Podfile`:

```
target 'MyApp' do
  use_frameworks!

  pod 'FBSDKCoreKit'
  pod 'FBSDKLoginKit'
  pod 'FBSDKShareKit'
end
```

Run pod `install` in the terminal and open `.xcworkspace` instead of `.xcodeproj` afterwards.

`FBSDKLoginKit` and `FBSDKShareKit` are optional. You may or may not need them.

**Step 2: Create an app on Facebook**

Go to [Quick Starts - Facebook for Developers](#) to create an app.

Facebook will ask you to download the SDK after creating the app. You can skip this part if you installed the SDK via CocoaPods already.

**Step 3: Edit `.plist`**

a. To make your app able to "communicate" with Facebook, you need to put some settings in your `.plist` file. Facebook will give you the customized snippet on the Quick Starts page.

b. Edit your `.plist` file as source code.
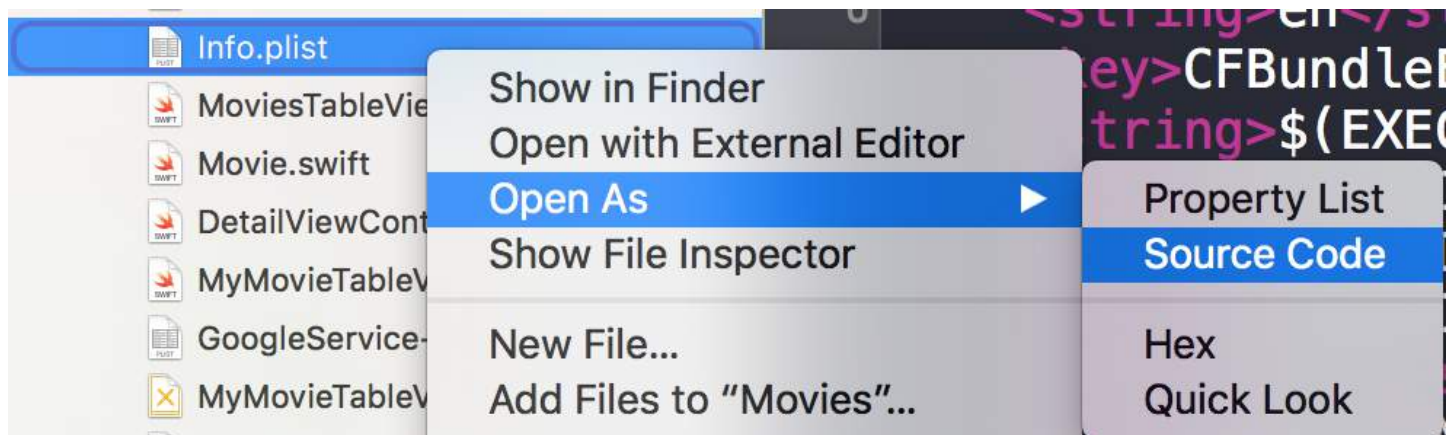


c. Paste your customized snippet in the source code. **Be careful!** The snippet must be exactly the child of the `<dict>` tag. Your source code should be something like:

```
<plist version="1.0">
<dict>
    // ...
    //some default settings
    // ...
    <key>CFBundleURLTypes</key>
    <array>
        <dict>
            <key>CFBundleURLSchemes</key>
            <array>
                <string>fb{FBAppId}</string>
            </array>
        </dict>
    </array>
    <key>FacebookAppID</key>
    <string>{FBAppId}</string>
```

```
    <key>FacebookDisplayName</key>
    <string>{FBAppName}</string>
    <key>LSApplicationQueriesSchemes</key>
    <array>
        <string>fbapi</string>
        <string>fb-messenger-api</string>
        <string>fbauth2</string>
        <string>fbshareextension</string>
    </array>
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>facebook.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>fbcdn.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
        <key>akamaihd.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <false/>
        </dict>
    </dict>
</dict>
</plist>
```

If you paste the snippet at a wrong place, you will run into problems.

**Step 4: Tell Facebook your bundle identifier on the Quick Starts page.**

=> How to get bundle identifier

**Step 5: Edit your `AppDelegate.swift`**

a.

```
import FBSDKCoreKit
```

b.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    FBSDKApplicationDelegate.sharedInstance().application(application,
didFinishLaunchingWithOptions: launchOptions)
    return true
}

func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
```

```
annotation: AnyObject) -> Bool {
    return FBSDKApplicationDelegate.sharedInstance().application(application, openURL: url,
sourceApplication: sourceApplication, annotation: annotation)
}
```

# Section 104.3: Fetching the facebook user data

After the user signed in to Facebook at your app, now it's time to fetch the data you requested at the
FBButton.readPermissions.

**Swift:**

```
enum FacebookParametesField : String
{
    case FIELDS_KEY = "fields"
    case FIELDS_VALUE = "id, email, picture, first_name, last_name"
}


if FBSDKAccessToken.currentAccessToken() != nil
{
//  Getting user facebook data
FBSDKGraphRequest(graphPath: "me",
                parameters: [FacebookParametesField.FIELDS_KEY.rawValue :
FacebookParametesField.FIELDS_VALUE.rawValue])
.startWithCompletionHandler({ (graphConnection : FBSDKGraphRequestConnection!, result : AnyObject!,
error : NSError!) -> Void in

        if error == nil
        {
            print("Facebook Graph phaze")

            let email = result["email"]
            let facebookToken = FBSDKAccessToken.currentAccessToken().tokenString
            let userFacebookId = result["id"]
            let firstName = result["first_name"]
            let lastName = result["last_name"]

            if let result = result as? Dictionary<String, AnyObject>
            {
                if let picture = result["picture"] as? Dictionary<String,AnyObject>
                {
                    if let data = picture["data"] as? Dictionary <String,AnyObject>
                    {
                        if let url = data["url"] as? String
                        {
                            //  Profile picture URL
                            let profilePictureURL = url
                        }
                    }
                }
            }
        }
    })
}
```

# Chapter 105: AFNetworking

## Section 105.1: Dispatching completion block on a custom thread

Whenever AFNetworking is used the call is dispatched on a custom thread provided by AFNetworking. When the call returns to the completion block, it gets executed on the main thread.

This example sets a custom thread that dispatch to the completion block:

**AFNetworking 2.xx:**

```
// Create dispatch_queue_t with your name and DISPATCH_QUEUE_SERIAL as for the flag
dispatch_queue_t myQueue = dispatch_queue_create("com.CompanyName.AppName.methodTest",
                 DISPATCH_QUEUE_SERIAL);

// init AFHTTPRequestOperation of AFNetworking
operation = [[AFHTTPRequestOperation alloc] initWithRequest:request];

// Set the FMDB property to run off the main thread
[operation setCompletionQueue:myQueue];
```

**AFNetworking 3.xx:**

```
AFHTTPSessionManager *manager = [[AFHTTPSessionManager alloc] init];
[self setCompletionQueue:myQueue];
```

# Chapter 106: CTCallCenter

## Section 106.1: CallKit - ios 10

```
//Header File

<CallKit/CXCallObserver.h>

CXCallObserver *callObserver = [[CXCallObserver alloc] init];

// If queue is nil, then callbacks will be performed on main queue

[callObserver setDelegate:self queue:nil];

// Don't forget to store reference to callObserver, to prevent it from being released

self.callObserver = callObserver;

// get call status
- (void)callObserver:(CXCallObserver *)callObserver callChanged:(CXCall *)call {
    if (call.hasConnected) {
        // perform necessary actions
    }
}
```

## Section 106.2: Intercepting calls from your app even from the background

From Apple documentation:

> Use the CTCallCenter class to obtain a list of current cellular calls, and to respond to state changes for calls such as from a dialing state to a connected state. Such state changes are known as cellular call events.

The purpose of CTCallCenter is to give the developer the opportunity to pause his app state during a call in order to give the user the best experience.

**Objective-C:**

First, we will define a new class member inside the class we want to handle the interceptions:

```
@property (atomic, strong) CTCallCenter *callCenter;
```

Inside our class init (constructor) we will allocate new memory for our class member:

```
[self setCallCenter:[CTCallCenter new]];
```

Afterwards, we will invoke our new method that actually handles the interceptions:

```
- (void)registerPhoneCallListener
{
[[self callCenter] setCallEventHandler:^(CTCall * _Nonnull call) {
    NSLog(@"CallEventHandler called - interception in progress");
```

```
        if ([call.callState isEqualToString: CTCallStateConnected])
        {
            NSLog(@"Connected");
        }
        else if ([call.callState isEqualToString: CTCallStateDialing])
        {
            NSLog(@"Dialing");
        }
        else if ([call.callState isEqualToString: CTCallStateDisconnected])
        {
            NSLog(@"Disconnected");

        } else if ([call.callState isEqualToString: CTCallStateIncoming])
        {
            NSLog(@"Incomming");
        }
  }];
 }
```

That's it, if the user will use your app and will receive a phone call you could intercept this call and handle your app for a save state.

It is worth mentioning that there are 4 call states you can intercept:

```
CTCallStateDialing
CTCallStateIncoming
CTCallStateConnected
CTCallStateDisconnected
```

**Swift:**

Define your class member at the relevant class and define it:

```
    self.callCenter = CTCallCenter()
    self.callCenter.callEventHandler = { call in
        //  Handle your interception
        if call.callState == CTCallStateConnected
        {
        }
    }
```

What will happen if your app is in the background and you need to intercept calls while the app is in the background ?

For example, if you develop an **enterprise** app you can basically just add 2 capabilities (VOIP & Background fetch) in the Capabilities tab:

Your project target -> Capabilities -> Background Modes -> mark Voice over IP & Background fetch

# Chapter 107: Push Notifications

| Parameter | Description |
|---|---|
| userInfo | A dictionary that contains remote notification info, potentially including a badge number for the app icon, alert sound, alert message, a notification identifier, and custom data. |

## Section 107.1: Registering device for Push Notifications

To register your device for push notifications, add the following code to your AppDelegate file in didFinishLaunchingWithOptions method:

**Swift**

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
    // Override point for customization after application launch.
    if UIDevice.currentDevice().systemVersion.compare(v, options: .NumericSearch) ==
NSOrderedAscending {
        // Register for Push Notitications, if running iOS < 8
        if application.respondsToSelector("registerUserNotificationSettings:") {
            let types:UIUserNotificationType = (.Alert | .Badge | .Sound)
            let settings:UIUserNotificationSettings = UIUserNotificationSettings(forTypes: types,
categories: nil)

            application.registerUserNotificationSettings(settings)
            application.registerForRemoteNotifications()
        } else {
            // Register for Push Notifications before iOS 8
            application.registerForRemoteNotificationTypes(.Alert | .Badge | .Sound)
        }
    } else {
        var center = UNUserNotificationCenter.currentNotificationCenter()
        center.delegate = self
        center.requestAuthorizationWithOptions((UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge)) {(granted: Bool, error: NSError) -> Void
in
            if !error {
                UIApplication.sharedApplication().registerForRemoteNotifications()
                // required to get the app to do anything at all about push notifications
                print("Push registration success.")
            } else {
                print("Push registration FAILED")
                print("ERROR: \(error.localizedFailureReason!) - \(error.localizedDescription)")
                print("SUGGESTIONS: \(error.localizedRecoveryOptions) -
\(error.localizedRecoverySuggestion!)")
            })
        }
    }

    return true
}
```

**Objective-C**

```
#define SYSTEM_VERSION_LESS_THAN(v) ([[[UIDevice currentDevice] systemVersion] compare:v
options:NSNumericSearch] == NSOrderedAscending)

if( SYSTEM_VERSION_LESS_THAN( @"10.0" ) )
{
        if ([application respondsToSelector:@selector(isRegisteredForRemoteNotifications)])
    {
        // iOS 8 Notifications
```

```objc
        [application registerUserNotificationSettings:[UIUserNotificationSettings
settingsForTypes:(UIUserNotificationTypeSound | UIUserNotificationTypeAlert |
UIUserNotificationTypeBadge) categories:nil]];

        [application registerForRemoteNotifications];
    }
    else
    {
        // iOS < 8 Notifications
        [application registerForRemoteNotificationTypes:
         (UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert |
UIRemoteNotificationTypeSound)];
    }

}
else
{

    UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
    center.delegate = self;
    [center requestAuthorizationWithOptions:(UNAuthorizationOptionSound |
UNAuthorizationOptionAlert | UNAuthorizationOptionBadge) completionHandler:^(BOOL granted, NSError
* _Nullable error)
    {
        if( !error )
        {
            [[UIApplication sharedApplication] registerForRemoteNotifications];  // required to
get the app to do anything at all about push notifications
            NSLog( @"Push registration success." );
        }
        else
        {
            NSLog( @"Push registration FAILED" );
            NSLog( @"ERROR: %@ - %@", error.localizedFailureReason, error.localizedDescription );
            NSLog( @"SUGGESTIONS: %@ - %@", error.localizedRecoveryOptions,
error.localizedRecoverySuggestion );
        }
    }];
}


//to check if your App lunch from Push notification
//-------------------------------------------------
    //Handel Push notification
    if (launchOptions != nil)
    {
        // Here app will open from pushnotification
        //RemoteNotification
        NSDictionary* dictionary1 = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];
        //LocalNotification
        NSDictionary* dictionary2 = [launchOptions
objectForKey:UIApplicationLaunchOptionsLocalNotificationKey];
        if (dictionary1 != nil)
        {
            //RemoteNotification Payload
            NSLog(@"Launched from push notification: %@", dictionary1);
            //here handle your push notification
        }
        if (dictionary2 != nil)
        {
            NSLog(@"Launched from dictionary2dictionary2dictionary2 notification: %@",
dictionary2);
```

```
                    double delayInSeconds = 7;
                    dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds *
NSEC_PER_SEC));
                    dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
                        // [self addMessageFromRemoteNotification:dictionary2 updateUI:NO];
                    });
                }


            }
            else
                {}
        //---------------------------------------------
```

The above code will try to communicate with APNs server to get device token (prerequiesites are you have APNs enabled in your iOS provisioning profile).

Once it establishes reliable connection with APNs server, the server provides you a device token.

After adding the code above, add these methods to the `AppDelegate` class:

**Swift**

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

**Objective-C**

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
NSString * deviceTokenString = [[[[deviceToken description]
                        stringByReplacingOccurrencesOfString: @"<" withString: @""]
                       stringByReplacingOccurrencesOfString: @">" withString: @""]
                      stringByReplacingOccurrencesOfString: @" " withString: @""];

NSLog(@"The generated device token string is : %@",deviceTokenString);
}

- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
{
    NSLog(@"Failed to get token, error: %@", error.description);
}
```

The above methods are called according to registration success or failure scenario.

Success scenario calls:

**Swift**

```
func applicaion(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    print("DEVICE TOKEN = \(deviceToken)")
}
```

In Swift3:

```
@objc(userNotificationCenter:willPresentNotification:withCompletionHandler:) @available(iOS 10.0,
*)
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification:
UNNotification, withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void)
{
    //To show notifications in foreground.
    print("Userinfo2 \(notification.request.content.userInfo)")
}
```

**Objective-C**

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
if(application.applicationState == UIApplicationStateInactive) {
    NSLog(@"Inactive - the user has tapped in the notification when app was closed or in
background");
    //do some tasks
    [self handelPushNotification:userInfo];
}
else if (application.applicationState == UIApplicationStateBackground) {
    NSLog(@"application Background - notification has arrived when app was in background");
    [self handelPushNotification:userInfo];
}
else {
    NSLog(@"application Active - notication has arrived while app was opened");
    //Show an in-app banner
    //do tasks
}
}
```

Failure scenario calls:

**Swift**

```
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print(error)
}
```

**Objective-C**

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error
```

> **Note**
>
> If none of the above methods are getting called, your device is not able to create reliable connection with
> APNs server, which might be because of internet access problems.

# Section 107.2: Registering App ID for use with Push Notifications

> **Things you need**
>
> - A paid Apple Developer Program Membership
> - A valid App ID and identifier for you app (like com.example.MyApp) which is not used before
>   anywhere

- Access to [developer.apple.com](developer.apple.com) and Member Center
- An iOS Device to test (as Push Notifications don't work on Simulator)

**Enabling the APNs access for App ID in Apple Developer Center**

1- Log in to developer.apple.com Member Center (the Account link on the home page)

Account

2- Go to "Certificates"

3- Select "App ID" from left panel

ID Identifiers

App IDs

4- Click on "+" on top right

+

5- Add App ID with Push Notifications option checked

6- Click on created App ID and select Edit

7- Click Configure in Push Notifications panel

8- Open Keychain Access app in your Mac

9- From Keychain Access menu, click Certificate Assistant -> Request a Certificate from a Certificate Authority

10- Enter your mail in the first text field

11- Enter your name in the second text field
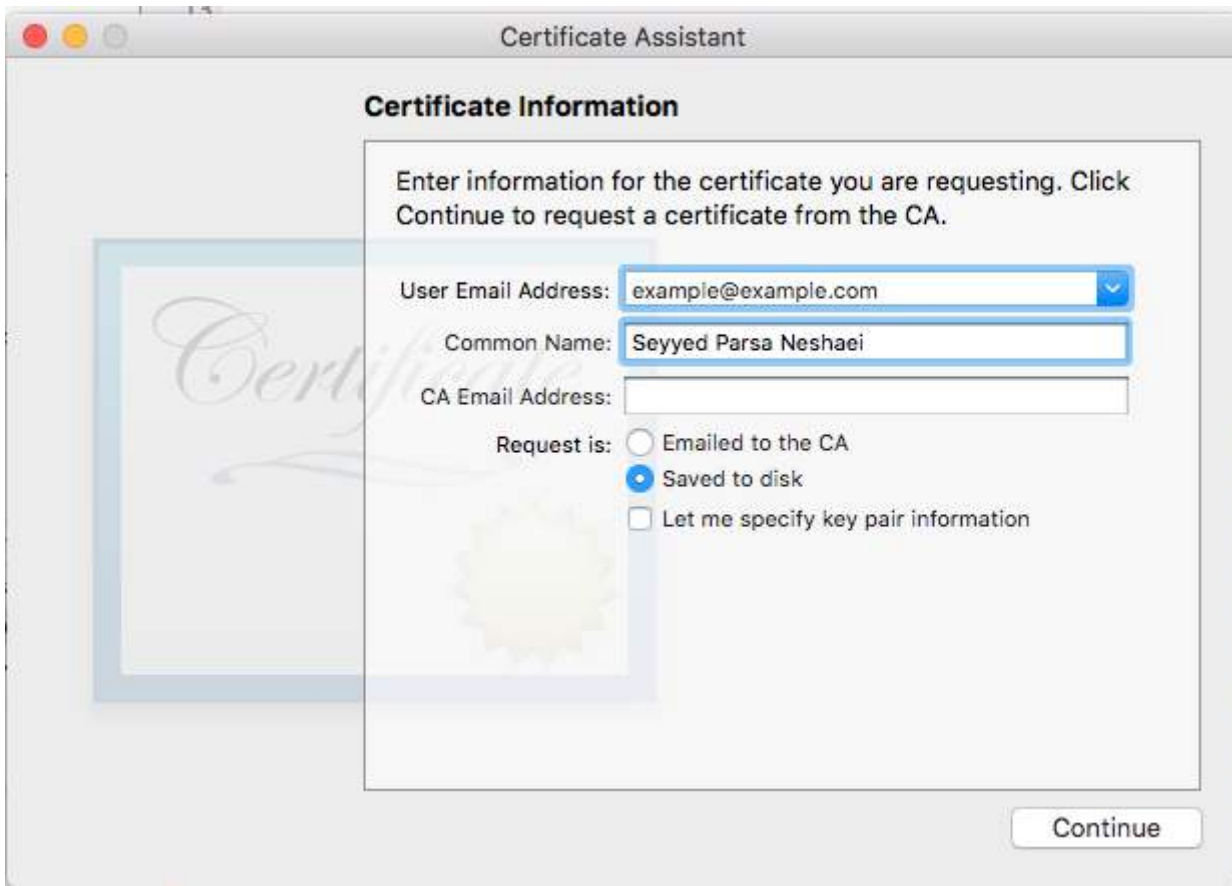
12- Leave CA Email Address empty

13- Select Saved to disk rather than Emailed to the CA

14- Click Continue and upload the generated file

15- Download the generated file by Apple and open it while Keychain Access is open

**Enabling the APNs access in Xcode**

1- Select your project

2- Open Capabilities tab

3- Find Push Notifications and turn it on

4-Find Background Modes and turn it on and check Remote Notifications

# Section 107.3: Testing push notifications

It is always a good practice to test how push notifications work even before you have your server side ready for them, just to make sure that everything is set up correctly on your side. It is quite easy to send yourself a push notification using a following PHP script.

1. Save the script as a file (send_push.php for example) in the same folder as your certificate (development or production)
2. Edit it to put your device token, password from the certificate
3. Choose the correct path for opening a connection, dev_path or prod_path (this is where 'Open a connection to the APNS server' happens in the script)
4. cd to the folder in Terminal and run command 'php send_push'

5. Receive the notification on your device

```php
<?php

// Put your device token here (without spaces):
$deviceToken = '20128697f872d7d39e48c4a61f50cb11d77789b39e6fc6b4cd7ec80582ed5229';
// Put your final pem cert name here. it is supposed to be in the same folder as this script
$cert_name = 'final_cert.pem';
// Put your private key's passphrase here:
$passphrase = '1234';

// sample point
$alert = 'Hello world!';
$event = 'new_incoming_message';

// You can choose either of the paths, depending on what kind of certificate you are using
$dev_path = 'ssl://gateway.sandbox.push.apple.com:2195';
$prod_path = 'ssl://gateway.push.apple.com:2195';

///////////////////////////////////////////////////////////////////////////////

$ctx = stream_context_create();
stream_context_set_option($ctx, 'ssl', 'local_cert', $cert_name);
stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

// Open a connection to the APNS server
$fp = stream_socket_client(
    $dev_path, $err,
    $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT, $ctx);

if (!$fp)
    exit("Failed to connect: $err $errstr" . PHP_EOL);

echo 'Connected to APNS' . PHP_EOL;

// Create the payload body
// it should be as short as possible
// if the notification doesnt get delivered that is most likely
// because the generated message is too long
$body['aps'] = array(
                    'alert' => $alert,
                    'sound' => 'default',
                    'event' => $event
                    );

// Encode the payload as JSON
$payload = json_encode($body);

// Build the binary notification
$msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n', strlen($payload)) . $payload;

// Send it to the server
$result = fwrite($fp, $msg, strlen($msg));

if (!$result)
    echo 'Message not delivered' . PHP_EOL;
else
    echo 'Message successfully delivered' . PHP_EOL;

// Close the connection to the server
fclose($fp);
```

# Section 107.4: Checking if your app is already registered for Push Notification

**Swift**

```
let isPushEnabled = UIApplication.sharedApplication().isRegisteredForRemoteNotifications()
```

# Section 107.5: Generating a .pem certificate from your .cer file, to pass on to the server developer

1. Save aps.cer to a folder

2. Open "Keychain access" and export the key that is under that certificate to a .p12 file (call it key.p12). To do that right click on it and choose Export. Save it to the same folder as step 1. On export you will be prompted for a password. Make something up and memorize it.
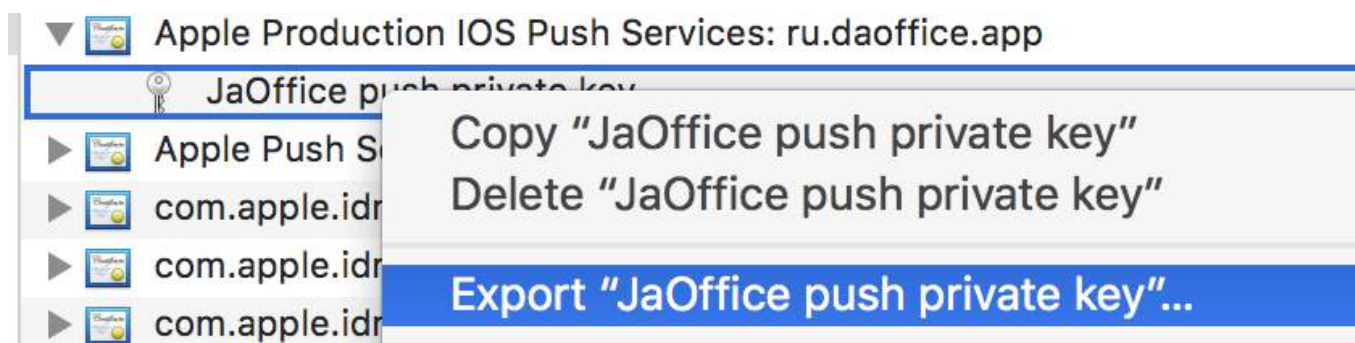


3. cd to that folder in Terminal and execute the following commands:

4. Convert .cer to a .pem certificate

```
openssl x509 -in aps.cer -inform der -out aps.pem
```

5. Convert your key to .pem format. To open the key, enter the password you exported it with from the keychain, in step 2. Then, enter another password that will protect the exported file. You will be prompted to enter it twice for confirmation.

```
openssl pkcs12 -nocerts -out key.pem -in key.p12
```

6. Merge the files into one final file

```
cat key.pem aps.pem > final_cert.pem
```

7. The final_cert.pem is the final result. Pass it on to server developers with the password from step 5, so that they will be able to use the protected certificate.

# Section 107.6: Unregistering From Push Notifications

To unregister from Remote Notifications programatically you can use

**Objective-C**

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

**Swift**

```
UIApplication.sharedApplication().unregisterForRemoteNotifications()
```

this is similar to going into the setting of your phone and manually switching off Notifications for the application.

**NOTE: There may be rare cases where you would need this(eg: when your app no longer supports push notifications)**

If you just want to allow the user to temporarily disable Notifications. You should implement a method to remove device token in the database on your server. else if you only disable Notification locally on your device your server will still send messages.

## Section 107.7: Setting the application icon badge number

Use the following piece of code to set the badge number from within your application (suppose `someNumber` has been declared before):

**Objective-C**

```
[UIApplication sharedApplication].applicationIconBadgeNumber = someNumber;
```

**Swift**

```
UIApplication.shared.applicationIconBadgeNumber = someNumber
```

In order to *remove* the badge completely, just set `someNumber = 0`.

## Section 107.8: Registering for (Non Interactive) Push Notification

The logic of registering for push notification is recommended to be added in `AppDelegate.swift` as the callback functions (success, failure) will be called their. To register just do the following:

```
let application = UIApplication.sharedApplication()
let settings = UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
application.registerUserNotificationSettings(settings)
```

Then the callback function `didRegisterUserNotificationSettings` will be called and in that case you just trigger the register like this:

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    application.registerForRemoteNotifications()
}
```

And in that case and system alert will be shown asking for persmission to receive push notification. One of the following callback functions will be called:

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    let tokenChars = UnsafePointer<CChar>(deviceToken.bytes)
    var tokenString = ""

    for i in 0..<deviceToken.length {
        tokenString += String(format: "%02.2hhx", arguments: [tokenChars[i]])
    }

    print("Push token: \(tokenString)")
}
```

```swift
func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    print("didFailToRegisterForRemoteNotificationsWithError: \(error)")

}
```

In very rare cases, neither success or failure callback functions are called. This happens when you have internet connection problems or the APNS Sandbox is down. The system do an API call to APNS to do some verification, failing to do so will lead to none of the two callbacks functions will be called. Visit Apple system status to make sure its fine.

# Section 107.9: Handling Push Notification

Once user clicks on a push notification, the following callback function will be called. You can parse the JSON to get any specific info sent from backend that will helps you in deep linking:

Swift

```swift
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    print("Received notification: \(userInfo)")
}
```

Objective C

```objc
- (void)application:(UIApplication *)application didReceiveRemoteNotification: (NSDictionary
*)userInfo
{
    NSLog(@"Received notification: %@", userInfo);
}
```

iOS 10

```objc
#define SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO(v)  ([[[UIDevice currentDevice] systemVersion]
compare:v options:NSNumericSearch] != NSOrderedAscending)

-(void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo fetchCompletionHandler:(void
 (^)(UIBackgroundFetchResult))completionHandler
{
    // iOS 10 will handle notifications through other methods
    NSLog(@"Received notification: %@", userInfo);


    if( SYSTEM_VERSION_GREATER_THAN_OR_EQUAL_TO( @"10.0" ) )
    {
        NSLog( @"iOS version >= 10. Let NotificationCenter handle this one." );
        // set a member variable to tell the new delegate that this is background
        return;
    }
    NSLog( @"HANDLE PUSH, didReceiveRemoteNotification: %@", userInfo );

    // custom code to handle notification content
```

```objc
      if( [UIApplication sharedApplication].applicationState == UIApplicationStateInactive )
      {
          NSLog( @"INACTIVE" );
          completionHandler( UIBackgroundFetchResultNewData );
      }
      else if( [UIApplication sharedApplication].applicationState == UIApplicationStateBackground )
      {
          NSLog( @"BACKGROUND" );
          completionHandler( UIBackgroundFetchResultNewData );
      }
      else
      {
          NSLog( @"FOREGROUND" );
          completionHandler( UIBackgroundFetchResultNewData );
      }
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
  willPresentNotification:(UNNotification *)notification
    withCompletionHandler:(void (^)(UNNotificationPresentationOptions options))completionHandler
{
    NSLog( @"Handle push from foreground" );
   // custom code to handle push while app is in the foreground
    NSLog(@"%@", notification.request.content.userInfo);
}

- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
     withCompletionHandler:(void (^)())completionHandler
{

    NSLog( @"Handle push from background or closed" );
    // if you set a member variable in didReceiveRemoteNotification, you  will know if this is from
closed or background
    NSLog(@"%@", response.notification.request.content.userInfo);
}
```

# Chapter 108: Extension for rich Push Notification - iOS 10.

iOS 10 gave us `UserNotifications.framework`, the new API for local/remote notifications. It offers viewing media attachments or responding to messages right from the notification.

Notification content consists of: title, subtitle, body and attachment. Attachment can contain images/gifs/videos up to 50 mb.

## Section 108.1: Notification Content Extension

**Why do we need it?**

Content extension helps us to create custom user interface upon notification expanasion.

You use this framework to define an extension that receives the notification data and provides the corresponding visual representation. Your extension can also respond to custom actions associated with those notifications.

## Section 108.2: Implementation

1. In xCode `Navigator` window go to `Targets` section. Press `Add New Target`.
2. Select `Notification Content Extension` template:
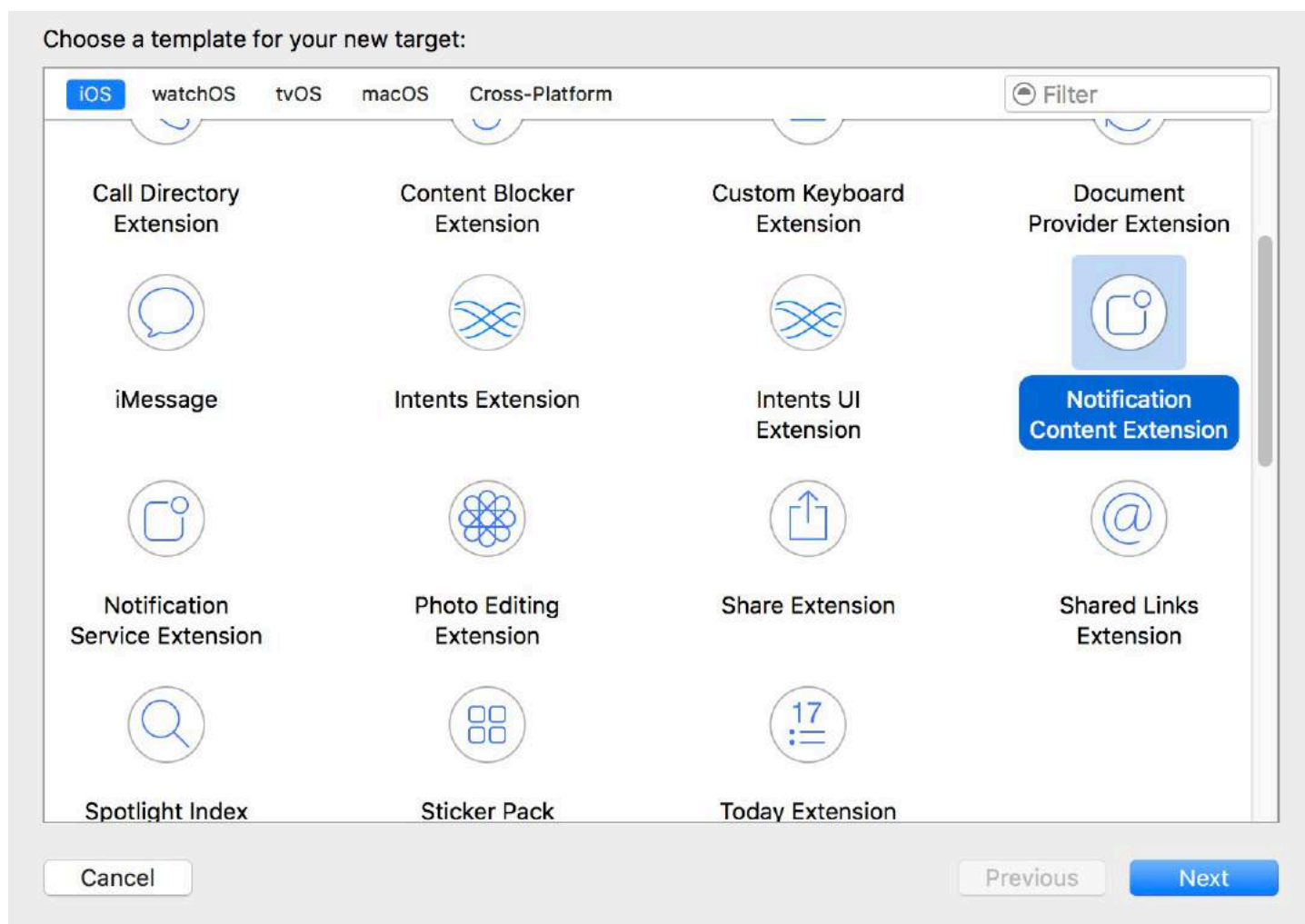


3. In your `info.plist` file set the identifier for `UNNotificationExtensionCategory` key:

| ▼ NSExtension | | Dictionary | (3 items) |
|---|---|---|---|
| ▼ NSExtensionAttributes | | Dictionary | (3 items) |
| | UNNotificationExtensionDefaultContentHidden | Boolean | YES |
| | UNNotificationExtensionCategory | String | customContentIdentifier |
| | UNNotificationExtensionInitialContentSizeRatio | Number | 0.7 |
| NSExtensionMainStoryboard | | String | MainInterface |
| NSExtensionPointIdentifier | | String | com.apple.usernotifications.content-extension |

*NSExtensionAttributes*:

`UNNotificationExtensionCategory` (Required)

The value of this key is a string or an array of strings. Each string contains the identifier of a category declared by the app using the UNNotification?Category class.

`UNNotificationExtensionInitialContentSizeRatio` (Required)

Number that represents the initial size of your view controller's view expressed as a ratio of its height to its width.

`UNNotificationExtensionDefaultContentHidden` (Optional)

When set to YES, the system displays only your custom view controller in the notification interface. When set to NO, the system displays the default notification content in addition to your view controller's content.

`UNNotificationExtensionOverridesDefaultTitle` (Optional)

The value of this key is a Boolean. When set to true, the system uses the title property of your view controller as the title of the notification. When set to false, the system sets the notification's title to the name of your app. If you do not specify this key, the default value is set to false.

4. Create custom view in `NotificationViewController.``swift` file
5. Add new `category key` and set its value to what we typed in the Info.plist (step 3):

Push:

```
{
 aps: {
 alert: { … },
 category: 'io.swifting.notification-category'
 }
}
```

Local:

```
let mutableNotificationContent = UNMutableNotificationContent()
mutableNotificationContent.category = "io.swifting.notification-category"
mutableNotificationContent.title = "Swifting.io Notifications"
mutableNotificationContent.subtitle = "Swifting.io presents"
mutableNotificationContent.body = "Custom notifications"
```

*Also check out the official API reference:*

https://developer.apple.com/reference/usernotificationsui/unnotificationcontentextension?utm_source=swifting.io&utm_medium=web&utm_campaign=blog%20post

# Chapter 109: Rich Notifications

The Rich Notifications lets you customize the appearance of local and remote notifications when they appear on the user's device.Rich notification mostly includes UNNotificationServiceExtension and UNNotificationContentExtension ie displaying the normal notification in an extended manner

## Section 109.1: Creating a simple UNNotificationContentExtension

**Step 1**

Making the environment suitable for Notification. Make sure you enabled **Background Modes** and **Push Notification**
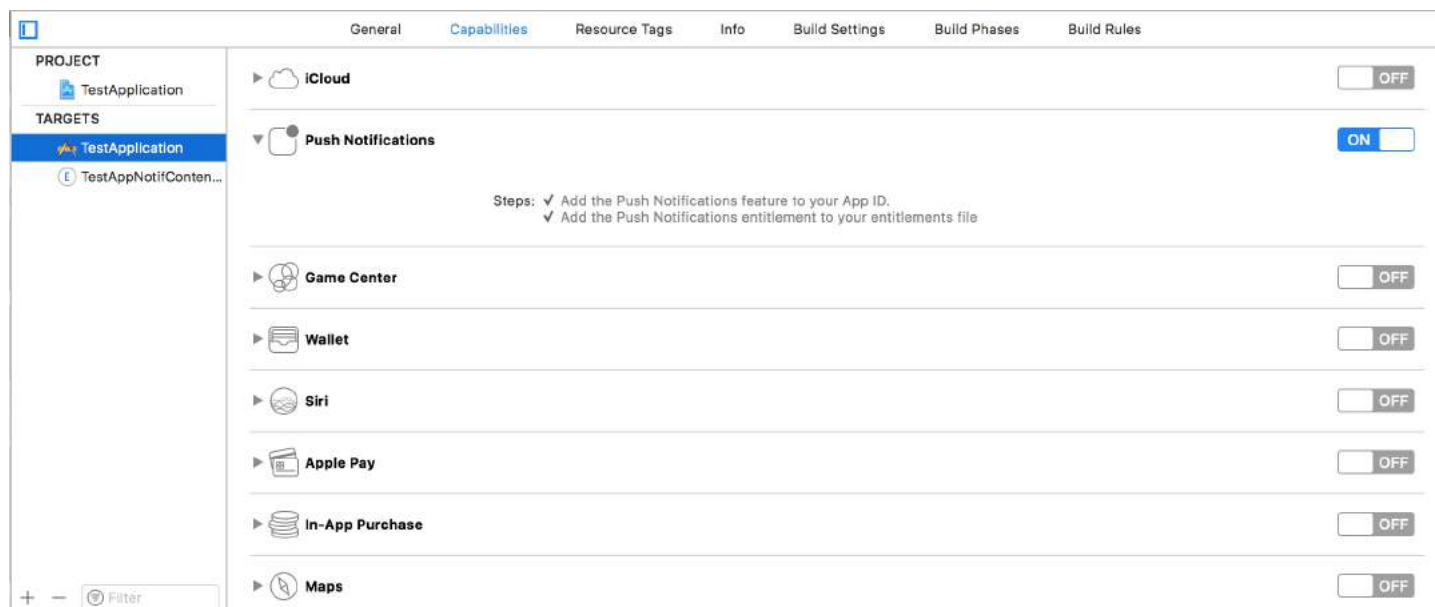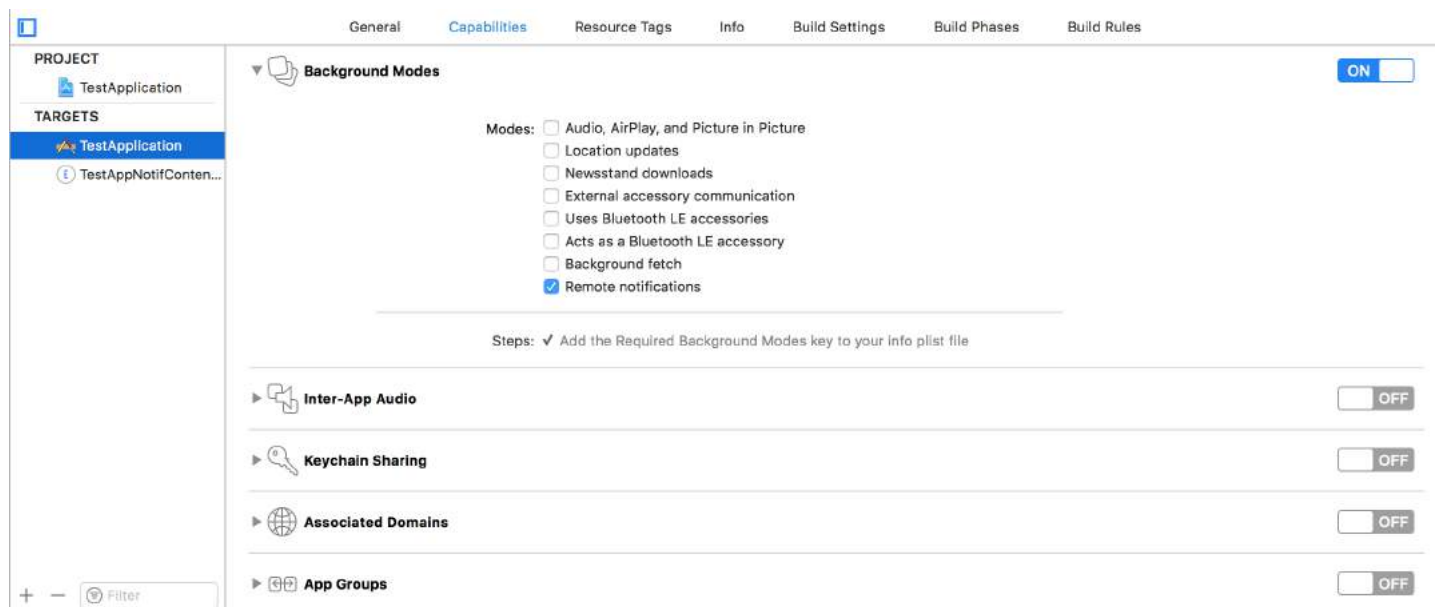




**Step 2: Creating an UNNotificationContentExtension**

Click on the **+** icon in the bottom which creates a target template and select Notification Content Extention -> next -> create a name for the content extension -> finish
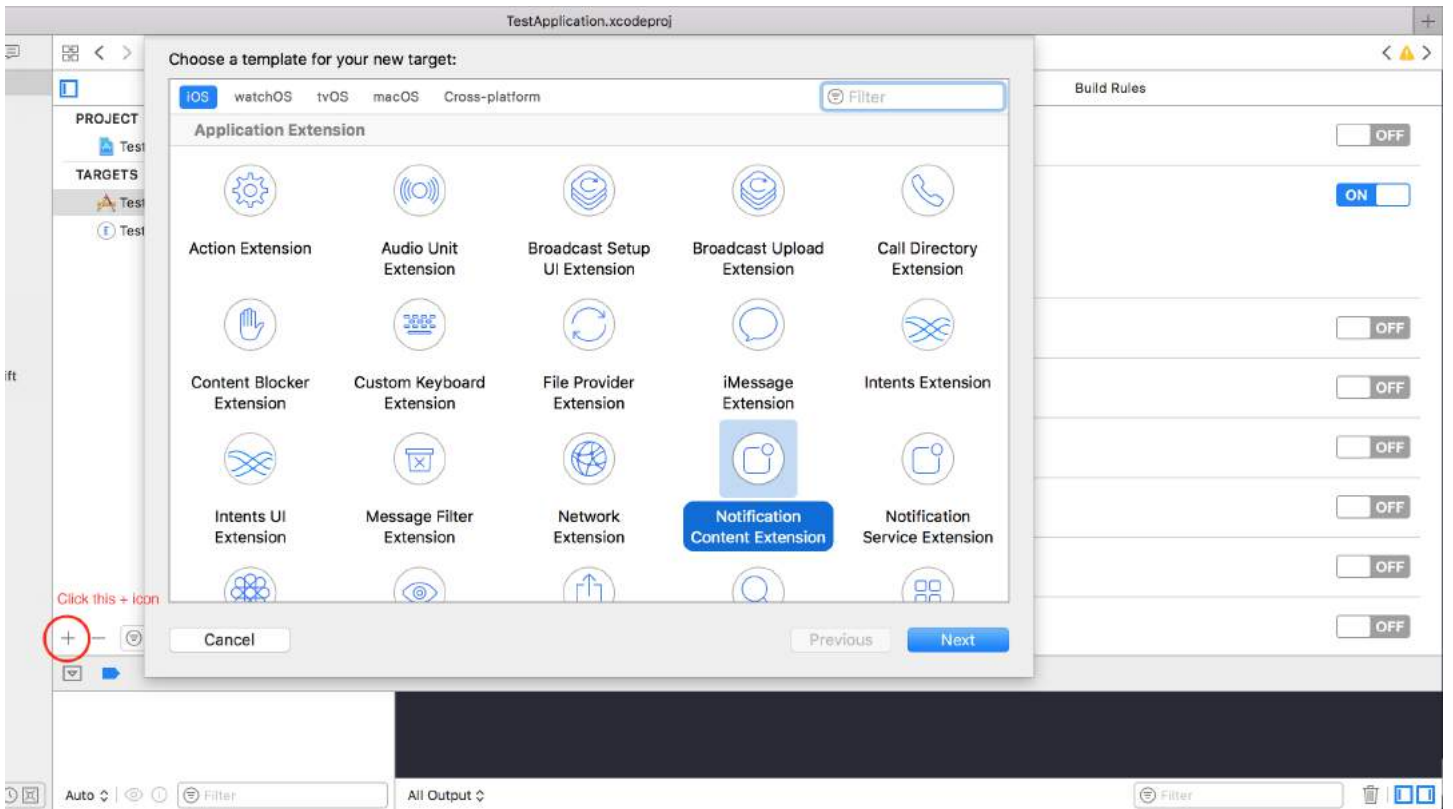
**Step 3:Configuring the info.plist file of the created extension**
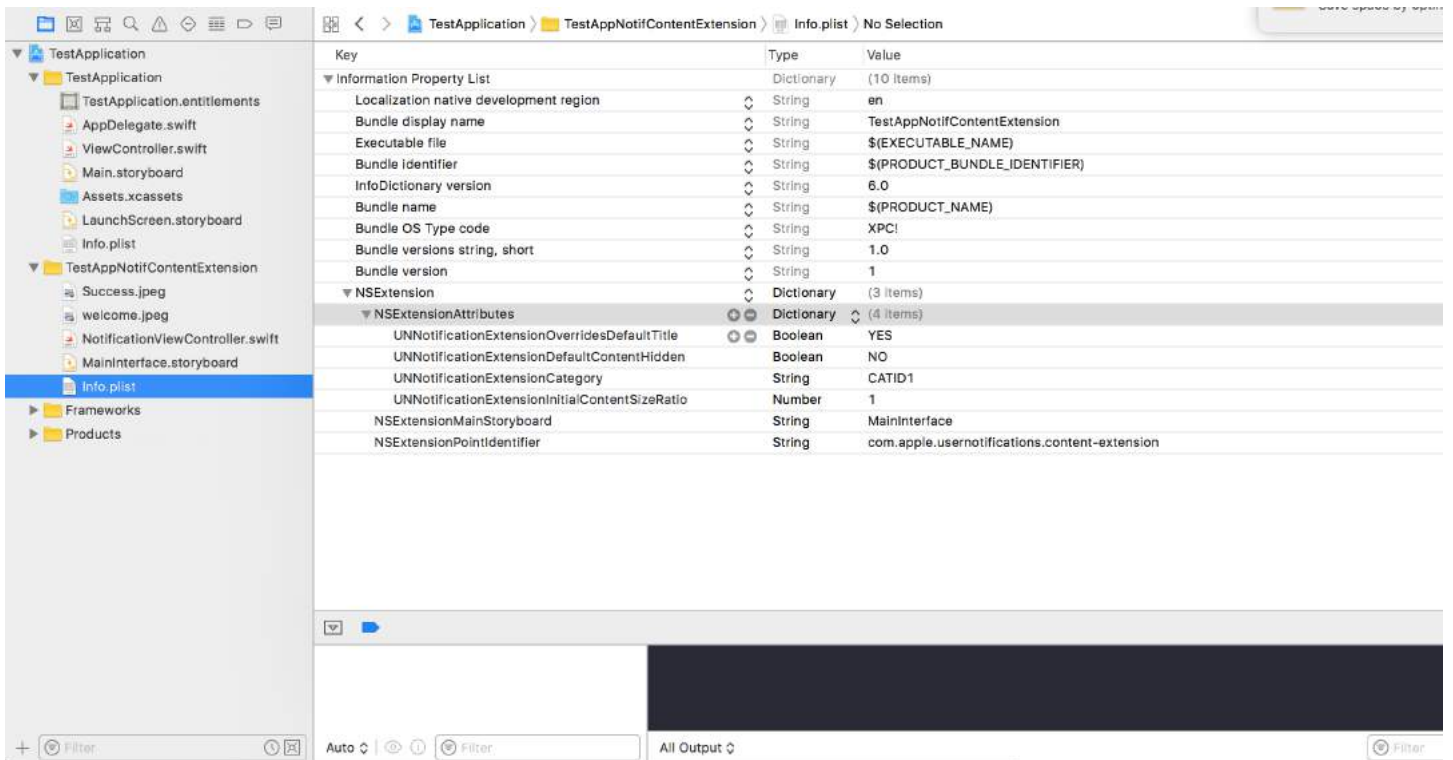


The dictionary in NSExtension signifies how the notification content is displayed, these are performed on long pressing the received notification

- UNNotificationExtensionOverridesDefaultTitle: We can give custom title for our notification by default it displays the name of the application `self.title = myTitle`
- UNNotificationDefaultContentHidden: This boolean determines whether the default body of the notification is to be hidden or not
- UNNotificationCategory: Category is created in `UNUserNotificationCenter` in your application. Here it can be either a string or an array of strings, so each category can gave different types of data from which we can

create different UI's. The payload we send must contain the category name in order to display this particular extension

- UNNotificationExtensionInitialContentSizeRatio: The size of the initial content ie when displaying the ContentExtension for the first time the initial size with respect to width of the device. here 1 denotes the height will be equal to the width

**Step 4: Creating `UNNotificationAction` and `UNNotificationCategory` in our application**

In your app's AppDelegate.swift `didFinishLaunchingWithOptions` function add

```
    let userNotificationAction:UNNotificationAction = UNNotificationAction.init(identifier: "ID1",
title: "???????", options: .destructive)
    let userNotificationAction2:UNNotificationAction = UNNotificationAction.init(identifier: "ID2",
title: "Success", options: .destructive)

    let notifCategory:UNNotificationCategory = UNNotificationCategory.init(identifier: "CATID1",
actions: [userNotificationAction,userNotificationAction2], intentIdentifiers: ["ID1","ID2"] ,
options:.customDismissAction)

    UNUserNotificationCenter.current().delegate = self
    UNUserNotificationCenter.current().setNotificationCategories([notifCategory])
    UIApplication.shared.registerForRemoteNotifications()
```

We created two `UNNotificationAction` with identifiers `ID1` and `ID2` and added those actions to a `UNNotificationCategory` with identifier `CATID1` (the categoryID in ContentExtension's info.plist file are same, what we created here should be used in payload and the plist file). We set the category to our application's `UNUserNotificationCenter` and in next line we are registering for the notification which calls the `didRegisterForRemoteNotificationsWithDeviceToken` function where we get the device token

Note: don't forget to `import` UserNotifications in your AppDelegate.swift and add `UNUserNotificationCenterDelegate`

**Step 5: Sample payload for the NotificationContent**

```
 'aps': {
     'badge': 0,
     'alert': {
         'title': "Rich Notification",
         'body': "Body of RICH NOTIFICATION",
         },
     'sound'  : "default",
     'category': "CATID1",
     'mutable-content':"1",
     },
'attachment': "2"
```

**Step 6: Configuring the ContentExtension**

The corresponding actions for the category is automatically displayed while the notification action is performed. Lets us see the code how its being performed

```
import UIKit
import UserNotifications
import UserNotificationsUI

class NotificationViewController: UIViewController, UNNotificationContentExtension {

@IBOutlet var imageView: UIImageView?
```

```
override func viewDidLoad() {
    super.viewDidLoad()
}

func didReceive(_ notification: UNNotification) {
     self.title = "Koushik"
    imageView?.backgroundColor = UIColor.clear
    imageView?.image = #imageLiteral(resourceName: "welcome.jpeg")
}

func didReceive(_ response: UNNotificationResponse, completionHandler completion: @escaping
(UNNotificationContentExtensionResponseOption) -> Void) {

    self.title = "Koushik"
    imageView?.image = UIImage.init(named: "Success.jpeg")

    if(response.actionIdentifier == "ID1")
    {
       imageView?.image = UIImage.init(named: "Success.jpeg")
    }
    else
    {
        imageView?.image = UIImage.init(named: "welcome.jpeg")
    }

    }
}
```

**Step 7: Result**

After receiving and long press/Clicking View notification, the notification looks like this
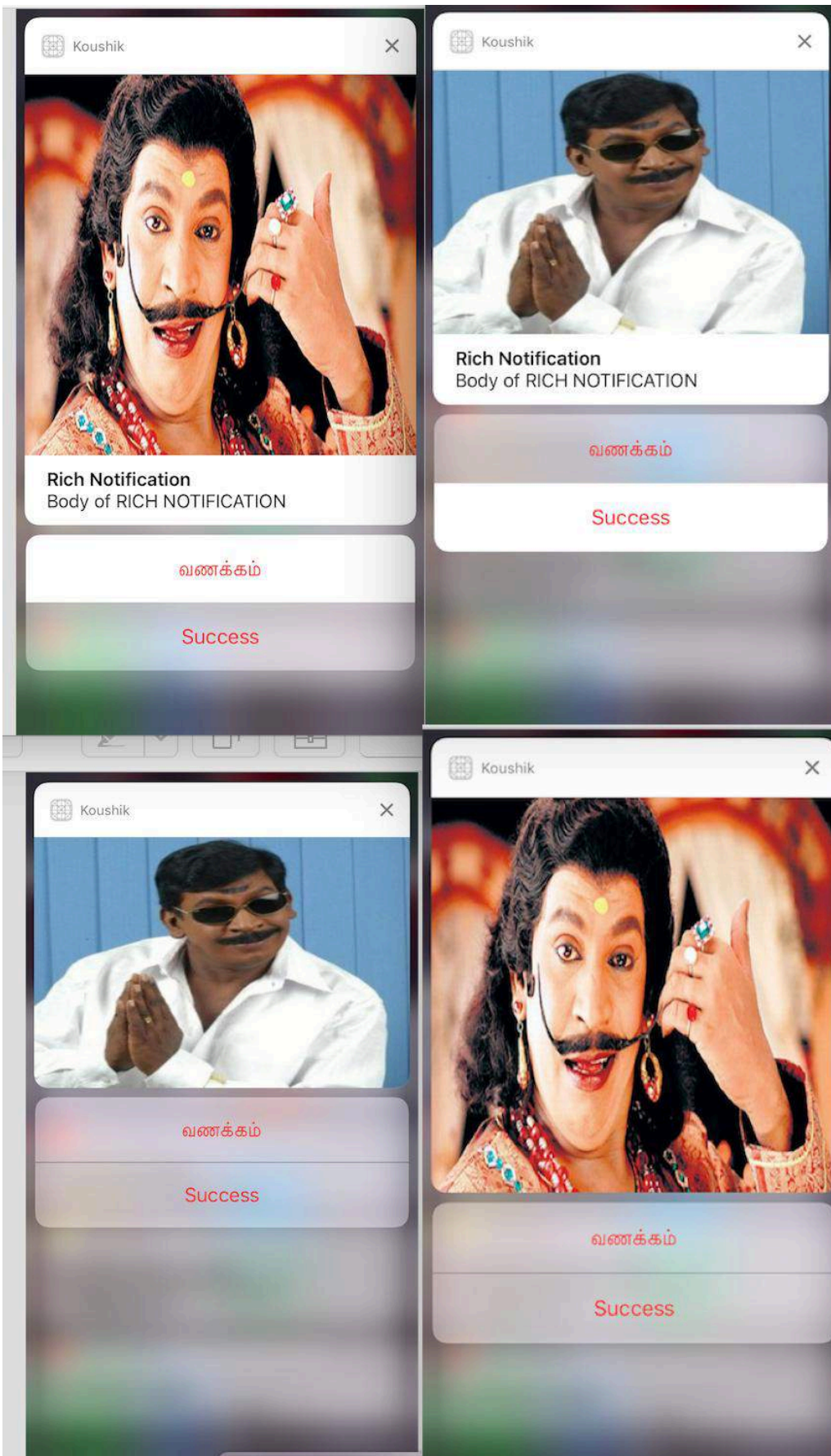
The title is "Koushik" since we gave `self.title = "Koushik"` and `UNNotificationExtensionOverrideDefaultTitle` as YES. In step 3 we gave `UNNotificationExtensionDefaultContentHidden` as NO if its YES then the notification will look like images 3 and 4.

# Chapter 110: Key Value Coding-Key Value Observation

## Section 110.1: Observing a property of a NSObject subclass

Most KVO and KVC functionality is already implemented by default on all `NSObject` subclasses.

To start observing a property named `firstName` of an object named `personObject` do this in the observing class:

```
[personObject addObserver:self
               forKeyPath:@"firstName"
                  options:NSKeyValueObservingOptionNew
                  context:nil];
```

The object that `self` in the above code refers to will then receive a
`observeValueForKeyPath:ofObject:change:context:` message whenever the observed key path changes.

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary<NSString *,id> *)change
                       context:(void *)context
{
    NSLog(@"new value of %@ is: %@", keyPath, change[NSKeyValueChangeNewKey]);
}
```

"Key path" is a KVC term. `NSObject` subclasses implement KVC functionality by default.

An instance variable named `_firstName` will be accessible by the `@"firstName"` key path.

A getter method named `firstName` will be called when accessing the `@"firstName"` key path, regardless of there being a `_firstName` instance variable or `setFirstName` setter method.

## Section 110.2: Use of context for KVO Observation

```
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary<NSString
*,id> *)change context:(void *)context
```

Context is important if you ship your class for others to use.Context lets your class observer verify that its you observer which is being called.

The problem with not passing an observer is, if some one subclass your class and register an observer for the same object,same key and he does not passes a context ,then the super class observer can be called multiple time.

A variable which is unique and internal for your use is a good context.

For more information.

importance and good context

# Chapter 111: Initialization idioms

## Section 111.1: Factory method with block

```swift
internal func Init<Type>(value : Type, block: @noescape (object: Type) -> Void) -> Type
{
    block(object: value)
    return value
}
```

Usage:

```swift
Init(UILabel(frame: CGRect.zero)) {
    $0.backgroundColor = UIColor.blackColor()
}
```

## Section 111.2: Set to tuples to avoid code repetition

Avoid code repetition in constructors by setting a tuple of variables with a one liner:

```swift
class Contact: UIView
{
    private var message: UILabel
    private var phone: UITextView

    required init?(coder aDecoder: NSCoder) {
        (message, phone) = self.dynamicType.setUp()
        super.init(coder: aDecoder)
    }

    override func awakeFromNib() {
        (message, phone) = self.dynamicType.setUp()
        super.awakeFromNib()
    }

    override init(frame: CGRect) {
        (message, phone) = self.dynamicType.setUp()
        super.init(frame: frame)
    }

    private static func setUp(){
        let message = UILabel()  // ...
        let phone = UITextView() // ...
        return (message, phone)
    }
}
```

## Section 111.3: Initialize with positional constants

```swift
let mySwitch: UISwitch = {
    view.addSubview($0)
    $0.addTarget(self, action: "action", forControlEvents: .TouchUpInside)
    return $0
}(UISwitch())
```

# Section 111.4: Initialize attributes in didSet

```
@IBOutlet weak var title: UILabel! {
  didSet {
    label.textColor = UIColor.redColor()
    label.font = UIFont.systemFontOfSize(20)
    label.backgroundColor = UIColor.blueColor()
  }
}
```

It's also possible to both set a value and initialize it:

```
private var loginButton = UIButton() {
    didSet(oldValue) {
        loginButton.addTarget(self, action: #selector(LoginController.didClickLogin),
forControlEvents: .TouchUpInside)
    }
}
```

# Section 111.5: Group outlets in a custom NSObject

Move every outlet to an NSObject. Then drag an Object from the library to the controller scene of the storyboard and hook the elements there.

```
class ContactFormStyle: NSObject
{
    @IBOutlet private weak var message: UILabel! {
      didSet {
        message.font = UIFont.systemFontOfSize(12)
        message.textColor = UIColor.blackColor()
      }
    }
}

class ContactFormVC: UIViewController
{
    @IBOutlet private var style: ContactFormStyle!
}
```

# Section 111.6: Initialize with then

This is similar in syntax to the example that initializes using positional constants, but requires the Then extension from https://github.com/devxoul/Then (attached below).

```
let label = UILabel().then {
    $0.textAlignment = .Center
    $0.textColor = UIColor.blackColor(
    $0.text = "Hello, World!"
}
```

The Then extension:

```
import Foundation

public protocol Then {}

extension Then
```

```
{
    public func then(@noescape block: inout Self -> Void) -> Self {
        var copy = self
        block(&copy)
        return copy
    }
}

extension NSObject: Then {}
```

# Chapter 112: Storyboard

Normally, view controllers in a storyboard are instantiated and created automatically in response to actions defined within the storyboard itself. However, you can use a storyboard object to instantiate the initial view controller in a storyboard file or instantiate other view controllers that you want to present programmatically. Below you will find examples of both use cases.

## Section 112.1: Initialize

```
//Swift
let storyboard = UIStoryboard(name: "Main", bundle: NSBundle.mainBundle())

//Objective-c
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
```

## Section 112.2: Fetch Initial ViewController

```
//Swift
let initialScreen = storyboard.instantiateInitialViewController()

//Objective-c
UIViewController *initailScreen = [storyboard instantiateInitialViewController];
```

## Section 112.3: Fetch ViewController

```
//Swift
let viewController = storyboard.instantiateViewControllerWithIdentifier("identifier")

//Objective-c
UIViewController *viewController = [storyboard
instantiateViewControllerWithIdentifier:@"identifier"];
```

# Chapter 113: Background Modes and Events

## Section 113.1: Play Audio in Background

Add a key named **Required background modes** in property list (.plist) file ..

as following picture..



| Key | Type | Value |
| --- | --- | --- |
| ▼ Information Property List | Dictionary | (16 items) |
| Localization native development r... | String | en |
| Bundle display name | String | |
| Executable file | String | ${EXECUTABLE_NAME} |
| ▶ Icon files | Array | (14 items) |
| Bundle identifier | String | |
| InfoDictionary version | String | 6.0 |
| Bundle name | String | ${PRODUCT_NAME} |
| Bundle OS Type code | String | APPL |
| Bundle versions string, short | String | 1.1 |
| Bundle creator OS Type code | String | ???? |
| Bundle version | String | 1.1 |
| Application requires iPhone envir... | Boolean | YES |
| ▼ Required background modes | Array | (1 item) |
| Item 0 | String | App plays audio or streams audio/video using AirPlay |
| Icon already includes gloss effects | Boolean | YES |
| ▶ Required device capabilities | Array | (1 item) |
| ▶ Supported interface orientations | Array | (1 item) |

And add following code in

**AppDelegate.h**

```
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>
```

**AppDelegate.m**

in application didFinishLaunchingWithOptions

```
[[AVAudioSession sharedInstance] setDelegate:self];
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback error:nil];
[[AVAudioSession sharedInstance] setActive:YES error:nil];
[[UIApplication sharedApplication] beginReceivingRemoteControlEvents];

UInt32 size = sizeof(CFStringRef);
CFStringRef route;
AudioSessionGetProperty(kAudioSessionProperty_AudioRoute, &size, &route);
NSLog(@"route = %@", route);
```

If you want changes as per events you have to add following code in AppDelegate.m

```
- (void)remoteControlReceivedWithEvent:(UIEvent *)theEvent {
```

```
    if (theEvent.type == UIEventTypeRemoteControl)     {
        switch(theEvent.subtype)          {
            case UIEventSubtypeRemoteControlPlay:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            case UIEventSubtypeRemoteControlStop:
                break;
            case UIEventSubtypeRemoteControlTogglePlayPause:
                [[NSNotificationCenter defaultCenter] postNotificationName:@"TogglePlayPause"
object:nil];
                break;
            default:
                return;
        }
    }
}
```

Based on notification have to work on it..

# Chapter 114: Fastlane

## Section 114.1: fastlane tools

[fastlane](#) is an open source build automation tool for Android and iOS for developers. It reduce your build generation time. It is a command line tool that uses [Ruby](#), so you need Ruby on your computer. Most Macs already have Ruby installed by default.

**Install fastlane**

1. Open a terminal.
2. Run `sudo gem install fastlane --verbose`
3. If you haven't installed the Xcode command-line tools yet, run `xcode-select --install` to install them
4. Now, `cd` into your project folder (type **`cd`** [with the space at the end] and drag your project folder into the terminal)
5. Run `fastlane init` to get fastlane setup.
6. Now you can able to use all the Fastlane tools:

**iOS Tools**

- [deliver](#): Upload screenshots, metadata, and your app to the App Store
- [snapshot](#): Automate taking localized screenshots of your iOS app on every device
- [frameit](#): Quickly put your screenshots into the right device frames
- [pem](#): Automatically generate and renew your push notification profiles
- [sigh](#): Because you would rather spend your time building stuff than fighting provisioning
- [produce](#): Create new iOS apps on iTunes Connect and Dev Portal using the command line
- [cert](#): Automatically create and maintain iOS code signing certificates
- [gym](#): Building your iOS apps has never been easier
- [match](#): Easily sync your certificates and profiles across your team using Git
- [scan](#): The easiest way to run tests for your iOS and Mac apps
- [spaceship](#): Ruby library to access the Apple Dev Center and iTunes Connect

**iOS TestFlight Tools**

- [pilot](#): The best way to manage your TestFlight testers and builds from your terminal
- [boarding](#): The easiest way to invite your TestFlight beta testers

**Android Tools**

- [supply](#): Upload your Android app and its metadata to Google Play
- [screengrab](#): Automate taking localized screenshots of your Android app on every device

# Chapter 115: CAShapeLayer

## Section 115.1: Draw Rectangle

```objc
 CAShapeLayer *mask = [[CAShapeLayer alloc] init];

mask.frame = CGRectMake(50, 50, 100, 100);

CGFloat width = 100;

CGFloat height = 100;

CGMutablePathRef path = CGPathCreateMutable();

CGPathMoveToPoint(path, nil, 30, 30);

CGPathAddLineToPoint(path, nil, width, 30);

CGPathAddLineToPoint(path, nil, width, height);

CGPathAddLineToPoint(path, nil,30, height);

CGPathAddLineToPoint(path, nil, 30, 30);

CGPathCloseSubpath(path);


mask.path = path;

CGPathRelease(path);

self.view.layer.mask = mask;
```
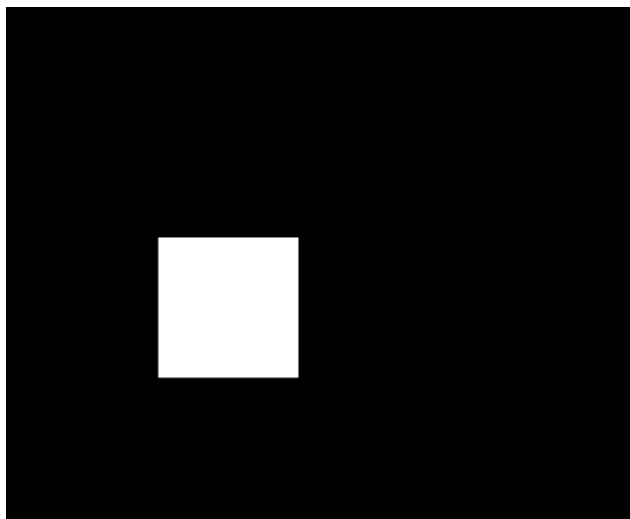


## Section 115.2: Draw Circle

```objc
 CAShapeLayer *circle = [CAShapeLayer layer];

    [circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)]
CGPath]];

    [circle setStrokeColor:[[UIColor blueColor] CGColor]];
```

```
[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];
```
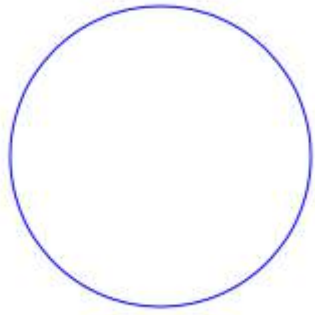


# Section 115.3: CAShapeLayer Animation

```
CAShapeLayer *circle = [CAShapeLayer layer];

[circle setPath:[[UIBezierPath bezierPathWithOvalInRect:CGRectMake(100, 100, 150, 150)] CGPath]];

[circle setStrokeColor:[[UIColor blueColor] CGColor]];

[circle setFillColor:[[UIColor clearColor] CGColor]];

[[self.view layer] addSublayer:circle];


CABasicAnimation *pathAnimation = [CABasicAnimation animationWithKeyPath:@"strokeEnd"];

pathAnimation.duration = 1.5f;

pathAnimation.fromValue = [NSNumber numberWithFloat:0.0f];

pathAnimation.toValue = [NSNumber numberWithFloat:1.0f];

pathAnimation.repeatCount = 10;

pathAnimation.autoreverses = YES;

[circle addAnimation:pathAnimation
            forKey:@"strokeEnd"];
```

# Section 115.4: Basic CAShapeLayer Operation

**UIBezierPath using to create a circular path ShapeLayer**

```
CAShapeLayer *circleLayer = [CAShapeLayer layer];
[circleLayer setPath:[[UIBezierPath bezierPathWithOvalInRect:
CGRectMake(50, 50, 100, 100)] CGPath]];
circleLayer.lineWidth = 2.0;
[circleLayer setStrokeColor:[[UIColor redColor] CGColor]];
[circleLayer setFillColor:[[UIColor clearColor] CGColor]];
circleLayer.lineJoin = kCALineJoinRound;  //4 types are available to create a line style
circleLayer.lineDashPattern = [NSArray arrayWithObjects:
[NSNumber numberWithInt:2],[NSNumber numberWithInt:3 ], nil];
// self.origImage is parentView
[[self.view layer] addSublayer:circleLayer];
self.currentShapeLayer = circleLayer;  // public value using to keep that reference of the shape
Layer
self.view.layer.borderWidth = 1.0f;
self.view.layer.borderColor = [[UIColor blueColor]CGColor];  // that will plotted in the mainview
```

**Remove ShapeLayer**

Keep a reference to that shape layer. For example, you might have a property currentShapeLayer: Now that you have a reference, you can easily remove the layer:

Type 1:

```
[self.currentShapeLayer removeFromSuperlayer];
```

Type 2:

```
self.view.layer.sublayers = nil ;                    //removed all earlier shapes
```

**Other Operation**

```
 //Draw Square Shape

CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 20, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = nil;
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//Draw Circle Shape

CAShapeLayer *circleShape = [CAShapeLayer layer];
circleShape.frame = CGRectMake(160, 20, 120, 120);
circleShape.lineWidth = 2.0;
circleShape.fillColor = nil;
circleShape.strokeColor = [[UIColor redColor] CGColor];
circleShape.path = [UIBezierPath bezierPathWithOvalInRect:circleShape.bounds].CGPath;
[[self.view layer] addSublayer:circleShape];


//Subpaths
//UIBezierPath can have any number of "path segments" (or subpaths) so you can effectively draw as
```

```
many shapes or lines as you want in a single path object

CAShapeLayer *shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(20, 140, 200, 200);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

CGMutablePathRef combinedPath= CGPathCreateMutableCopy(circleShape.path);
CGPathAddPath(combinedPath, NULL, squareLayer.path);

shapeLayer.path = combinedPath;
[[self.view layer] addSublayer:shapeLayer];

//Open Path
// Paths do not need to connect their end points back to their starting points. A path that
connects back to its starting point is called a closed path, and one that does not is called an
open path.

shapeLayer = [CAShapeLayer layer];
shapeLayer.frame = CGRectMake(160, 140, 300, 300);
shapeLayer.lineWidth = 2.0;
shapeLayer.fillColor = nil;
shapeLayer.strokeColor = [[UIColor redColor] CGColor];

UIBezierPath *linePath=[UIBezierPath bezierPath];
[linePath moveToPoint:CGPointZero];
[linePath addLineToPoint:CGPointMake(0 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 120)];
[linePath addLineToPoint:CGPointMake(120 , 0)];
shapeLayer.path = linePath.CGPath;
[[self.view layer] addSublayer:shapeLayer];
```
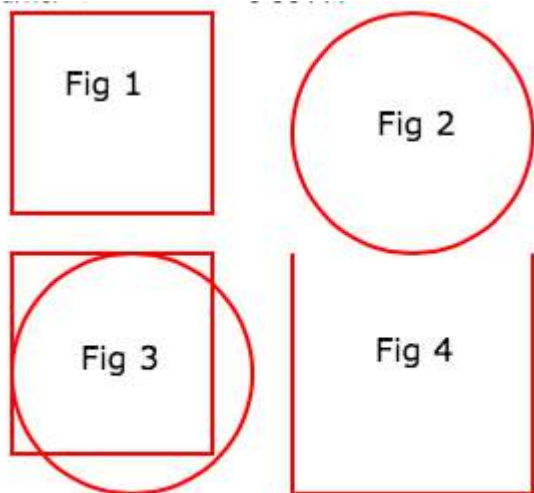


**Fill Concepts** //Fill Color

```
CAShapeLayer *squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(20, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];
```

```objc
//Fill Pattern Color
//images.jpeg

squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 30, 100, 100);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor colorWithPatternImage:[UIImage
imageNamed:@"images.jpeg"]]CGColor];
squareLayer.strokeColor = [[UIColor redColor] CGColor];
squareLayer.path = [UIBezierPath bezierPathWithRect:squareLayer.bounds].CGPath;
[[self.view layer] addSublayer:squareLayer];


//Fill Rule

//Type 1: kCAFillRuleNonZero
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(0, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleNonZero;    // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
UIBezierPath *outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0,
20.0)];
UIBezierPath *innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0,
50.0)];
CGMutablePathRef combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];


//Type 2: kCAFillRuleEvenOdd
squareLayer = [CAShapeLayer layer];
squareLayer.frame = CGRectMake(140, 140, 150, 150);
squareLayer.lineWidth = 2.0;
squareLayer.fillColor = [[UIColor yellowColor]CGColor];
squareLayer.fillRule = kCAFillRuleEvenOdd;    // indicate the rule type
squareLayer.strokeColor = [[UIColor redColor] CGColor];
outerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 20.0, 20.0)];
innerPath = [UIBezierPath bezierPathWithRect:CGRectInset(squareLayer.bounds, 50.0, 50.0)];
combinedPath= CGPathCreateMutableCopy(outerPath.CGPath);
CGPathAddPath(combinedPath, NULL, innerPath.CGPath);
squareLayer.path = combinedPath;
[[self.view layer] addSublayer:squareLayer];
```
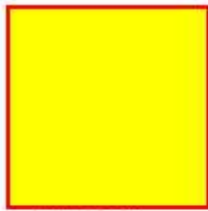
Fill color    Fill Pattern
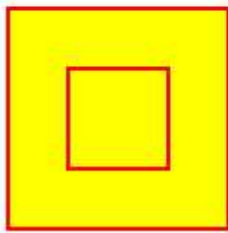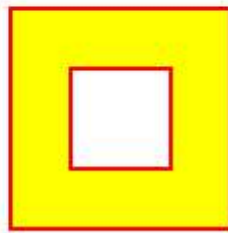
Fill Rule Zero    Fill Rule Even

Listed the accessing Style properties

```
fillColor
     Fill the color based on the drawed shape.

fillRule
     Fill Rule the there are two rule is applied to draw the shape.
     1.  kCAFillRuleNonZero
     2.  kCAFillRuleEvenOdd

lineCap
     Below type used to change the style of the line.
     1.  kCALineCapButt
     2.  kCALineCapRound
     3.  kCALineCapSquare

lineDashPattern
     The dash pattern applied to the shape's path when stroked.
     Create DashStyle while you will stroke the line.

lineDashPhase
     The dash phase applied to the shape's path when stroked. Animatable.
lineJoin
     Line join style for the shape path.Below style use to draw the line join style.
     1.  kCALineJoinMiter
     2.  kCALineJoinRound
     3.  kCALineJoinBevel

lineWidth
     Which using to set the line width.

miterLimit
     The miter limit used when stroking the shape's path. Animatable.

strokeColor
     Set the stroke color based on the path of the line.

strokeStart
     When the stroke will start.

strokeEnd
     When the stroke will end.
```

# Chapter 116: WKWebView

WKWebView is the centerpiece of the modern WebKit API introduced in iOS 8 & OS X Yosemite. It replaces UIWebView in UIKit and WebView in AppKit, offering a consistent API across the two platforms.

Boasting responsive 60fps scrolling, built-in gestures, streamlined communication between app and webpage, and the same JavaScript engine as Safari, WKWebView is one of the most significant announcements to come out of WWDC 2014.

## Section 116.1: Adding custom user script loaded from app bundle

```
let configuration = WKWebViewConfiguration()

if let path = NSBundle.mainBundle().pathForResource("customUserScript", ofType: "js"),
       source = try? NSString(contentsOfFile: path, encoding: NSUTF8StringEncoding) as String {

       let userScript = WKUserScript(source: source, injectionTime:
WKUserScriptInjectionTime.AtDocumentStart, forMainFrameOnly: false)

       let userContentController = WKUserContentController()
       userContentController.addUserScript(userScript)
       configuration.userContentController = userContentController
    }

let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

Any value in the WKUserScriptInjectionTime enum is valid: .AtDocumentStart, .AtDocumentEnd

## Section 116.2: Send messages from JavaScript and Handle them on the native side

Messages can be sent from JavaScript using the following code

```
window.webkit.messageHandlers.{NAME}.postMessage()
```

Here how to create a script message handler to handle the messages:

```
class NotificationScriptMessageHandler: NSObject, WKScriptMessageHandler {
    func userContentController(userContentController: WKUserContentController,
didReceiveScriptMessage message: WKScriptMessage!) {
        if message.name == "{NAME}" {
            // to be sure of handling the correct message
            print(message.body)
        }
    }
}
```

Here how to configure the script message handler in the WKWebView:

```
let configuration = WKWebViewConfiguration()
let userContentController = WKUserContentController()
let handler = NotificationScriptMessageHandler()
userContentController.addScriptMessageHandler(handler, name: "{NAME}")
configuration.userContentController = userContentController
```

```
let webView = WKWebView(frame: self.view.bounds, configuration: configuration)
```

> **NOTE:** adding same "{NAME}" handler with addScriptMessageHandler:name: more than once, results in
> NSInvalidArgumentExceptionexception.

# Section 116.3: Creating a Simple WebBrowser

```swift
import UIKit
import WebKit


class ViewController: UIViewController, UISearchBarDelegate, WKNavigationDelegate, WKUIDelegate {

    var searchbar: UISearchBar! //All web-browsers have a search-bar.
    var webView: WKWebView! //The WKWebView we'll use.
    var toolbar: UIToolbar! //Toolbar at the bottom just like in Safari.
    var activityIndicator: UIActivityIndicatorView! //Activity indicator to let the user know the
page is loading.

    override func viewDidLoad() {
        super.viewDidLoad()

        self.initControls()
        self.setTheme()
        self.doLayout()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }


    func initControls() {
        self.searchbar = UISearchBar()

        //WKUserContentController allows us to add Javascript scripts to our webView that will run
either at the beginning of a page load OR at the end of a page load.

        let configuration = WKWebViewConfiguration()
        let contentController = WKUserContentController()
        configuration.userContentController = contentController

        //create the webView with the custom configuration.
        self.webView = WKWebView(frame: .zero, configuration: configuration)

        self.toolbar = UIToolbar()
        self.layoutToolbar()

        self.activityIndicator = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        self.activityIndicator.hidesWhenStopped = true
    }

    func setTheme() {
        self.edgesForExtendedLayout = UIRectEdge(rawValue: 0)
        self.navigationController?.navigationBar.barTintColor = UIColor.white()

        //Theme the keyboard and searchBar. Setup delegates.
        self.searchbar.delegate = self
        self.searchbar.returnKeyType = .go
```

```
        self.searchbar.searchBarStyle = .prominent
        self.searchbar.placeholder = "Search or enter website name"
        self.searchbar.autocapitalizationType = .none
        self.searchbar.autocorrectionType = .no

        //Set the WebView's delegate.
        self.webView.navigationDelegate = self //Delegate that handles page navigation
        self.webView.uiDelegate = self //Delegate that handles new tabs, windows, popups, layout,
etc..

        self.activityIndicator.transform = CGAffineTransform(scaleX: 1.5, y: 1.5)
    }

    func layoutToolbar() {
        //Browsers typically have a back button, forward button, refresh button, and
newTab/newWindow button.

        var items = Array<UIBarButtonItem>()

        let space = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil, action: nil)

        items.append(UIBarButtonItem(title: "<", style: .plain, target: self, action:
#selector(onBackButtonPressed)))
        items.append(space)
        items.append(UIBarButtonItem(title: ">", style: .plain, target: self, action:
#selector(onForwardButtonPressed)))
        items.append(space)
        items.append(UIBarButtonItem(barButtonSystemItem: .refresh, target: self, action:
#selector(onRefreshPressed)))
        items.append(space)
        items.append(UIBarButtonItem(barButtonSystemItem: .organize, target: self, action:
#selector(onTabPressed)))

        self.toolbar.items = items
    }

    func doLayout() {
        //Add the searchBar to the navigationBar.
        self.navigationItem.titleView = self.searchbar

        //Add all other subViews to self.view.
        self.view.addSubview(self.webView)
        self.view.addSubview(self.toolbar)
        self.view.addSubview(self.activityIndicator)

        //Setup which views will be constrained.

        let views: [String: AnyObject] = ["webView": self.webView, "toolbar": self.toolbar,
"activityIndicator": self.activityIndicator];
        var constraints = Array<String>();

        constraints.append("H:|-0-[webView]-0-|")
        constraints.append("H:|-0-[toolbar]-0-|")
        constraints.append("V:|-0-[webView]-0-[toolbar(50)]-0-|")


        //constrain the subviews using the above visual constraints.

        for constraint in constraints {
            self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }
```

```swift
        for view in self.view.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }


        //constraint the activity indicator to the center of the view.
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerX, relatedBy: .equal, toItem: self.view, attribute: .centerX, multiplier: 1.0, constant:
0.0))
        self.view.addConstraint(NSLayoutConstraint(item: self.activityIndicator, attribute:
.centerY, relatedBy: .equal, toItem: self.view, attribute: .centerY, multiplier: 1.0, constant:
0.0))
    }

    //Searchbar Delegates

    func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {
        self.searchbar.resignFirstResponder()

        if let searchText = self.searchbar.text, url = URL(string: searchText) {
            //Get the URL from the search bar. Create a new NSURLRequest with it and tell the
webView to navigate to that URL/Page. Also specify a timeout for if the page takes too long. Also
handles cookie/caching policy.

            let request = URLRequest(url: url, cachePolicy: .useProtocolCachePolicy,
timeoutInterval: 30)
            self.webView.load(request)
        }
    }


    //Toolbar Delegates

    func onBackButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoBack) { //allow the user to go back to the previous page.
            self.webView.goBack()
        }
    }

    func onForwardButtonPressed(button: UIBarButtonItem) {
        if (self.webView.canGoForward) { //allow the user to go forward to the next page.
            self.webView.goForward()
        }
    }

    func onRefreshPressed(button: UIBarButtonItem) {
        self.webView.reload()  //reload the current page.
    }

    func onTabPressed(button: UIBarButtonItem) {
        //TODO: Open a new tab or web-page.
    }


    //WebView Delegates

    func webView(_ webView: WKWebView, decidePolicyFor navigationAction: WKNavigationAction,
decisionHandler: (WKNavigationActionPolicy) -> Void) {

        decisionHandler(.allow) //allow the user to navigate to the requested page.
    }
```

```swift
    func webView(_ webView: WKWebView, decidePolicyFor navigationResponse: WKNavigationResponse,
decisionHandler: (WKNavigationResponsePolicy) -> Void) {

        decisionHandler(.allow) //allow the webView to process the response.
    }

    func webView(_ webView: WKWebView, didStartProvisionalNavigation navigation: WKNavigation!) {
        self.activityIndicator.startAnimating()
    }

    func webView(_ webView: WKWebView, didFailProvisionalNavigation navigation: WKNavigation!,
withError error: NSError) {
        self.activityIndicator.stopAnimating()

        //Handle the error. Display an alert to the user telling them what happened.

        let alert = UIAlertController(title: "Error", message: error.localizedDescription,
preferredStyle: .alert)
        let action = UIAlertAction(title: "OK", style: .default) { (action) in
            alert.dismiss(animated: true, completion: nil)
        }
        alert.addAction(action)
        self.present(alert, animated: true, completion: nil)
    }

    func webView(_ webView: WKWebView, didFinish navigation: WKNavigation!) {
        self.activityIndicator.stopAnimating()

        //Update our search bar with the webPage's final endpoint-URL.
        if let url = self.webView.url {
            self.searchbar.text = url.absoluteString ?? self.searchbar.text
        }
    }

    func webView(_ webView: WKWebView, didReceiveServerRedirectForProvisionalNavigation navigation:
WKNavigation!) {
        //When the webview receives a "Redirect" to a different page or endpoint, this is called.
    }

    func webView(_ webView: WKWebView, didCommit navigation: WKNavigation!) {
        //When the content for the webpage starts arriving, this is called.
    }

    func webView(_ webView: WKWebView, didFail navigation: WKNavigation!, withError error: NSError)
{

    }

    func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge,
completionHandler: (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

        completionHandler(.performDefaultHandling, .none) //Handle SSL connections by default. We
aren't doing SSL pinning or custom certificate handling.

    }


    //WebView's UINavigation Delegates

    //This is called when a webView or existing loaded page wants to open a new window/tab.
    func webView(_ webView: WKWebView, createWebViewWith configuration: WKWebViewConfiguration, for
```

```
navigationAction: WKNavigationAction, windowFeatures: WKWindowFeatures) -> WKWebView? {

        //The view that represents the new tab/window. This view will have an X button at the top
left corner + a webView.
        let container = UIView()

        //New tabs need an exit button.
        let XButton = UIButton()
        XButton.addTarget(self, action: #selector(onWebViewExit), for: .touchUpInside)
        XButton.layer.cornerRadius = 22.0

        //Create the new webView window.
        let webView = WKWebView(frame: .zero, configuration: configuration)
        webView.navigationDelegate = self
        webView.uiDelegate = self

        //Layout the tab.
        container.addSubview(XButton)
        container.addSubview(webView)

        let views: [String: AnyObject] = ["XButton": XButton, "webView": webView];
        var constraints = Array<String>()

        constraints.append("H:|-(-22)-[XButton(44)]")
        constraints.append("H:|-0-[webView]-0-|")
        constraints.append("V:|-(-22)-[XButton(44)]-0-[webView]-0-|")


        //constrain the subviews.
        for constraint in constraints {
            container.addConstraints(NSLayoutConstraint.constraints(withVisualFormat: constraint,
options: NSLayoutFormatOptions(rawValue: 0), metrics: nil, views: views))
        }

        for view in container.subviews {
            view.translatesAutoresizingMaskIntoConstraints = false
        }

        //TODO: Add the containerView to self.view or present it with a new controller. Keep track
of tabs..

        return webView
    }

    func onWebViewExit(button: UIButton) {
        //TODO: Destroy the tab. Remove the new tab from the current window or controller.
    }
}
```

Showing the custom `GO` button in the keyboard:
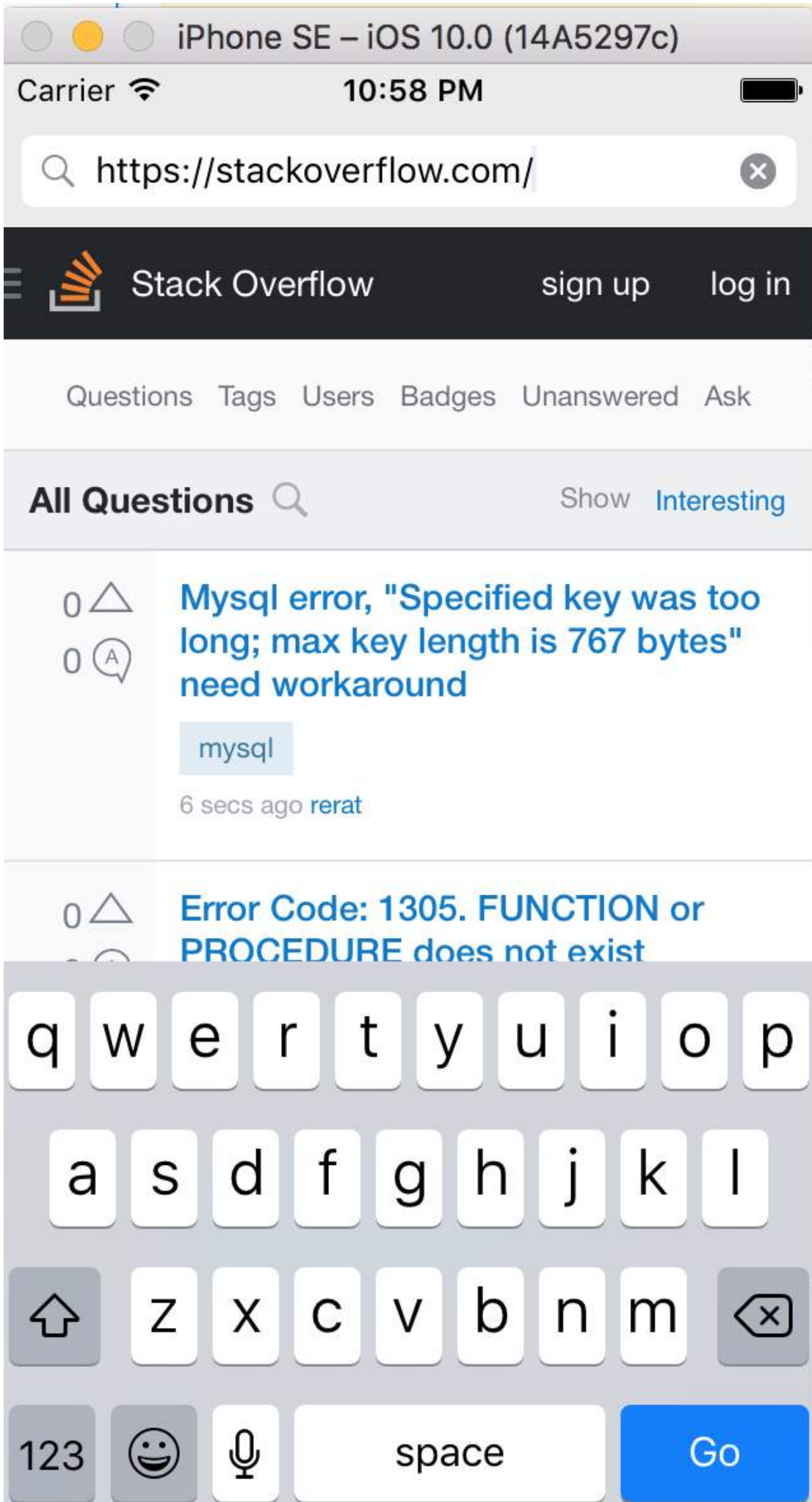
Showing the toolbar and fully loaded page.

iPhone SE – iOS 10.0 (14A5297c)

Carrier 🔋 10:58 PM 🔋

🔍 https://stackoverflow.com/ ✕

Stack Overflow    sign up    log in

Questions  Tags  Users  Badges  Unanswered  Ask

**All Questions** 🔍    Show  Interesting

0 △
0 Ⓐ
**Mysql error, "Specified key was too long; max key length is 767 bytes" need workaround**

mysql

6 secs ago rerat

0 △
0 Ⓐ
**Error Code: 1305. FUNCTION or PROCEDURE does not exist**

mysql

8 secs ago neubert

0 △
0 Ⓐ
**cqlengine python lib is not recognizing batchquery object**

python    cqlengine

14 secs ago gotham

‹    ›    ↻    🗀

# Chapter 117: UUID (Universally Unique Identifier)

## Section 117.1: Apple's IFA vs. IFV (Apple Identifier for Advertisers vs. Identifier for Vendors)

- You can use the IFA for measuring ad clicks and the IFV for measuring app installs.
- IFA has built-in privacy mechanisms that make it perfect for advertising. In contrast, the IFV is for developers to use internally to measure users who install their apps.

**IFA**

- ASIdentifierManager class provides
  - **advertisingIdentifier: UUID**: An alphanumeric string unique to each device, used only for serving advertisements.

  - **isAdvertisingTrackingEnabled**: A Boolean value that indicates whether the user has limited ad tracking.

**IFV**

- ASIdentifierManager class provides
  - **identifierForVendor: UUID**: An alphanumeric string that uniquely identifies a device to the app's vendor.

Find your device IFA and IFV [here](#).

## Section 117.2: Create UUID String for iOS devices

Here we can create UUID `String` with in one line.

Represents UUID strings, which can be used to uniquely identify types, interfaces, and other items.

**Swift 3.0**

```
print(UUID().uuidString)
```

It is very useful for identify multiple devices with unique id.

## Section 117.3: Generating UUID

**Random UUID**
**Swift**

```
func randomUUID() -> NSString{
    return NSUUID.UUID().UUIDString()
}
```

**Objective-C**

```
+ (NSString *)randomUUID {
    if(NSClassFromString(@"NSUUID")) { // only available in iOS >= 6.0
        return [[NSUUID UUID] UUIDString];
    }
    CFUUIDRef uuidRef = CFUUIDCreate(kCFAllocatorDefault);
    CFStringRef cfuuid = CFUUIDCreateString(kCFAllocatorDefault, uuidRef);
```

```
    CFRelease(uuidRef);
    NSString *uuid = [((__bridge NSString *) cfuuid) copy];
    CFRelease(cfuuid);
    return uuid;
}
```

# Section 117.4: Identifier for vendor

`Version ≥ iOS 6`

Within a single line, we can get an UUID like below:

**Swift**

```
let UDIDString = UIDevice.currentDevice().identifierForVendor?.UUIDString
```

**Objective-C**

```
NSString *UDIDString = [[[UIDevice currentDevice] identifierForVendor] UUIDString];
```

The `identifierForVendor` is an unique identifier that stays the same for every app of a single vendor on a single device, unless all of the vendor's apps are deleted from this device. See Apple's documentation about when this UUID changes.

# Chapter 118: Categories

## Section 118.1: Create a Category

Categories provide the ability to add some extra functionality to an object without subclassing or changing the actual object.

For example we want to set some custom fonts. Lets create a category that add functionality to `UIFont` class. Open your Xcode project, click on File -> New -> File and choose Objective-C file , click Next enter your category name say "CustomFont" choose file type as Category and Class as UIFont then Click "Next" followed by "Create."
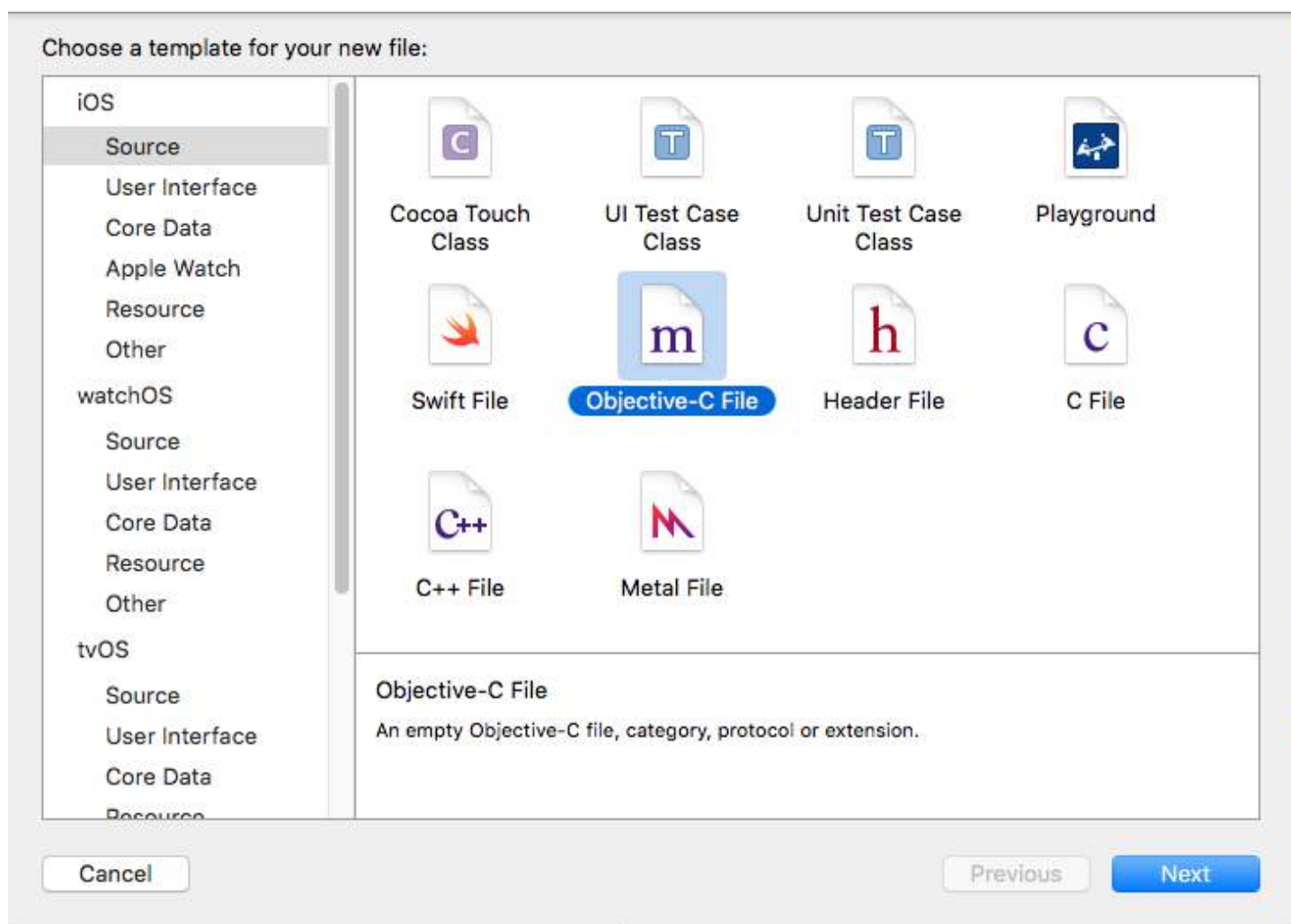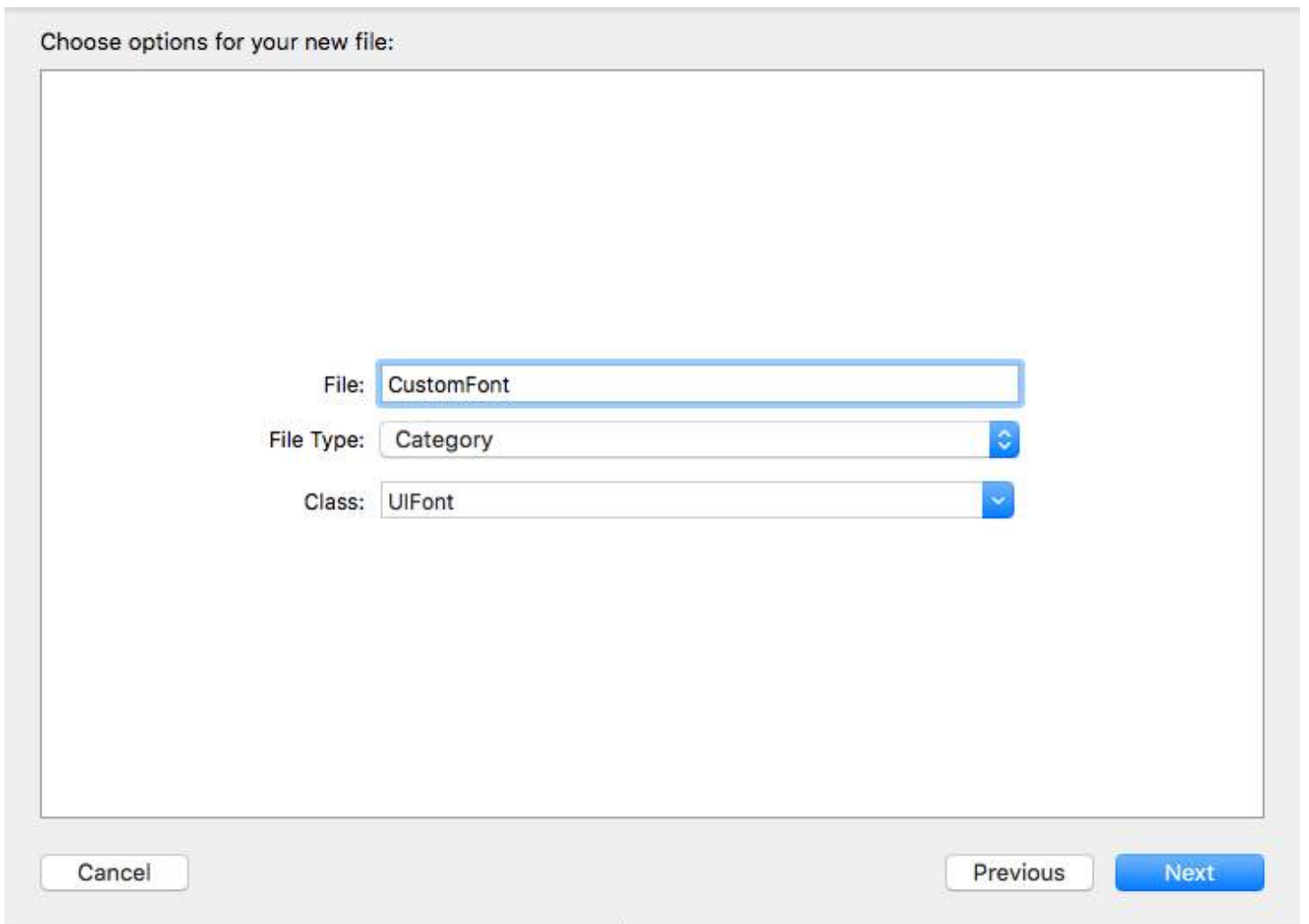
**Declare the Category Method:**

Click "UIFont+CustomFonts.h" to view the new category's header file. Add the following code to the interface to declare the method.

```
@interface UIFont (CustomFonts)

+(UIFont *)productSansRegularFontWithSize:(CGFloat)size;

@end
```

**Now Implement the Category Method:**

Click "UIFont+CustomFonts.m" to view the category's implementation file. Add the following code to create a method that will set ProductSansRegular Font.

```
+(UIFont *)productSansRegularFontWithSize:(CGFloat)size{

    return [UIFont fontWithName:@"ProductSans-Regular" size:size];

}
```

**Import your category**

```
#import "UIFont+CustomFonts.h"
```

Now set the Label font

```
[self.label setFont:[UIFont productSansRegularFontWithSize:16.0]];
```

# Chapter 119: Handling URL Schemes

| Parameter | Meaning |
|---|---|
| aUrl | a NSURL instance which stores a built-in or custom scheme string |

## Section 119.1: Using built-in URL scheme to open Mail app

**Swift:**

```swift
if let url = URL(string: "mailto://azimov@demo.com") {
    if UIApplication.shared.canOpenURL(url) {
        UIApplication.shared.openURL(url)
    } else {
        print("Cannot open URL")
    }
}
```

**Objective-C:**

```objc
NSURL *url = [NSURL URLWithString:@"mailto://azimov@demo.com"];
if ([[UIApplication sharedApplication] canOpenURL:url]) {
    [[UIApplication sharedApplication] openURL:url];
} else {
    NSLog(@"Cannot open URL");
}
```

## Section 119.2: Apple URL Schemes

These are URL schemes supported by native apps on iOS, OS X, and watchOS 2 and later.

**Opening link in Safari:**

Objective-C

```objc
NSString *stringURL = @"http://stackoverflow.com/";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```swift
let stringURL = "http://stackoverflow.com/"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

**Starting a phone conversation**

Objective-C

```objc
NSString *stringURL = @"tel:1-408-555-5555";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```swift
let stringURL = "tel:1-408-555-5555"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
```

```
}
```

HTML

```html
<a href="tel:1-408-555-5555">1-408-555-5555</a>
```

**Starting a FaceTime conversation**

Objective-C

```objc
NSString *stringURL = @"facetime:14085551234";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```swift
let stringURL = "facetime:14085551234"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```html
<a href="facetime:14085551234">Connect using FaceTime</a>
<a href="facetime:user@example.com">Connect using FaceTime</a>
```

**Opening Messages App to compose an sms to recipient:**

Objective-C

```objc
NSString *stringURL = @"sms:1-408-555-1212";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```swift
let stringURL = "sms:1-408-555-1212"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```html
<a href="sms:">Launch Messages App</a>
<a href="sms:1-408-555-1212">New SMS Message</a>
```

**Opening Mail app to compose an email to recipient:**

Objective-C

```objc
NSString *stringURL = @"mailto:foo@example.com";
NSURL *url = [NSURL URLWithString:stringURL];
[[UIApplication sharedApplication] openURL:url];
```

Swift:

```
let stringURL = "mailto:foo@example.com"
if let url = URL(string: stringURL) {
    UIApplication.shared.openURL(url)
}
```

HTML

```
<a href="mailto:frank@wwdcdemo.example.com">John Frank</a>
```

You can also include a subject field, a message, and multiple recipients in the To, Cc, and Bcc fields. (In iOS, the from attribute is ignored.) The following example shows a mailto URL that includes several different attributes:

```
mailto:foo@example.com?cc=bar@example.com&subject=Greetings%20from%20Cupertino!&body=Wish%20you%20w
ere%20here!
```

*Note:* Compose email dialog can also be presented within app using `MFMailComposeViewController`.

# Chapter 120: Realm

## Section 120.1: RLMObject Base Model Class with Primary Key - Objective-C

An example of a RLMObject base model class that uses a primary key and some generic default properties. Subclasses can then set metadata specific to their needs.

```objc
@interface BaseModel : RLMObject

@property NSString *uuid;
@property NSString *metadata;

@end

@implementation BaseModel

+ (NSString *)primaryKey
{
    return @"uuid";
}

+ (NSDictionary *)defaultPropertyValues
{
    NSMutableDictionary *defaultPropertyValues = [NSMutableDictionary
dictionaryWithDictionary:[super defaultPropertyValues]];
    NSString *uuid = [[NSUUID UUID] UUIDString];
    [defaultPropertyValues setValue:@"" forKey:@"metadata"];
    [defaultPropertyValues setValue:uuid forKey:@"uuid"];
    return defaultPropertyValues;
}

+ (NSArray *)ignoredProperties
{
    return @[];
}

@end
```
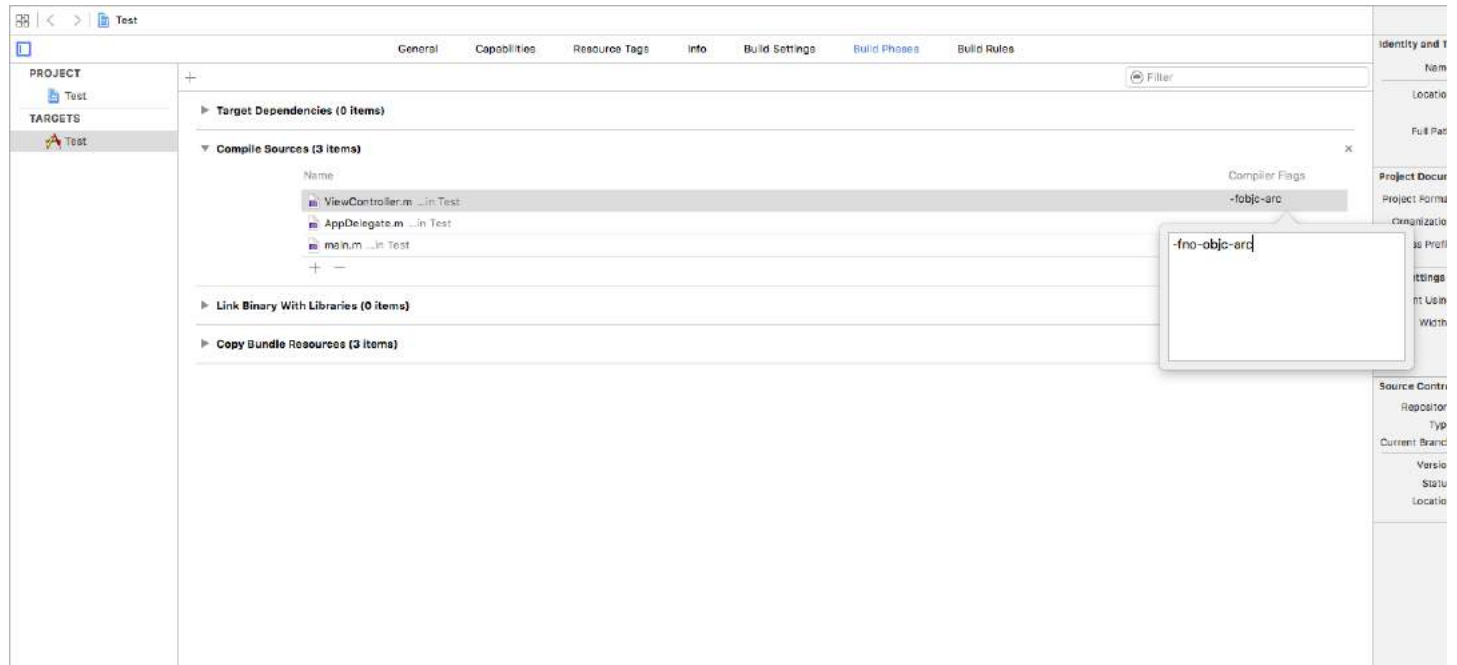
# Chapter 121: ARC (Automatic Reference Counting)

## Section 121.1: Enable/Disable ARC on a file

ARC® can be disabled for individual files by adding the **-fno-objc-arc** compiler flag for each file. Conversely it can be added in *Targets ? Build Phases ? Compile Sources*

# Chapter 122: Dynamic Type

## Section 122.1: Matching Dynamic Type Font Size in WKWebView

WKWebView resizes the fonts on web content so that a full-sized web page will fit on the device's form factor. If you want the web text in both portrait and landscape to be similar in size to the user's preferred reading size, you need to set it explicitly.

**Swift**

```swift
// build HTML header for dynamic type and responsive design
func buildHTMLHeader() -> String {

    // Get preferred dynamic type font sizes for html styles
    let bodySize = UIFont.preferredFont(forTextStyle: UIFontTextStyle.body).pointSize
    let h1Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title1).pointSize
    let h2Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title2).pointSize
    let h3Size = UIFont.preferredFont(forTextStyle: UIFontTextStyle.title3).pointSize

    // On iPad, landscape text is larger than preferred font size
    var portraitMultiplier = CGFloat(1.0)
    var landscapeMultiplier = CGFloat(0.5)

    // iPhone text is shrunken
    if UIDevice.current.model.range(of: "iPhone") != nil {
        portraitMultiplier = CGFloat(3.0)
        landscapeMultiplier = CGFloat(1.5)
    }

    // Start HTML header text
    let patternText = "<html> <head> <style> "

    // Match Dynamic Type for this page.
    + "body { background-color: \(backgroundColor);} "
    + "@media all and (orientation:portrait) {img {max-width: 90%; height: auto;} "
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize * portraitMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) } "
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h3Size * portraitMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } } "
    + "@media all and (orientation:landscape) {img {max-width: 65%; height: auto;}"
    + "p, li { font: -apple-system-body; font-family: Georgia, serif; font-size:calc(\(bodySize * landscapeMultiplier)px + 1.0vw); font-weight: normal; color: \(fontColor) }"
    + "h1 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h1Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h2 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h2Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } "
    + "h3, h4 { font: -apple-system-headine; font-family: Verdana, sans-serif; font-size:calc(\(h3Size * landscapeMultiplier)px + 1.0vw); font-weight: bold; color: \(headFontColor) } } </style>"
    + "</head><body>"
    + "<meta name=\"viewport\" content=\"width: device-width\">"

    return patternText
```

```
}
```

## Section 122.2: Get the Current Content Size

**Swift**

```swift
UIApplication.sharedApplication().preferredContentSizeCategory
```

**Objective-C**

```objc
[UIApplication sharedApplication].preferredContentSizeCategory;
```

This returns a content size category constant, or an accessibility content size category constant.

## Section 122.3: Handling Preferred Text Size Change Without Notifications on iOS 10

UILabel, UITextField, & UITextView classes have a new property starting from iOS 10 for automatically resizing their font when a user changes their preferred reading size named adjustsFontForContentSizeCategory.

**Swift**

```swift
@IBOutlet var label:UILabel!

if #available(iOS 10.0, *) {
    label.adjustsFontForContentSizeCategory = true
} else {
    // Observe for UIContentSizeCategoryDidChangeNotification and handle it manually
    // since the adjustsFontForContentSizeCategory property isn't available.
}
```

## Section 122.4: Text Size Change Notification

You can register for notifications of when the device text size is changed.

**Swift**

```swift
NSNotificationCenter.defaultCenter().addObserver(self, selector: #selector(updateFont), name:
name:UIContentSizeCategoryDidChangeNotification, object: nil)
```

**Objective-C**

```objc
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(updateFont)
name:UIContentSizeCategoryDidChangeNotification object:nil];
```

The notification userInfo object contains the new size under UIContentSizeCategoryNewValueKey.

# Chapter 123: SWRevealViewController

## Section 123.1: Setting up a basic app with SWRevealViewController

Create a basic application with single view application template with swift as language

Add `SWRevealViewController.h` and `SWRevealViewController.m`

then click on Create Bridging Header button
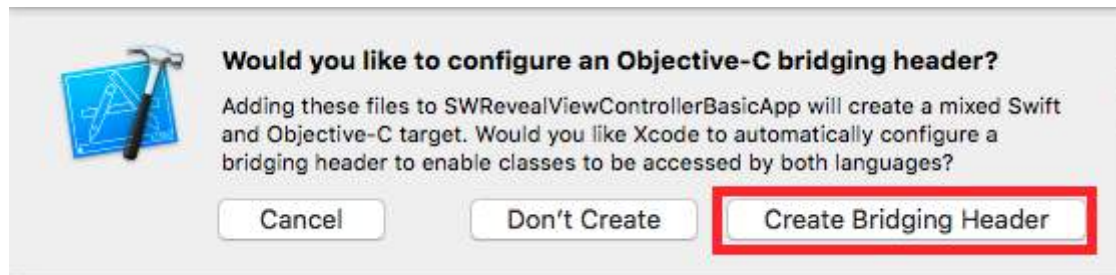


and add

```
#import "SWRevealViewController.h"
```

on the Bridging header

Then select viewController on storyboard and change class to `SWRevealViewController`
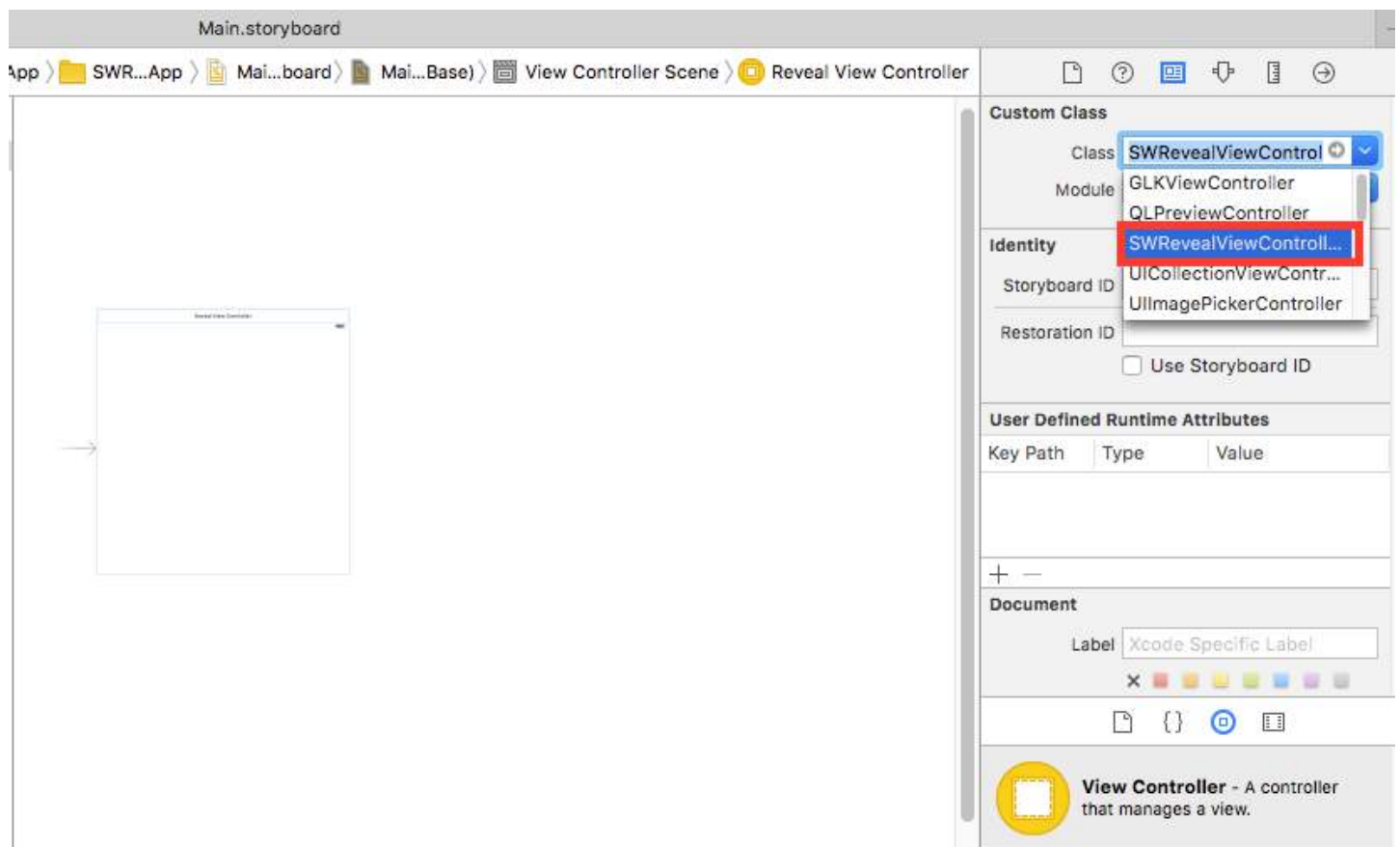


Then rename the viewController on files to MainViewController and add new ViewController with RightViewController name
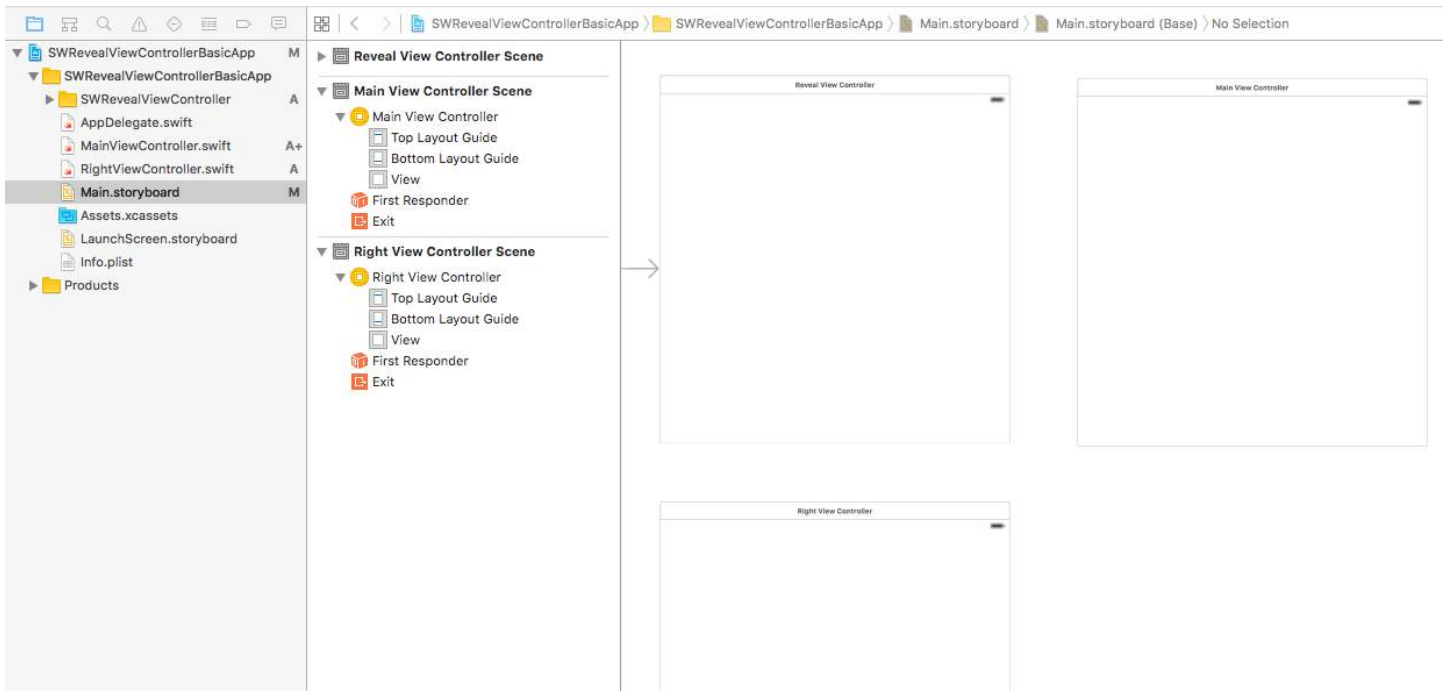
then we add two segues from SWRevealViewController to MainViewController and from SWRevealViewController to RightViewController, then we need to select the first (from SWRevealViewController to MainViewController) and edit properties

on identifier set `sw_front` on Class set `SWRevealViewControllerSegueSetController`
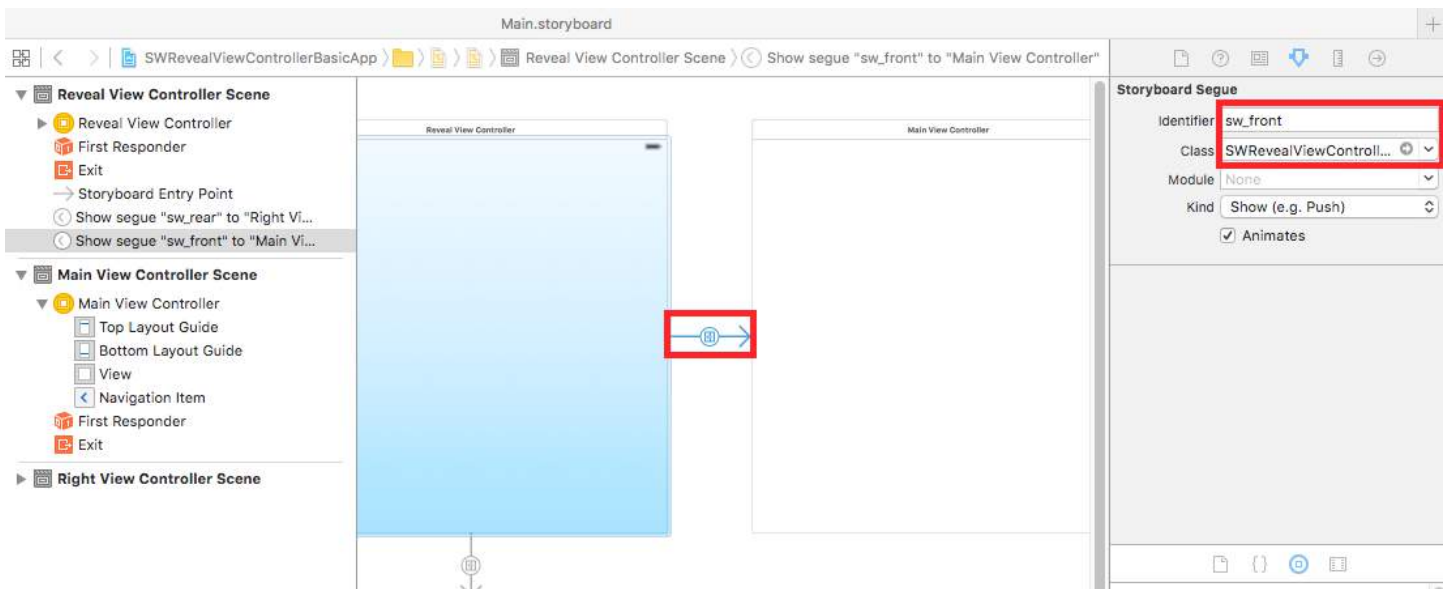


after this we need to do the same with the segue (from SWRevealViewController to RightViewController)

on identifier set `sw_rear` on Class set `SWRevealViewControllerSegueSetController`
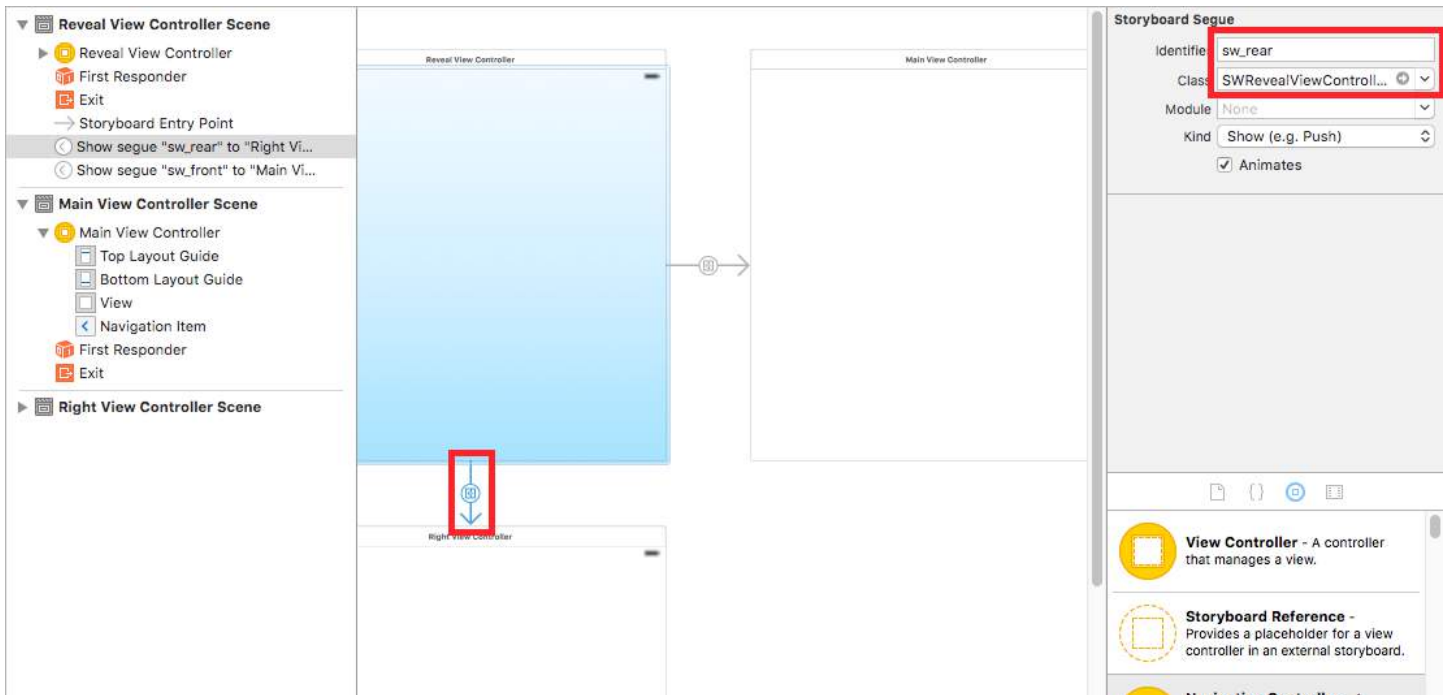
then on MainViewController add this line on `viewDidLoad` method

```
self.view.addGestureRecognizer(self.revealViewController().panGestureRecognizer());
```

And this is all, you have a basic app with SWRevealViewController integrated, you can swipe to right to show `RightViewController` as lateral menu

# Chapter 124: DispatchGroup

Related topics:

Grand Central Dispatch

Concurrency

## Section 124.1: Introduction

Suppose you have multiple threads running. Each thread is doing one task. You want to get notified either on the mainThread OR another thread, when all the task-threads are completed.

The simplest solution to such a problem is a `DispatchGroup`.

When using a `DispatchGroup`, for each request, you `enter` the group and for each completed request, you `leave` the group.

When there are no longer requests in the group, you will be `notify` (notified).

**Usage:**

```swift
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()


        let dispatchGroup = DispatchGroup() //Create a group for the tasks.
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //Process Response..

            dispatchGroup.leave() //Leave the group for the first task.
        }


        dispatchGroup.enter()  //Enter the group for the second task.

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in

            //Process Response..

            dispatchGroup.leave()  //Leave the group for the second task.
        }


        //Get notified on the main thread/queue.. when ALL of the tasks above has been completed.
        dispatchGroup.notify(queue: DispatchQueue.main) {
```

```
            print("Every task is complete")

        }


        //Start the tasks.
        firstTask.resume()
        secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

With the above, you don't have to `wait` infinitely until all the tasks are completed. You can display a loader BEFORE all the tasks have started and dismiss the loader AFTER all tasks are completed. This way, your main thread does not get blocked and your code remains clean.

Now suppose you also wanted the tasks to be `ordered` or add their responses to an array sequentially. You could do the following:

```
import UIKit

//Locking mechanism..
func synchronized(_ lock: AnyObject, closure: () -> Void) {
    objc_sync_enter(lock)
    closure()
    objc_sync_exit(lock)
}

class ViewController: UIViewController {

    let lock = NSObject() //Object to lock on.
    var responseArray = Array<Data?>() //Array of responses.

    override func viewDidLoad() {
        super.viewDidLoad()

        let dispatchGroup = DispatchGroup()
        let session: URLSession = URLSession.shared

        dispatchGroup.enter() //Enter the group for the first task.

        let firstTask = session.dataTask(with: URLRequest(url: URL(string:
"https://stackoverflow.com")!)) { (data, response, error) in

            //Process Response..

            synchronized(self.lock, closure: { () -> Void in
                self.responseArray[0] = data ?? nil
            })

            dispatchGroup.leave() //Leave the group for the first task.
        }


        dispatchGroup.enter()  //Enter the group for the second task.

        let secondTask = session.dataTask(with: URLRequest(url: URL(string: "https://google.ca")!))
{ (data, response, error) in
```

```
            //Process Response..

            synchronized(self.lock, closure: { () -> Void in
                self.responseArray[1] = data ?? nil
            })

            dispatchGroup.leave()  //Leave the group for the second task.
        }


        //Get notified on the main thread.. when ALL of the requests above has been completed.
        dispatchGroup.notify(queue: DispatchQueue.main) {

            print("Every task is complete..")

            for i in 0..<self.responseArray.count {

                if self.responseArray[i] == nil {
                    print("Request #\(i) Failed.\n")
                }
                else {
                    print("Request #\(i) Succeeded.\n")
                }
            }
        }

        //Two tasks added to the array. Responses are assumed nil until they complete.
        self.responseArray.append(nil)
        self.responseArray.append(nil)

        //Start the tasks.
        firstTask.resume()
        secondTask.resume()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

**Notes**

Every entry must have an exit in a `DispatchGroup`. If you forget to `leave` after `entering`, you are setting yourself up.
You will NEVER be notified when the tasks are completed.

The amount of `enter` must equal the amount of `leave`.

# Chapter 125: GCD (Grand Central Dispatch)

Grand Central Dispatch (GCD) is Apple's answer to multithreading. It is a lightweight framework for performing tasks synchronously or asynchronously in queues and handles CPU threads for you behind the scenes.

Related Topic: Concurrency

## Section 125.1: Dispatch Semaphore

> DispatchSemaphore provides an efficient implementation of a traditional counting semaphore, which can be used to control access to a resource across multiple execution contexts.

A scenario for when to use a semaphore could be if you are doing some file reading/writing, if multiple tasks are trying to read and write from file at the same time, it could increase your performance to make each task wait its turn so as to not overburden the I/O controller.

**Swift 3**

```swift
func do2TasksAtATime () {
    print("starting long running tasks (2 at a time)")
    let sem = DispatchSemaphore(value: 2)           //this semaphore only allows 2 tasks to run at
the same time (the resource count)
    for i in 0...7 {                                //launch a bunch of tasks
        DispatchQueue.global().async {              //run tasks on a background thread
            sem.wait()                              //wait here if no resources available
            sleep(2)                                //do some long task eg file access (here we
are just sleeping for a 2 seconds for demonstration purposes)
            print("long task \(i) done! \(Date())")
            sem.signal()                            //let the semaphore know this resource is now
available
        }
    }
}
```

Example output: (notice the time stamps)

```
starting long running tasks (2 at a time)
long task 0 done! 2017-02-16 07:11:53 +0000
long task 1 done! 2017-02-16 07:11:53 +0000
long task 2 done! 2017-02-16 07:11:55 +0000
long task 3 done! 2017-02-16 07:11:55 +0000
long task 5 done! 2017-02-16 07:11:57 +0000
long task 4 done! 2017-02-16 07:11:57 +0000
long task 6 done! 2017-02-16 07:11:59 +0000
long task 7 done! 2017-02-16 07:11:59 +0000
```

For more info, refer to the [Apple Docs](#)

## Section 125.2: Dispatch Group

> DispatchGroup allows for aggregate synchronization of work. You can use them to submit multiple different work items and track when they all complete, even though they might run on different queues. This behavior can be helpful when progress can't be made until all of the specified tasks are complete.

A Scenario when this could be useful is if you have multiple webservice calls that all need to finish before continuing. For example, you need to download multiple sets of data that needs to be processed by some function. You have to wait for all webservices to complete before calling the function to process all the received data.

**Swift 3**

```
func doLongTasksAndWait () {
    print("starting long running tasks")
    let group = DispatchGroup()          //create a group for a bunch of tasks we are about to do
    for i in 0...3 {                     //launch a bunch of tasks (eg a bunch of webservice calls
that all need to be finished before proceeding to the next ViewController)
        group.enter()                    //let the group know that something is being added
        DispatchQueue.global().async {   //run tasks on a background thread
            sleep(arc4random() % 4)      //do some long task eg webservice or database lookup (here
we are just sleeping for a random amount of time for demonstration purposes)
            print("long task \(i) done!")
            group.leave()                //let group know that the task is finished
        }
    }
    group.wait()                         //will block whatever thread we are on here until all the
above tasks have finished (so maybe don't use this function on your main thread)
    print("all tasks done!")
}
```

Alternatively, if you do not want to wait for the groups to finish, but instead want to run a function once all the tasks have completed, use the `notify` function in place of the `group.wait()`

```
group.notify(queue: DispatchQueue.main) { //the queue: parameter is which queue this block will run
on, if you need to do UI updates, use the main queue
    print("all tasks done!")             //this will execute when all tasks have left the group
}
```

Example output:

```
starting long running tasks
long task 0 done!
long task 3 done!
long task 1 done!
long task 2 done!
all tasks done!
```

For more info, refer to the Apple Docs or the related topic

# Section 125.3: Getting the Main Queue

The main queue is the dispatch queue in which all the UI updates take place and the code involving UI changes are placed.

You need to get to the main queue in order to update UI on completion of an asynchronous process like `NSURLSession`

There are two types of main queue calls `synchronous` and `asynchronous`. When you invoke something `synchronously`, it means that the thread that initiated that operation will wait for the task to finish before continuing. `Asynchronous` means that it will not wait.

**Code Objective-C**

**Synchronous Main Queue call**

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

**Asynchronous Main Queue call**

```
dispatch_async(dispatch_get_main_queue(), ^{
    // do work here to Usually to update the User Interface
});
```

**SWIFT 3**

**Asynchronous Main Queue call**

```
DispatchQueue.main.async {

}
```

**Synchronous Main Queue call**

```
DispatchQueue.main.sync {

}
```

# Section 125.4: Create a dispatch queue

You can create your own queue using `dispatch_queue_create`

**Objective-C**

```
dispatch_queue_t queue = dispatch_queue_create("com.example.myqueue",  DISPATCH_QUEUE_SERIAL);
```

**Swift**

```
// Before Swift 3
let queue = dispatch_queue_create("com.example.myqueue", DISPATCH_QUEUE_SERIAL)
// Swift 3
let queue = DispatchQueue(label: "com.example.myqueue") //default is serial queue, unless
.concurrent is specified as an attribute otherwise
```

# Section 125.5: Serial vs Concurrent Dispatch Queues

**Swift 3**

**Serial Queue**

```
func serialQueues () {
    let serialQueue = DispatchQueue(label: "com.example.serial") //default queue type is a serial
queue
    let start = Date ()
    for i in 0...3 {                                        //launch a bunch of tasks
        serialQueue.async {                                //run tasks on a background
thread, using our serial queue
            sleep(2)                                       //do some long task eg webservice
or database lookup
            let timeTaken = Date().timeIntervalSince(start)
```

```
            print("serial long task \(i) done! total time taken: \(timeTaken)")
        }
    }
}
```

Example output:

```
serial long task 0 done! total time taken: 2.07241100072861
serial long task 1 done! total time taken: 4.16347700357437
serial long task 2 done! total time taken: 6.23209798336029
serial long task 3 done! total time taken: 8.30682599544525
```

**Concurrent Queue**

```
func concurrentQueues () {
    let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)
//explicitly specify the queue to be a concurrent queue
    let start = Date ()
    for i in 0...3 {              //launch a bunch of tasks
        concurrentQueue.async { //run tasks on a background thread, using our concurrent queue
            sleep(2)             //do some long task eg webservice or database lookup
            let timeTaken = Date().timeIntervalSince(start)
            print("concurrent long task \(i) done! total time taken: \(timeTaken)")
        }
    }
}
```

Example output:

```
concurrent long task 3 done! total time taken: 2.07092100381851
concurrent long task 0 done! total time taken: 2.07087397575378
concurrent long task 2 done! total time taken: 2.07086700201035
concurrent long task 1 done! total time taken: 2.07089096307755
```

**Discussion**

As we can see from the examples above, a serial queue will complete each task in the order they are submitted to the queue. Each task will wait for the previous task to finish before executing. As for the concurrent queue, each task does not wait on the others in the queue and executes as soon as possible; the advantage is that all tasks on the queue will run at the same time on separate threads, making a concurrent queue take less time than a serial queue.

If order of execution of tasks is not important, always use a concurrent queue for the best efficiency.

# Chapter 126: Size Classes and Adaptivity

## Section 126.1: Trait Collections

In an iOS app, your user interface can take on one of a few different general shapes and sizes. These are defined using **size classes**, which are available through a view or view controller's **trait collection**.

Apple defines two size classes: **regular** and **compact**. Each of these size classes are available on both axes of the device (**horizontal** and **vertical**). Your app may exist in any these four states throughout its lifetime. As a shorthand, developers often describe a size class combination by saying or writing the two size classes, with the horizontal axis first: "Compact/Regular" describes an interface that is horizontally compact but vertically regular.

In your app, use methods on the UITraitEnvironment protocol to check your current size class and respond to changes:

```
class MyViewController: UIViewController {
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("Horizontal size class: \(traitCollection.horizontalSizeClass)")
        print("Vertical size class: \(traitCollection.verticalSizeClass)")
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        print("Trait collection changed; size classes may be different.")
    }
}
```

Both UIView and UIViewController conform to UITraitEnvironment, so you can look at your current trait collection and handle changes in subclasses of either.

## Section 126.2: Updating Auto Layout with Trait Collection Changes

Making an app **adaptive** – that is, responding to size class changes by changing your layout – often involves lots of help from the Auto Layout system. One of the primary ways apps become adaptive is by updating the active Auto Layout constraints when a view's size class changes.

For example, consider an app that uses a UIStackView to arrange two UILabels. We might want these labels to stack on top of each other in horizontally compact environments, but sit next to each other when we have a little more room in horizontally regular environments.

```
class ViewController: UIViewController {
    var stackView: UIStackView!

    override func viewDidLoad() {
        super.viewDidLoad()

        stackView = UIStackView()
        for text in ["foo", "bar"] {
            let label = UILabel()
            label.translatesAutoresizingMaskIntoConstraints = false
            label.text = text
            stackView.addArrangedSubview(label)
        }
```

```
        view.addSubview(stackView)
        stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
        stackView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        updateAxis(forTraitCollection: traitCollection)
    }

    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)
        updateAxis(forTraitCollection: traitCollection)
    }

    private func updateAxis(forTraitCollection traitCollection: UITraitCollection) {
        switch traitCollection.horizontalSizeClass {
        case .regular:
            stackView.axis = .horizontal
        case .compact:
            stackView.axis = .vertical
        case .unspecified:
            print("Unspecified size class!")
            stackView.axis = .horizontal
        }
    }
}
```

## Section 126.3: Supporting iOS Multitasking on iPad

A key piece of adaptivity in a modern iOS app is supporting multitasking on iPad. By default, apps created in Xcode 7 and newer will be configured to support multitasking: they'll have a LaunchScreen.storyboard file that uses Auto Layout.

The easiest way for existing apps to opt in to multitasking is to create such a storyboard, then set it as the project's Launch Screen:
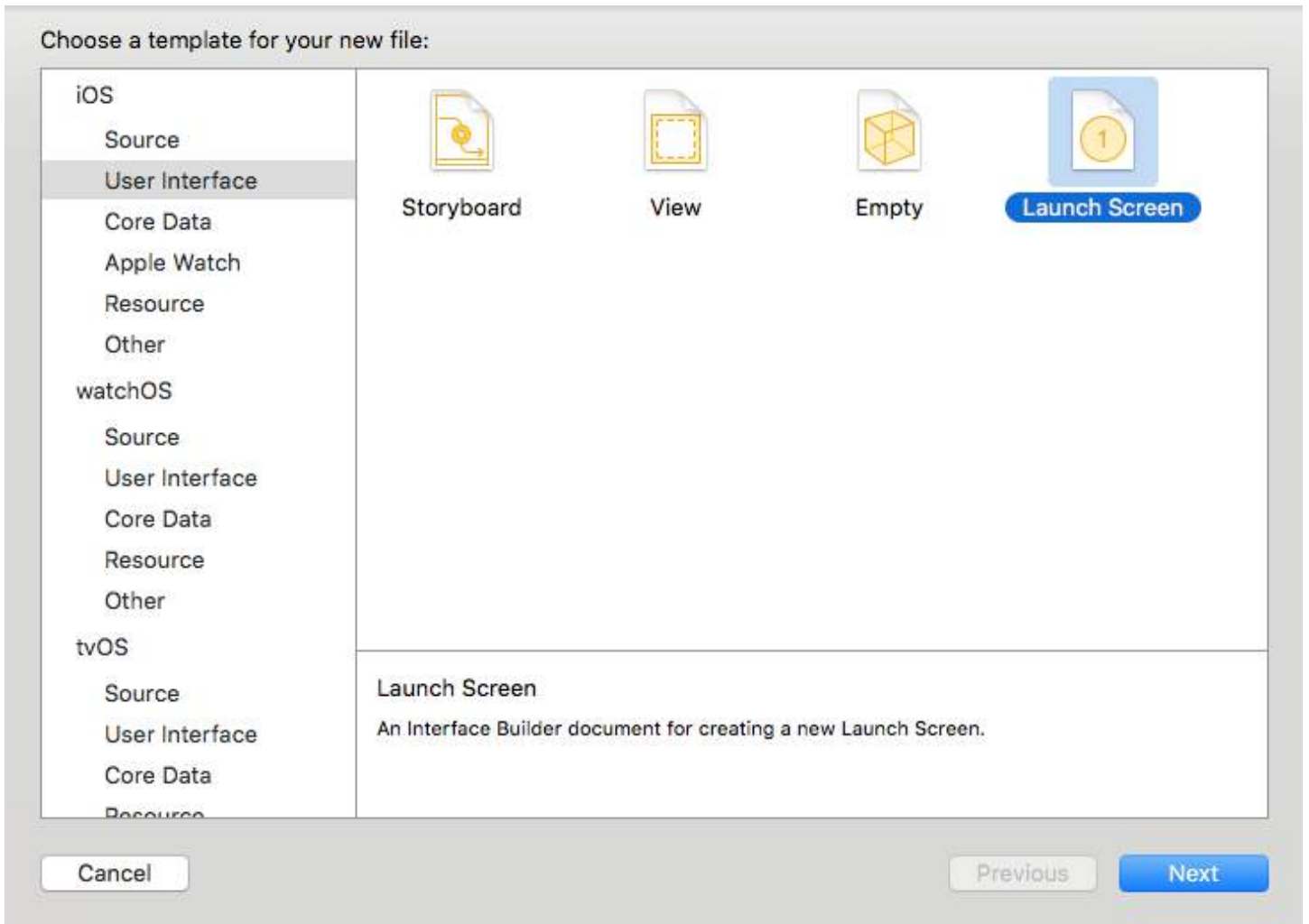
Once your app supports iPad multitasking, audit existing views and view controllers to make sure that they use Auto Layout and can support a variety of size class combinations.

# Chapter 127: IBOutlets

## Section 127.1: Using an IBOutlet in a UI Element

In general, IBOutlets are used to connect an user interface object to another object, in this case a UIViewController. The connection serves to allow for the object to be affected my code or events programmatically. This can be done simply by using the assistant from a storyboard and control-clicking from the element to the view controller's .h property section, but it can also be done programmatically and manually connecting the IBOutlet code to the "connections" tab of the object the utility bar on the right. Here is an objective-c example of a UIViewController with a label outlet:

```objectivec
//ViewController.h
#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

//This is the declaration of the outlet
@property (nonatomic, weak) IBOutlet UILabel *myLabel;

@end

//ViewController.m
#import "ViewController.h"

@implementation ViewController

@synthesize myLabel;

-(void) viewDidLoad {

    [super viewDidLoad];
    //Editing the properties of the outlet
    myLabel.text = @"TextHere";

}

@end
```

And swift:

```swift
import UIKit
class ViewController: UIViewController {
    //This is the declaration of the outlet
    @IBOutlet weak var myLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        //Editing the properties of the outlet
        myLabel.text = "TextHere"
    }
}
```

The connection between the storyboard object, and the programmed object can be verified as connected if the dot to the left of the declaration of the outlet in the .h is filled. An empty circle implied an incomplete connection.

# Chapter 128: AWS SDK

## Section 128.1: Upload an image or a video to S3 using AWS SDK

Before starting with the example I'd recommend to create a Singleton with a delegate class member so you could achieve a use case of uploading a file in the background and let the user keep using your app while the files are being uploaded even when the app is the background.

Let's start, first, we should create an enum that represent the S3 configuration:

```
enum S3Configuration : String
{
    case IDENTITY_POOL_ID   = "YourIdentityPoolId"
    case BUCKET_NAME        = "YourBucketName"
    case CALLBACK_KEY       = "YourCustomStringForCallBackWhenUploadingInTheBackground"
    case CONTENT_TYPE_IMAGE = "image/png"
    case CONTENT_TYPE_VIDEO = "video/mp4"
}
```

Now, we should set the credentials when your app launch for the first time, thus, we should set them inside the `AppDelegate` at the `didFinishLaunchingWithOptions` method (pay attention that you should set your region at the `regionType` param):

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool
{
  let credentialProvider = AWSCognitoCredentialsProvider(regionType: .EUWest1, identityPoolId:
S3Configuration.IDENTITY_POOL_ID.rawValue)
  let configuration = AWSServiceConfiguration(region: .EUWest1, credentialsProvider:
credentialProvider)
  AWSS3TransferUtility.registerS3TransferUtilityWithConfiguration(configuration, forKey:
S3Configuration.CALLBACK_KEY.rawValue)
}
```

Since we are already inside the AppDelegate, we should implement the background callback that is handled by the AWS SDK:

```
func application(application: UIApplication, handleEventsForBackgroundURLSession identifier:
String, completionHandler: () -> Void)
{
    //  Will print the identifer you have set at the enum: .CALLBACK_KEY
    print("Identifier: " + identifier)
    //  Stores the completion handler.
    AWSS3TransferUtility.interceptApplication(application,
                                    handleEventsForBackgroundURLSession: identifier,
                                    completionHandler: completionHandler)
}
```

Now, when the user will move the app to the background your upload will continue the actual upload.

In order to upload the file using the AWS SDK we will have to write the file to the device and give the SDK the actual path. For the sake of the example, imagine we have a UIImage (could be a video also..) and we will write it to a temp folder:

```
// Some image....
```

```
let image = UIImage()
let fileURL = NSURL(fileURLWithPath: NSTemporaryDirectory()).URLByAppendingPathComponent(fileName)
let filePath = fileURL.path!
let imageData = UIImageJPEGRepresentation(image, 1.0)
imageData!.writeToFile(filePath, atomically: true)
```

FileURL and fileName will be used for the actual uploading later.

There are 2 closures we will have to define that are provided by the AWS SDK,

1. `AWSS3TransferUtilityUploadCompletionHandlerBlock` - A closure that notifies when the upload is done (or not)
2. `AWSS3TransferUtilityUploadProgressBlock` - A closure that notifies each byte sent

If you plan to have a Singleton you should define those types as class members. The implementation should look like this:

```
var completionHandler : AWSS3TransferUtilityUploadCompletionHandlerBlock? =
    { (task, error) -> Void in

        if ((error) != nil)
        {
          print("Upload failed")
        }
        else
        {
          print("File uploaded successfully")
        }
    }

var progressBlock : AWSS3TransferUtilityUploadProgressBlock? =
    { [unowned self] (task, bytesSent:Int64, totalBytesSent:Int64,  totalBytesExpectedToSend:Int64)
-> Void in

    let progressInPercentage = Float(Double(totalBytesSent) / Double(totalBytesExpectedToSend)) *
100
    print(progressInPercentage)
    }
```

**NOTE:** If you are using a Singleton you might want to define a delegate that will report back with progress or when the file is done. If you are not using a Singleton you can create a static method that would have the relevant types:

```
    static func uploadImageToS3(fileURL : NSURL,
                         fileName : String,
            progressFunctionUpdater : Float -> Void,
                       resultBlock : (NSError?) -> Void)
{
 //    Actual implementation .....
 //    ...
 //    ...
}
```

1. `progressFunctionUpdater` - will report back to a function with progress.
2. `resultBlock` - If you return nil then upload was successfully else, you send the error object

Ladies and gentlemen, the actual upload:

```
    let fileData = NSData(contentsOfFile: fileURL.relativePath!)
```

```swift
        let expression = AWSS3TransferUtilityUploadExpression()
        expression.uploadProgress = progressBlock

        let transferUtility =
AWSS3TransferUtility.S3TransferUtilityForKey(S3Configuration.CALLBACK_KEY.rawValue)

        transferUtility?.uploadData(fileData!,
            bucket: S3Configuration.BUCKET_NAME.rawValue,
            key: fileName,
            contentType: S3Configuration.CONTENT_TYPE_IMAGE.rawData,
            expression: expression,
            completionHander: completionHandler).continueWithBlock
            { (task : AWSTask) -> AnyObject? in

                if let error = task.error
                {
                    print(error)
                }
                if let exception = task.exception
                {
                    print("Exception: " + exception.description)
                }
                if let uploadTask = task.result as? AWSS3TransferUtilityUploadTask
                {
                    print("Upload started...")
                }

                return nil
        }
```

Happy S3 uploading :)

# Chapter 129: Debugging Crashes

## Section 129.1: Debugging EXC_BAD_ACCESS

`EXC_BAD_ACCESS` means the process tried to access memory in an invalid way, like dereferencing a `NULL` pointer or writing to read-only memory. This is the hardest kind of crash to debug, because it usually does not have an error message, and some crashes can be *very* difficult to reproduce and/or occur in code completely unrelated to the problem. This error is very rare in Swift, but if it occurs, you can often get easier-to-debug crashes by reducing compiler optimizations.

Most `EXC_BAD_ACCESS` errors are caused by trying to dereference a `NULL` pointer. If this is the case, the address listed in the red arrow will usually be a hexadecimal number that is lower than a normal memory address, often `0x0`. Set breakpoints in the debugger or add occasional `printf/NSLog` statements to find out why that pointer is `NULL`.

An `EXC_BAD_ACCESS` that occurs less reliably or makes no sense at all could be the result of a memory management problem. Common problems that can cause this are:

- Using memory that has been deallocated
- Trying to write past the end of a C array or other kind of buffer
- Using a pointer which has not been initialized

In the Diagnostics section of the Scheme Editor, Xcode includes a few useful tools to help debug memory problems:
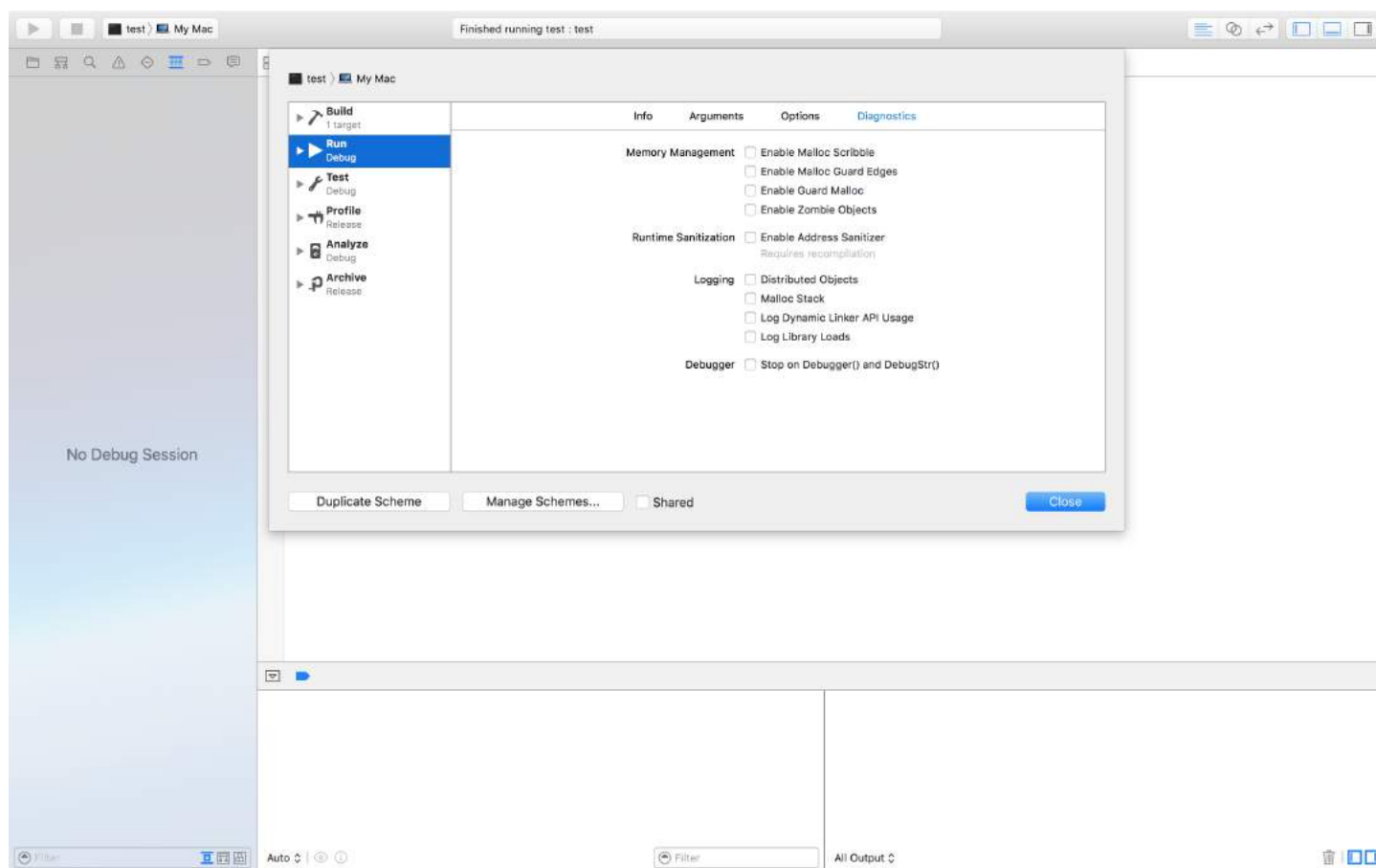


The Address Sanitizer adds lots of checks that will stop the app whenever memory problems occur and provide a helpful error message detailing exactly what happened. Zombie Objects detects problems with deallocated Objective-C objects, but you shouldn't get these kinds of problems with Automatic Reference Counting turned on.

# Section 129.2: Finding information about a crash

When your app crashes, Xcode will enter the debugger and show you more information about the crash:
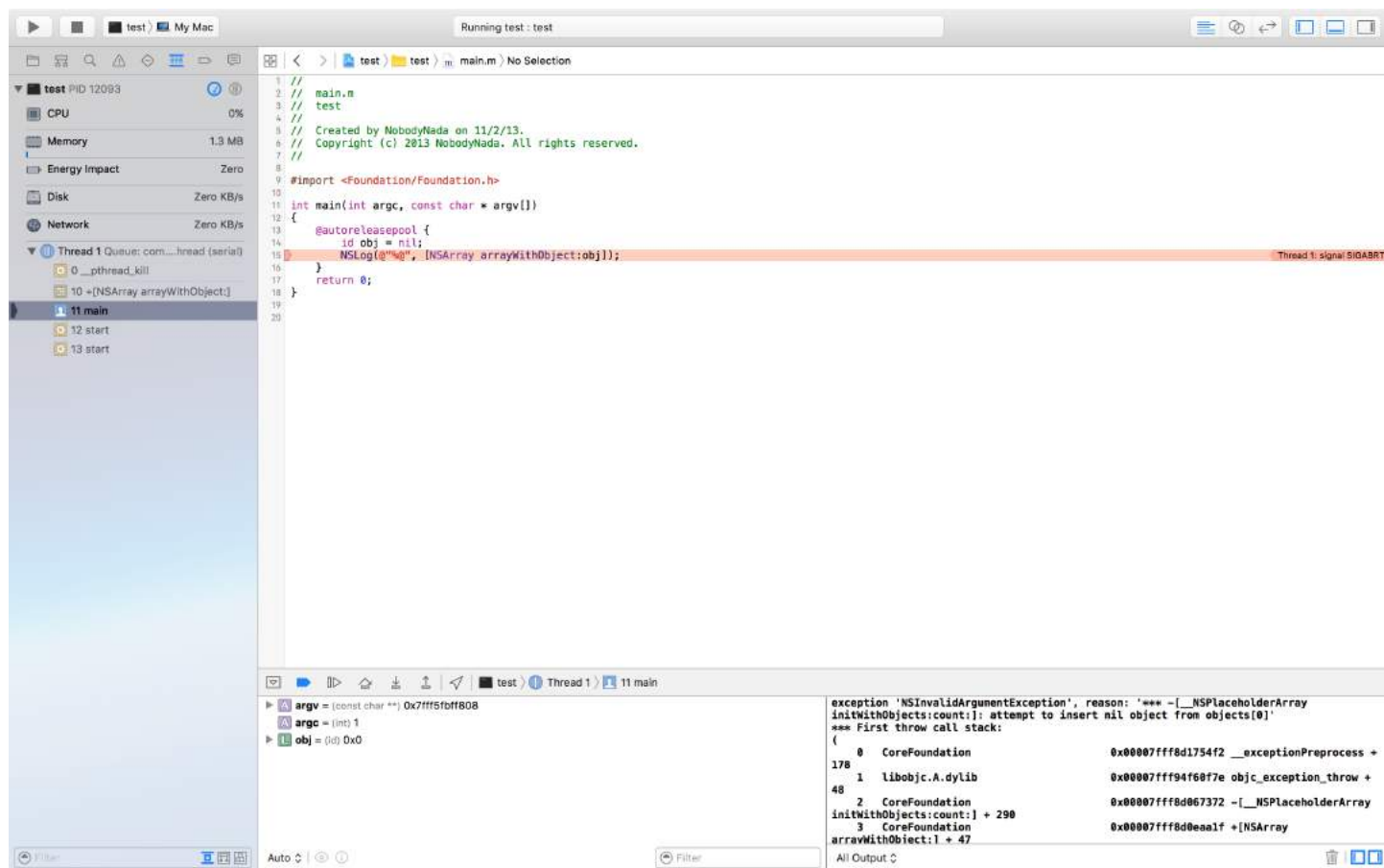


The most important parts are:

**The red arrow**

The red arrow displays which line of code crashed & why it crashed.

**The debugger console**

Many crashes log more information to the debugger console. It should automatically appear when the app crashes, but if it's not there, show the debugger by selecting the ⬜ button in the top-right corner of Xcode, and show the console by clicking the ⬜ button in the bottom-right corner of the debugger.

**The stack trace**

The stack trace lists the functions the program came from before it got to the code that crashed.

Part of the stack trace is displayed in the Debug Navigator on the left of the screen, and the debugger controls allow you to select a stack frame to view in the debugger:
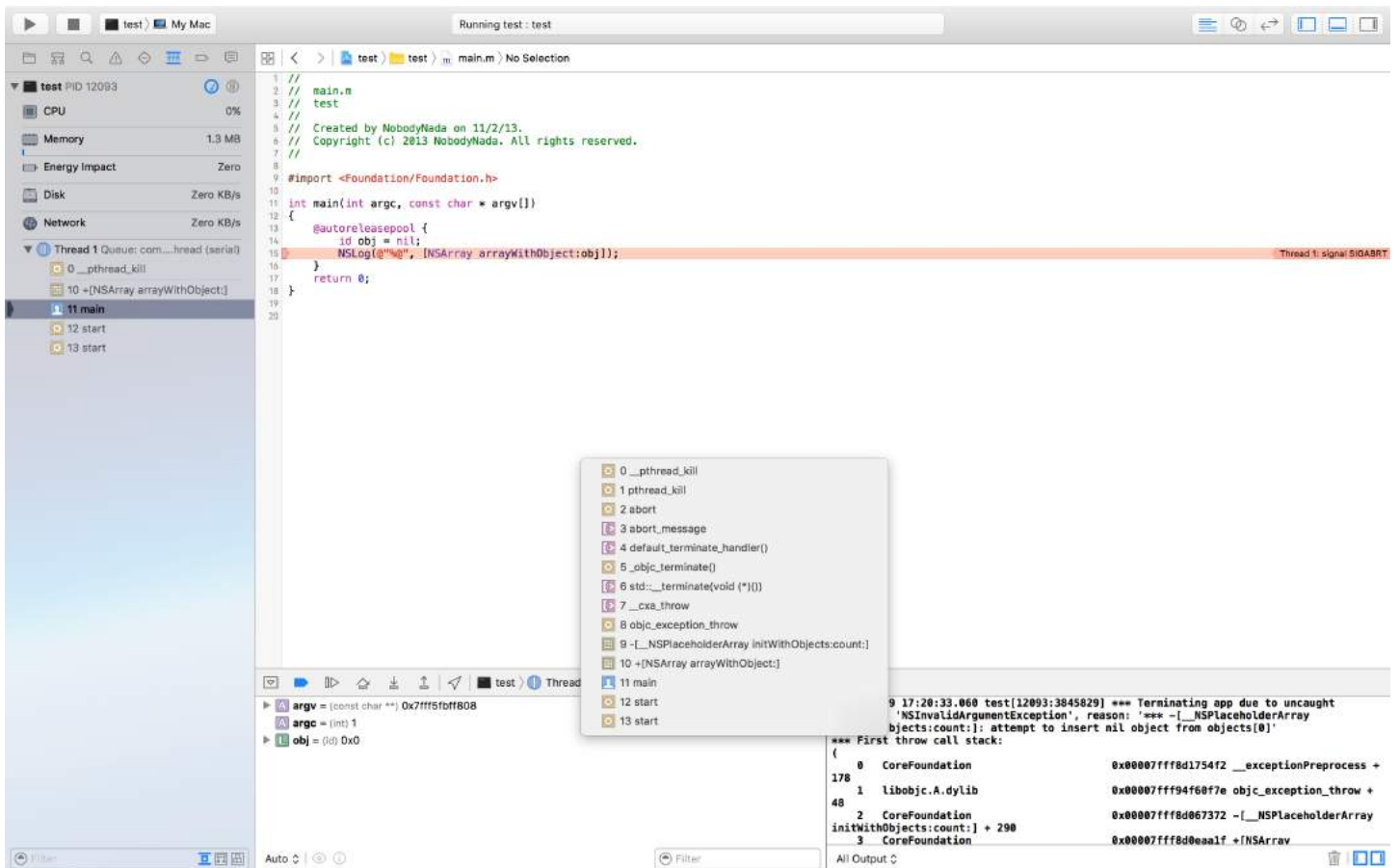
If you enter the bt command at the (lldb) prompt in the debugger and press return , you will get a textual representation of the stack trace that you can copy and paste:

```
(lldb) bt
* thread #1: tid = 0x3aaec5, 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10, queue =
'com.apple.main-thread', stop reason = signal SIGABRT
    frame #0: 0x00007fff91055f06 libsystem_kernel.dylib`__pthread_kill + 10
    frame #1: 0x000000010008142d libsystem_pthread.dylib`pthread_kill + 90
    frame #2: 0x00007fff96dc76e7 libsystem_c.dylib`abort + 129
    frame #3: 0x00007fff8973bf81 libc++abi.dylib`abort_message + 257
    frame #4: 0x00007fff89761a47 libc++abi.dylib`default_terminate_handler() + 267
    frame #5: 0x00007fff94f636ae libobjc.A.dylib`_objc_terminate() + 103
    frame #6: 0x00007fff8975f19e libc++abi.dylib`std::__terminate(void (*)()) + 8
    frame #7: 0x00007fff8975ec12 libc++abi.dylib`__cxa_throw + 121
    frame #8: 0x00007fff94f6108c libobjc.A.dylib`objc_exception_throw + 318
    frame #9: 0x00007fff8d067372 CoreFoundation`-[__NSPlaceholderArray initWithObjects:count:] +
290
    frame #10: 0x00007fff8d0eaa1f CoreFoundation`+[NSArray arrayWithObject:] + 47
  * frame #11: 0x0000000100001b54 test`main(argc=1, argv=0x00007fff5fbff808) + 68 at main.m:15
    frame #12: 0x00007fff8bea05ad libdyld.dylib`start + 1
    frame #13: 0x00007fff8bea05ad libdyld.dylib`start + 1
```

## Section 129.3: Debugging SIGABRT and EXC_BAD_INSTRUCTION crashes

A SIGABRT or an EXC_BAD_INSTRUCTION usually means the app crashed itself intentionally because some check failed. These should log a message to the debugger console with more information; check there for more information.

Many SIGABRTs are caused by uncaught Objective-C exceptions. There are a *lot* of reasons exceptions can be thrown, and they will *always* log a lot of helpful information to the console.

- `NSInvalidArgumentException`, which means the app passed an invalid argument to a method
- `NSRangeException`, which means the app tried to access an out-of-bounds index of an object such as an `NSArray` or an `NSString`
- `NSInternalInconsistencyException` means an object discovered it was in an unexpected state.
- `NSUnknownKeyException` usually means you have a bad connection in an XIB. Try some of the answers to [this question](#).

# Chapter 130: CloudKit

## Section 130.1: Registering app for use with CloudKit

What you need is to get an entitlements file so the app can access your iCloud and write records using CloudKit.

Follow the steps to grant access to iCloud from your app:

1- Select the project in the Project Navigator, and then open the General tab.

2- In the Identity section, set your developer Apple ID to the Team dropdown menu. (If it is not available, add it in Xcode menu -> Preferences -> Accounts.

3- Go to Capabilities tab in the project properties and turn iCloud on. Then, select "Key-Value Storage" and "CloudKit".
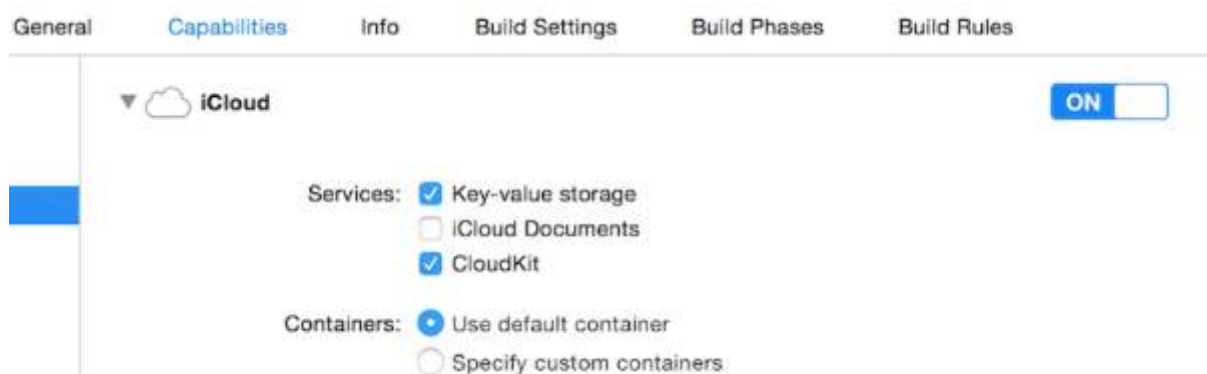


4- Make sure these items are checked:



If all of the items are checked, then your app is ready to use CloudKit.

## Section 130.2: Using CloudKit Dashboard

All the records create using CloudKit-related code can be previewed, edited and even removed in CloudKit Dashboard. To access CloudKit Dashboard, go here.

There are several parts in the dashboard:

- Record Types (which will be discussed later)
- Security Roles (which is where you can set databases as public or private)
- Subscription Types (which your app could register for Apple Push Notifications (APNs) to notify you when a record is changed)

**Record Types**

Here, you get a list of all the existing record types in the app. When you first open CloudKit Dashboard for an app, there's a record type called Users there, which you can use it or just delete it and use your own.

In this page, you could make your data typed manually. Of course, in most cases this is pointless, because iOS SDK

can handle it way better than the dashboard, but the functionality is also there if you prefer. The most use of this page is for previewing types.

# Section 130.3: Saving data to CloudKit

To save date to CloudKit, we must make:

- A CKRecordID (the key of your unique record)
- A CKRecord (which includes data)

**Making a record key**

To ensure that every new record identifier is unique, we use the current *timestamp*, which is unique. We get the timestamp using NSDate's method timeIntervalSinceReferenceDate(). It is in form of ###.### (# are numbers), which we will use the integer part. To do this, we split the string:

**Swift**

```swift
let timestamp = String(format: "%f", NSDate.timeIntervalSinceReferenceDate())
let timestampParts = timestamp.componentsSeparatedByString(".")
let recordID = CKRecordID(recordName: timestampParts[0])
```

**Making the record**

To make the record, we should specify the record type (explained in Using CloudKit Dashboard) as Users, the ID as the thing we made just now and the data. Here, we will add a sample text, a picture and the current date to the record:

**Swift**

```swift
let record = CKRecord(recordType: "Users", recordID: recordID)
record.setObject("Some Text", forKey: "text")
record.setObject(CKAsset(fileURL: someValidImageURL), forKey: "image")
record.setObject(NSDate(), forKey: "date")
```

**Objective-C**

```objc
CKRecord *record = [[CKRecord alloc] initWithRecordType: "Users" recordID: recordID];
[record setObject: "Some Text" forKey: "text"];
[record setObject: [CKAsset assetWithFileURL: someValidImageURL] forKey: "image"];
[record setObject: [[NSDate alloc] init] forKey: "date"];
```

> **Note**
>
> Here, we didn't add the UIImage directly to the record, because as mentioned in Remarks, image format isn't directly supported in CloudKit, so we have converted UIImage into CKAsset.

**Accessing the container**

**Swift**

```swift
let container = CKContainer.defaultContainer()
let database = container.privateCloudDatabase // or container.publicCloudDatabase
```

**Saving the records to CloudKit database**

**Swift**

```swift
database.saveRecord(record, completionHandler: { (_, error) -> Void in
    print(error ?? "")
})
```

# Chapter 131: GameplayKit

## Section 131.1: Generating random numbers

Although `GameplayKit` (which is introduced with iOS 9 SDK) is about implementing game logic, it could also be used to generate random numbers, which is very useful in apps and games.

Beside the `GKRandomSource.sharedRandom` which is used in the following chapters there are three additional types of GKRandomSource's out of the box.

- **GKARC4RandomSource** Which uses the the ARC4 algorithm
- **GKLinearCongruentialRandomSource** Which is a fast but not so random `GKRandomSource`
- **GKMersenneTwisterRandomSource** Which implements a MersenneTwister algorithm. It is slower but more random.

In the following chapter we only use the `nextInt()` method of a GKRandomSource. In addition to this there is the `nextBool() -> Bool` and the `nextUniform() -> Float`

**Generation**

First, import `GameplayKit`:

**Swift**

```
import GameplayKit
```

**Objective-C**

```
#import <GameplayKit/GameplayKit.h>
```

Then, to generate a random number, use this code:

**Swift**

```
let randomNumber = GKRandomSource.sharedRandom().nextInt()
```

**Objective-C**

```
int randomNumber = [[GKRandomSource sharedRandom] nextInt];
```

> **Note**
>
> The nextInt() function, when used without parameters, will return a random number between -2,147,483,648 and 2,147,483,647, including themselves, so we are not sure that it is always a positive or non-zero number.

**Generating a number from 0 to n**

To achieve this, you should give n to `nextIntWithUpperBound()` method:

**Swift**

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 10)
```

**Objective-C**

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 10];
```

This code will give us a number between 0 and 10, including themselves.

**Generating a number from m to n**

To do this you create a `GKRandomDistribution` object with a `GKRandomSource` and pass in the bounds. A `GKRandomDistribution` can be used to change the distribution behaviour like `GKGaussianDistribution` or `GKShuffledDistribution`.

After that the object can be used like every regular `GKRandomSource` since it does implement the `GKRandom` protocol too.

**Swift**

```
let randomizer = GKRandomDistribution(randomSource: GKRandomSource(), lowestValue: 0, highestValue:
6)
let randomNumberInBounds = randomizer.nextInt()
```

**Objective-C** *outdated*

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: n - m] + m;
```

For example, to generate a random number between 3 and 10, you use this code:

**Swift**

```
let randomNumber = GKRandomSource.sharedRandom().nextInt(upperBound: 7) + 3
```

**Objective-C** *outdated*

```
int randomNumber = [[GKRandomSource sharedRandom] nextIntWithUpperBound: 7] + 3;
```

# Section 131.2: GKEntity and GKComponent

An entity represents an object of a game like a player figure or an enemy figure. Since this object does not do much without arms and legs we can add the components to this. To create this system apple has the `GKEntity` and `GKComponent` classes.

Lets assume we have the following classe for the following chapters:

```
class Player: GKEntity{}
class PlayerSpriteComponent: GKComponent {}
```

**GKEntity**

An entity is a collection of components and offers several functions to add, remove and interact with components of it.

While we could just use the GKEntity it is common to Subclass it for a specific type of game entity.

It is important that it is only possible to add a component of a class once. In case you add a second component of the same class it will override the first exsisting component inside of the `GKEntity`

```
let otherComponent = PlayerSpriteComponent()
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.addComponent(otherComponent)
print(player.components.count) //will print 1
print(player.components[0] === otherComponent) // will print true
```

You may ask why. The reason for this is the methods called `component(for: T.Type)` which returns the component of a specific type of the entity.

```
let component = player.component(ofType: PlayerSpriteComponent.self)
```

In addition to the components-methods it has an `update` method which is used to delegate the delta time or current time of the game logic to it's components.

```
var player = Player()
player.addComponent(PlayerSpriteComponent())
player.update(deltaTime: 1.0) // will call the update method of the PlayerSpriteComponent added to
it
```

**GKComponent**

A component represents something of an entity for example the visual component or the logic component.

If an the update method of an entity is called it will delegate this to all of it's components. Overriding this method is used to manipulate an Entity.

```
class PlayerSpriteComponent: GKComponent {
    override func update(deltaTime seconds: TimeInterval) {
        //move the sprite depending on the update time
    }
}
```

In addition to this it is possible to override the method `didAddToEntity` and `willRemoveFromEntity` to inform other components about it's removal or add.

To manipulate a other component inside of a component it is possible to get the GKEntity which the component is added to.

```
override func update(deltaTime seconds: TimeInterval) {
    let controller = self.entity?.component(ofType: PlayerControlComponent.self)
    //call methods on the controller
}
```

While this is possible it is **not** a common pattern since it wires the two components together.

**GKComponentSystem**

While we just talked about using the update delegate mechanism of the `GKEntity` to update the `GKComponents` there is a different way to update `GKComponents` which is called `GKComponentSystem`.

It is used in case it is needed that all components of a specific type need to be updated in one go.

A `GKComponentSystem` is created for a specific type of component.

```
let system = GKComponentSystem(componentClass: PlayerSpriteComponent.self)
```

To add a component you can use the add method:

```
system.addComponent(PlayerSpriteComponent())
```

But a more common way is to pass the created entity with it's components to the `GKComponentSystem` and it will find a matching component inside of the entity.

```
system.addComponent(foundIn: player)
```

To update all components of a specific type call the update:

```
system.update(deltaTime: delta)
```

In case you want to use the `GKComponentSystem` instead of a entity based update mechanism you have to have a `GKComponentSystem` for every component and call the update on all of the systems.

# Chapter 132: Xcode Build & Archive From Command Line

| Option | Description |
|---|---|
| -project | Build the project name.xcodeproj. |
| -scheme | Required if building a workspace. |
| -destination | Use the destination device |
| -configuration | Use the build configuration |
| -sdk | specified SDK |

## Section 132.1: Build & Archive

**Build:**

```
xcodebuild -exportArchive -exportFormat ipa \
    -archivePath "/Users/username/Desktop/MyiOSApp.xcarchive" \
    -exportPath "/Users/username/Desktop/MyiOSApp.ipa" \
    -exportProvisioningProfile "MyCompany Distribution Profile"
```

**Archive:**

```
xcodebuild -project <ProjectName.xcodeproj>
    -scheme <ProjectName>
    -sdk iphonesimulator
    -configuration Debug
    -destination "platform=iOS Simulator,name=<Device>,OS=9.3"
    clean build
```

# Chapter 133: XCTest framework - Unit Testing

## Section 133.1: Adding Test Files to Xcode Project

**When creating the project**

You should check "Include Unit Tests" in the project creation dialog.
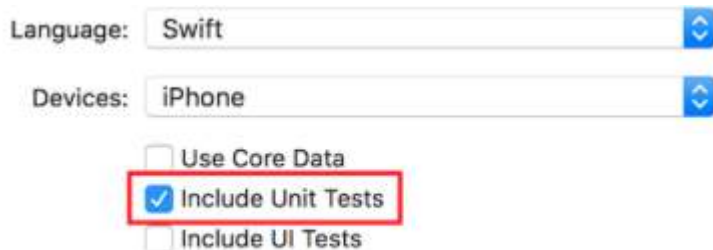


**After creating the project**

If you missed checking that item while creating your project, you could always add test files later. To do so:

1- Go to your project settings in Xcode

2- Go to "Targets"

3- Click "Add Target"

4- Under "Other", select "Cocoa Touch Unit Test Testing Bundle"

At the end, you should have a file named [Your app name]Tests.swift. In Objective-C, you should have two files named [Your app name]Tests.h and [Your app name]Tests.m instead.

[Your app name]Tests.swift or .m file will include by default :

- A XCTest module import
- A [Your app name]Tests class which extends XCTestCase
- setUp, tearDown, testExample, testPerformanceExample methods

**Swift**

```
import XCTest

class MyProjectTests: XCTestCase {

override func setUp() {
    super.setUp()
    // Put setup code here. This method is called before the invocation of each test method in the
class.
}

override func tearDown() {
    // Put teardown code here. This method is called after the invocation of each test method in
the class.
    super.tearDown()
}
```

```swift
func testExample() {
    // This is an example of a functional test case.
    // Use XCTAssert and related functions to verify your tests produce the correct results.

}

func testPerformanceExample() {
    // This is an example of a performance test case.
    self.measure {
        // Put the code you want to measure the time of here.
    }
}

}
```

**Objective-C**

```objc
#import <XCTest/XCTest.h>

@interface MyProjectTests : XCTestCase

@end

@implementation MyProjectTests

- (void)setUp {
    [super setUp];
// Put setup code here. This method is called before the invocation of each test method in the
class.
}

- (void)tearDown {
// Put teardown code here. This method is called after the invocation of each test method in the
class.
    [super tearDown];
}

- (void)testExample {
// This is an example of a functional test case.
// Use XCTAssert and related functions to verify your tests produce the correct results.
}

- (void)testPerformanceExample {
// This is an example of a performance test case.
    [self measureBlock:^{
    // Put the code you want to measure the time of here.
    }];
}

@end
```

# Section 133.2: Adding test methods

According to Apple:

> **Test Methods**
>
> A test method is an instance method of a test class that begins with the prefix test, takes no parameters, and returns void, for example, (void)testColorIsRed(). A test method exercises code in your project and, if that code does not produce the expected result, reports failures using a set of assertion APIs. For example, a function's return value might be compared against an expected value or your test might

> assert that improper use of a method in one of your classes throws an exception.

So we add a test method using "test" as the prefix of the method, like:

**Swift**
```swift
func testSomething() {

}
```
**Objective-C**
```objc
- (void)testSomething {

}
```

To actually test the results, we use `XCTAssert()` method, which takes a boolean expression, and if true, marks the test as succeeded, else it will mark it as failed.

Let's say we have a method in View Controller class called `sum()` which calculates sum of two numbers. To test it, we use this method:

**Swift**
```swift
func testSum(){
    let result = viewController.sum(4, and: 5)
    XCTAssertEqual(result, 9)
}
```
**Objective-C**
```objc
- (void)testSum {
    int result = [viewController sum:4 and:5];
    XCTAssertEqual(result, 9);
}
```

> **Note**
>
> By default, you can't access label, text box or other UI items of the View Controller class from test class if they are first made in Storyboard file. This is because they are initialized in `loadView()` method of the View Controller class, and this will not be called when testing. The best way to call `loadView()` and all other required methods is accessing the `view` property of our `viewController` property. You should add this line before testing UI elements:

```swift
XCTAssertNotNil(viewController.view)
```

# Section 133.3: Writing a test class

```swift
import XCTest
@testable import PersonApp

class PersonTests: XCTestCase {
    func test_completeName() {
        let person = Person(firstName: "Josh", lastName: "Brown")
        XCTAssertEqual(person.completeName(), "Josh Brown")
    }
}
```

Now let's discuss what's going on here. The `import` XCTest line will allow us to extend XCTestCase and use

XCTAssertEqual (among other assertions). Extending XCTestCase and prefixing our test name with test will ensure that Xcode automatically runs this test when running the tests in the project (**?U** or **Product** > **Test**). The @testable import PersonApp line will import our PersonApp target so we can test and use classes from it, such as the Person in our example above. And finally, our XCTAssertEqual will ensure that person.completeName() is equal to the string "Josh Brown".

# Section 133.4: Adding Storyboard and View Controller as instances to test file

To get started with unit testing, which will be done in the tests file and will be testing the View Controller and Storyboard, we should introduce these two files to the test file.

**Defining the View Controller**
**Swift**

```
var viewController : ViewController!
```

**Introducing the Storyboard and initializing the View Controller**

Add this code to the setUp() method:

**Swift**

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
viewController = storyboard.instantiateInitialViewController() as! ViewController
```

**Objective-C**

```
UIStoryboard *storyboard = [UIStoryboard storyboardWithName:"Main" bundle:nil];
viewController = (ViewController *) [storyboard instantiateInitialViewController];
```

This way, you could write test methods, and they will know where to check for errors. In this case, there are View Controller and the Storyboard.

# Section 133.5: Import a module that it can be tested

Classes, structs, enums and all their methods are internal by default. This means they can be only accessed from the same module. The test cases are in a different target an this means they are in a different module. To be able to access the method you want to test, you need to import the module to be tested using the @testable keyword.

Let's say we have a main module called ToDo and we want to write tests for it. We would import that module like this:

```
@testable import ToDo
```

All test methods in the file with this import statement can now access all internal classes, structs, enums and all their internal methods of the ToDo module.

You should never add the files with the elements you want to test to the test target because that can lead to hard to debug errors.

# Section 133.6: Trigger view loading and appearance

**View loading**

In a test for a view controller you want sometimes to trigger the execution of loadView() or viewDidLoad(). This can be done by accessing the view. Let's say you have view controller instance in your test called sut (system under

test), then the code would look like this:

```
XCTAssertNotNil(sut.view)
```

**View appearance**

You can also trigger the methods `viewWillAppear(_:)` and `viewDidAppear(_:)` by adding the following code:

```
sut.beginAppearanceTransition(true, animated: true)
sut.endAppearanceTransition()
```

# Section 133.7: Start Testing

**Testing a specific method**

To test a specific method, click the square next to the method definition.

**Testing all methods**

To test all methods, click the square next to the class definition.

**See the testing result**

If there is a green check next to the definition, the test has succeeded.



If there is a red cross next to the definition, the test has failed.



**Running all tests**

```
Product -> Test OR Cmd + U
```

It will run all the tests from all the test targets!

# Chapter 134: AVPlayer and AVPlayerViewController

## Section 134.1: Playing Media using AVPlayer and AVPlayerLayer

**Objective C**

```objc
NSURL *url = [NSURL URLWithString:@"YOUR URL"];
AVPlayer *player = [AVPlayer playerWithURL:videoURL];
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
playerLayer.frame = self.view.bounds;
[self.view.layer addSublayer:playerLayer];
[player play];
```

**Swift**

```swift
let url = NSURL(string: "YOUR URL")
let player = AVPlayer(URL: videoURL!)
let playerLayer = AVPlayerLayer(player: player)
playerLayer.frame = self.view.bounds
self.view.layer.addSublayer(playerLayer)
player.play()
```

## Section 134.2: Playing Media Using AVPlayerViewController

**Objective-C**

```objc
NSURL *url = [[NSURL alloc] initWithString:@"YOUR URL"]; // url can be remote or local

AVPlayer *player = [AVPlayer playerWithURL:url];
// create a player view controller

AVPlayerViewController *controller = [[AVPlayerViewController alloc] init];
[self presentViewController:controller animated:YES completion:nil];
controller.player = player;
[player play];
```

**Swift**

```swift
let player = AVPlayer(URL: url) // url can be remote or local

let playerViewController = AVPlayerViewController()
// creating a player view controller
playerViewController.player = player
self.presentViewController(playerViewController, animated: true) {

    playerViewController.player!.play()
}
```

## Section 134.3: AVPlayer Example

AVPlayer *avPlayer = [AVPlayer playerWithURL:[NSURL URLWithString:@"YOUR URL"]];

```objc
    AVPlayerViewController *avPlayerCtrl = [[AVPlayerViewController alloc] init];
    avPlayerCtrl.view.frame = self.view.frame;
    avPlayerCtrl.player = avPlayer;
    avPlayerCtrl.delegate = self;
    [avPlayer play];
    [self presentViewController:avPlayerCtrl animated:YES completion:nil
```

# Chapter 135: Deep Linking in iOS

## Section 135.1: Adding a URL scheme to your own app

Let's say you're working on an app called `MyTasks`, and you want to allow inbound URLs to create a new task with a title and a body. The URL you're designing might look something like this:

```
mytasks://create?title=hello&body=world
```

(Of course, the `text` and body parameters are used to populate our task that we're creating!)

**Here are the Big Steps to adding this URL scheme to your project:**

1. Register a URL scheme in your app's `Info.plist` file, so the system knows when to route a URL to your app.
2. Add a function to your `UIApplicationDelegate` that accepts and handles incoming URLs.
3. Perform whatever task needs to occur when that URL is opened.

**Step One: Register a URL scheme in Info.plist:**

First, we need to add a "URL Types" entry to our Info.plist file. Click the (+) button here:
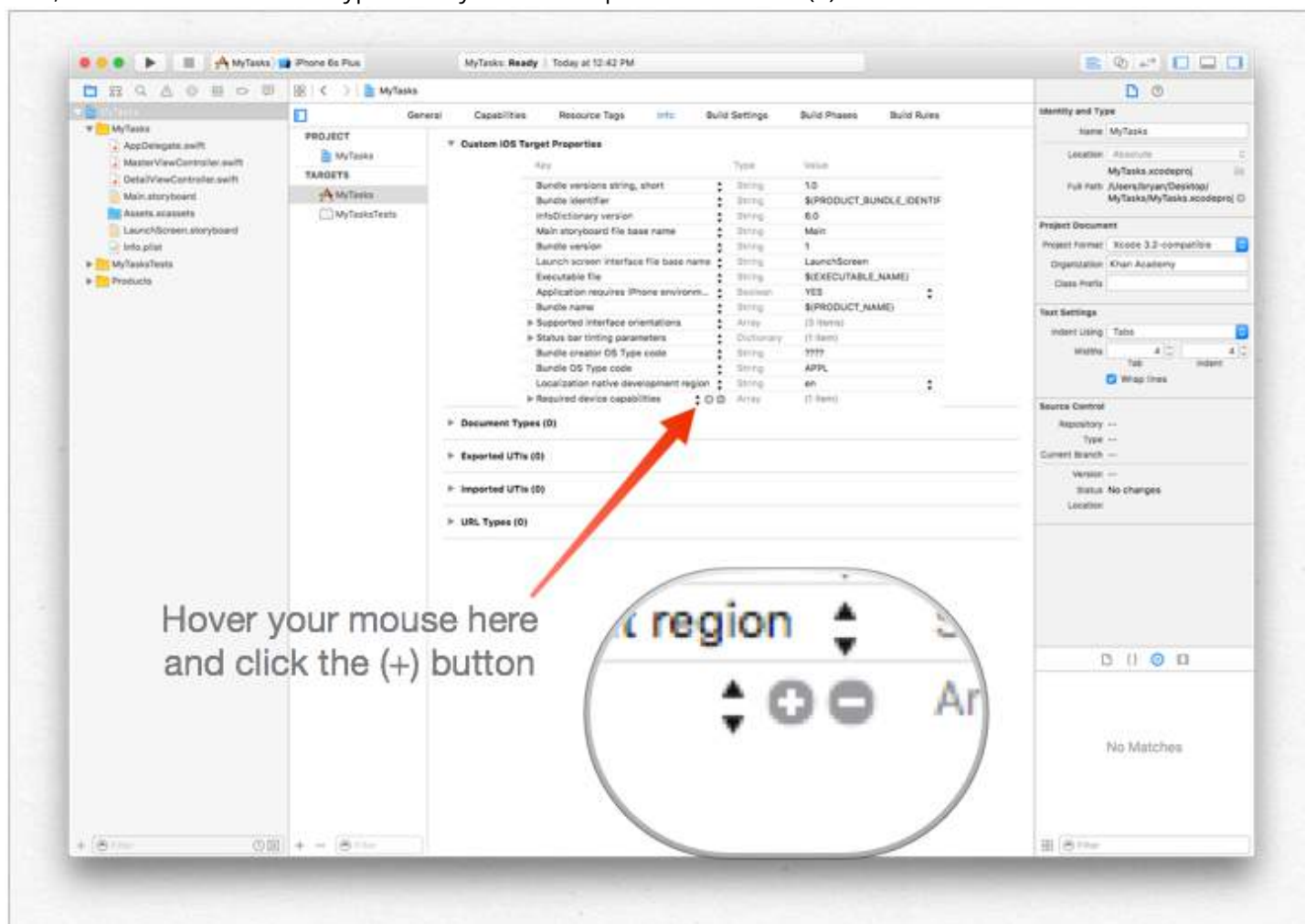


...then enter a unique identifier for your app, as well as the URL scheme you want to use. Be specific! You don't want the URL scheme to conflict with another app's implementation. Better to be too-long here than too-short!

| | | | |
|---|---|---|---|
| ▼ URL types | ↕ | Array | (5 items) |
| ▼ Item 0 | | Dictionary | (2 items) |
| URL identifier | ↕ | String | com.mycompany |
| ▼ URL Schemes | ↕ | Array | (1 item) |
| Item 0 | ⊕ ⊖ | String | mytasks |
| ▼ Item 1 | | Dictionary | (1 item) |

**Step Two: Handle the URL in the UIApplicationDelegate**

We need to implement `application:openURL:options:` on our `UIApplicationDelegate`. We'll inspect the incoming URL and see if there's an action we can take!

One implementation would be this:

```swift
func application(app: UIApplication, openURL url: NSURL, options: [String : AnyObject]) -> Bool {
    if url.scheme == "mytasks" && url.host == "create" {
        let title = // get the title out of the URL's query using a method of your choice
        let body = // get the title out of the URL's query using a method of your choice
        self.rootViewController.createTaskWithTitle(title, body: body)
        return true
    }

    return false
}
```

**Step Three: Perform a task depending on the URL.**

When a user opens your app via a URL, they probably expected *something* to happen. Maybe that's navigating to a piece of content, maybe that's creating a new item - in this example, we're going to create a new task in the app!

In the above code, we can see a call to `self.rootViewController.createTaskWithTitle(:body:)` - so assuming that your `AppDelegate` has a pointer to its root view controller which implements the function properly, you're all set!

# Section 135.2: Opening an app based on its URL scheme

To open an app with defined URL scheme `todolist://`:

Objective-C

```objc
NSURL *myURL = [NSURL URLWithString:@"todolist://there/is/something/to/do"];
[[UIApplication sharedApplication] openURL:myURL];
```

Swift

```swift
let stringURL = "todolist://there/is/something/to/do"
if let url = NSURL(string: stringURL) {
    UIApplication.shared().openURL(url)
}
```

HTML

```html
<a href="todolist://there/is/something/to/do">New SMS Message</a>
```

> **Note:** It's useful to check if link can be opened to otherwise display an appropriate message to the user. This can be done using `canOpenURL:` method.

# Section 135.3: Setting up deeplink for your app

Setting up deep-linking for your app is easy.You just need a small url using which you want to open your app.

Follow the steps to set up deep-linking for your app.

1. Lets create a project and name it DeepLinkPOC.

2. Now select your project target.

3. After selecting target,select the 'info' tab.

4. Scroll down to the bottom until you see an option of **URL Types**

5. Click '+' option.

6. You will see **URL schemes** add a string using which you want to open your app.Lets add "**DeepLinking**" in URL schemes.

So, to open your app you can launch it by typing **"DeepLinking://"** into your safari. Your deep-linking string has following format.

```
[scheme]://[host]/[path]  --> DeepLinking://path/Page1
```

where, Scheme : "DeepLinking" Host : "path" path : "Page1"

**Note** : Even if don't add host and path it will launch the app,so no worries.But you can add host and path to additionally redirect to particular page after application launch.

7. Now add following method to your appdelegate.

Swift:

```
func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
annotation: AnyObject) -> Bool
```

Objective-c:

```
-(BOOL)application:(UIApplication *)application
        openURL:(NSURL *)url
        sourceApplication:(NSString *)sourceApplication
        annotation:(id)annotation
```

The above method is called whenever your app is launched using a deep-linking string you set for your app.

8. Now is the time to install your app but wait before you directly jump to run button.Lets do a small change in scheme's app-launch method.

- Select and edit your scheme as



- change its launch type and close

---
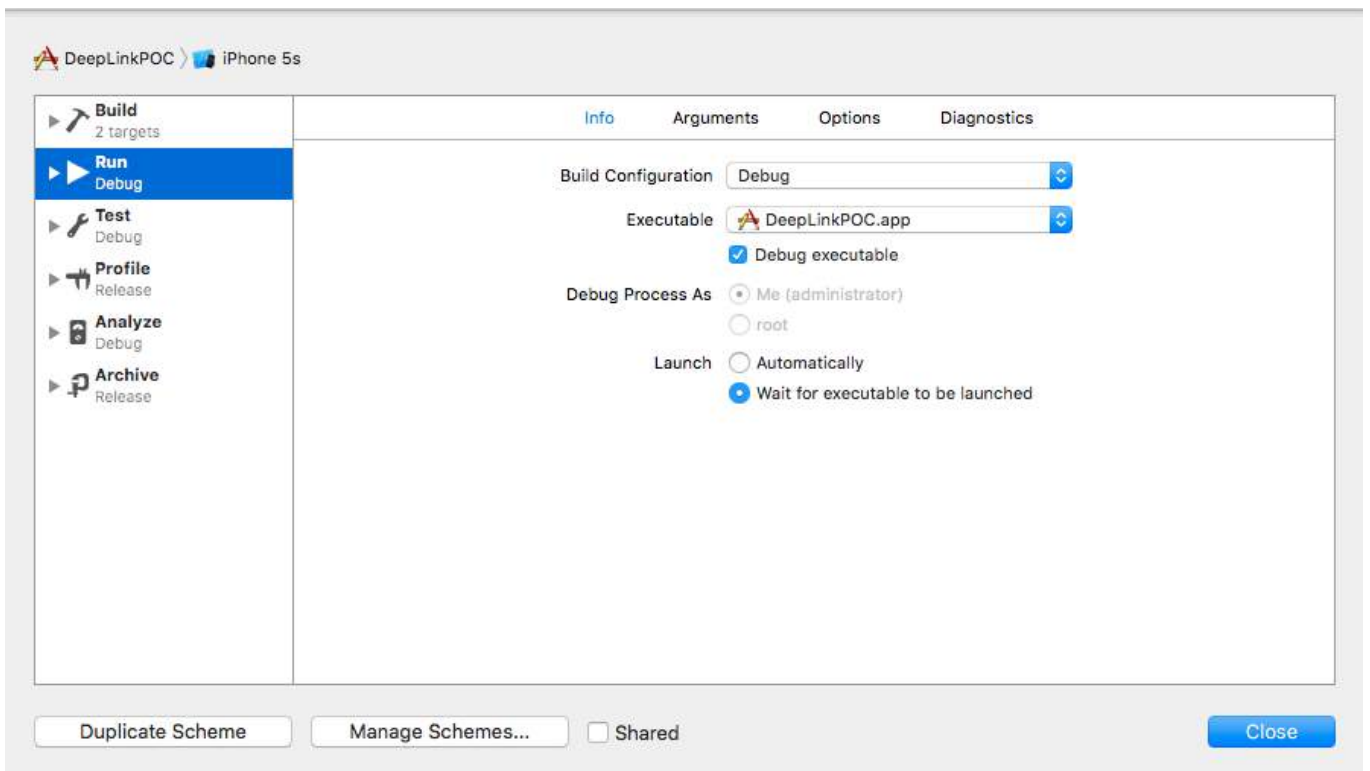
9. Now click the Run button (if you want you can add breakpoint to your didFinishLaunchingWithOptions and openURL methods to observe values)

10. You'll see a message "Waiting for DeepLinkPOC(or your app name) to launch".

11. Open safari and type in "**DeepLinking://**" into the search bar this will show prompt "open this page in DeepLinkPOC" click open to launch your app.

Hope you got to know how to set up deep-linking for your app :)

# Chapter 136: Core Graphics

## Section 136.1: Creating a Core Graphics Context

> **Core Graphics context**
>
> A Core Graphics context is a canvas which we can draw in it and set some properties like the line thickness.

**Making a context**

To make a context, we use the `UIGraphicsBeginImageContextWithOptions()` C function. Then, when we are done with drawing, we just call `UIGraphicsEndImageContext()` to end the context:

**Swift**

```swift
let size = CGSize(width: 256, height: 256)

UIGraphicsBeginImageContextWithOptions(size, false, 0)

let context = UIGraphicsGetCurrentContext()

// drawing code here

UIGraphicsEndImageContext()
```

**Objective-C**

```objc
CGSize size = [CGSize width:256 height:256];

UIGraphicsBeginImageContextWithOptions(size, NO, 0);

CGContext *context = UIGraphicsGetCurrentContext();

// drawing code here

UIGraphicsEndImageContext();
```

In the code above, we passed 3 parameters to the `UIGraphicsBeginImageContextWithOptions()` function:

1. A `CGSize` object which stores the whole size of the context (the canvas)

2. A boolean value which if it is true, the context will be opaque

3. An integer value which sets the scale (1 for non-retina, 2 for retina and 3 for retina HD screens). If set to 0, the system automatically handles the scale based on the target device.

## Section 136.2: Presenting the Drawn Canvas to User

**Swift**

```swift
let image = UIGraphicsGetImageFromCurrentImageContext()
imageView.image = image //assuming imageView is a valid UIImageView object
```

**Objective-C**

```objc
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
imageView.image = image; //assuming imageView is a valid UIImageView object
```

# Chapter 137: Segues

## Section 137.1: Using Segues to navigate backwards in the navigation stack

**Unwind Segues**

> Unwind Segues give you a way to "unwind" the navigation stack and specify a destination to go back to. The signature of this function is key to Interface Builder recognizing it. **It must have a return value of IBAction and take one parameter of UIStoryboardSegue**. The name of the function does not matter. In fact, the function does not even have to do anything. It's just there as a marker of which UIViewController is the destination of the Unwind Segue. [source][1]

Required signature of an unwind segue

Objective C:

```
-(IBAction)prepareForUnwind:(UIStoryboardSegue *)segue {
}
```

Swift:

```
@IBAction func prepareForUnwind(segue: UIStoryboardSegue) {
}
```

## Section 137.2: An Overview

From the Apple documentation:

> A UIStoryboardSegue object is responsible for **performing the visual transition between two view controllers**. In addition, segue objects are used to prepare for the transition from one view controller to another. **Segue objects contain information about the view controllers involved in a transition**. When a segue is triggered, but before the visual transition occurs, the storyboard runtime calls the current view controller's prepareForSegue:sender: method so that it can pass any needed data to the view controller that is about to be displayed.

**Attributes**

Swift

```
sourceViewController: UIViewController {get}
destinationViewController: UIViewController {get}
identifier: String? {get}
```

References:

- UIViewController Class Reference
- UIStoryBoardSegue Class Reference

# Section 137.3: Preparing your view controller before a triggering a Segue

***PrepareForSegue***:

```
func prepareForSegue(_ segue:UIStoryboardSegue, sender sender:AnyObject?)
```

> Notifies the view controller that a segue is about to be performed

**Parameters**

*segue*: The segue object.

*sender*: The object that initialized the segue.

**Example in Swift**

Perform a task if the identifier of the segue is "SomeSpecificIdentifier"

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "SomeSpecificIdentifier" {
        //- Do specific task
    }
}
```

# Section 137.4: Deciding if an invoked Segue should be performed

***ShouldPerformSegueWithIdentifier***:

```
func shouldPerformSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?) -> Bool
```

> Determines whether the segue with the specified identifier should be performed.

**Parameters**

*Identifier*: String that identifies the triggered segue

*Sender*: The object that initialized the segue.

**Example in Swift**

Only perform segue if the identifier is "SomeSpecificIdentifier"

```
override func shouldPerformSegueWithIdentifier(identifier:String, sender:AnyObject?) -> Bool {
    if identifier == "SomeSpecificIdentifier" {
        return true
    }
    return false
}
```

# Section 137.5: Trigger Segue Programmatically

**PerformSegueWithIdentifier:**

```
func performSegueWithIdentifier(_ identifier:String, sender sender:AnyObject?)
```

> Initiates the segue with the specified identifier from the current view controller's storyboard file

**Parameters**

*Identifier*: String that identifies the triggered segue

*Sender*: The object that will initiate the segue.

**Example in Swift**

Performing a segue with identifier "SomeSpecificIdentifier" from a table view row selection:

```
func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath) {
    performSegueWithIdentifier("SomeSpecificIdentifier", sender: indexPath.item)
}
```

# Chapter 138: EventKit

## Section 138.1: Accessing different types of calendars

**Accessing the array of calendars**

To access the array of `EKCalendars`, we use the `calendarsForEntityType` method:

**Swift**

```swift
let calendarsArray = eventStore.calendarsForEntityType(EKEntityType.Event) as! [EKCalendar]
```

**Iterating through calendars**

Just use a simple `for` loop:

**Swift**

```swift
for calendar in calendarsArray{
    //...
}
```

**Accessing the calendar title and color**

**Swift**

```swift
let calendarColor = UIColor(CGColor: calendar.CGColor)
let calendarTitle = calendar.title
```

**Objective-C**

```objc
UIColor *calendarColor = [UIColor initWithCGColor: calendar.CGColor];
NSString *calendarTitle = calendar.title;
```

## Section 138.2: Requesting Permission

Your app can't access your reminders and your calendar without permission. Instead, it must show an alert to user, requesting him/her to grant access to events for the app.

To get started, import the `EventKit` framework:

**Swift**

```swift
import EventKit
```

**Objective-C**

```objc
#import <EventKit/EventKit.h>
```

**Making an `EKEventStore`**

Then, we make an `EKEventStore` object. This is the object from which we can access calendar and reminders data:

**Swift**

```swift
let eventStore = EKEventStore()
```

**Objective-C**

```objc
EKEventStore *eventStore = [[EKEventStore alloc] init];
```

> **Note**
>
> Making an `EKEventStore` object every time we need to access calendar is not efficient. Try to make it once and use it everywhere in your code.

**Checking Availability**

Availability has three different status: Authorized, Denied and Not Determined. Not Determined means the app needs to grant access.

To check availability, we use `authorizationStatusForEntityType()` method of the `EKEventStore` object:

**Swift**

```swift
switch EKEventStore.authorizationStatusForEntityType(EKEntityTypeEvent){
    case .Authorized: //...
    case .Denied: //...
    case .NotDetermined: //...
    default: break
}
```

**Objective-C**

```objc
switch ([EKEventStore authorizationStatusForEntityType:EKEntityTypeEvent]){
    case EKAuthorizationStatus.Authorized:
        //...
        break;
    case EKAuthorizationStatus.Denied:
        //...
        break;
    case EKAuthorizationStatus.NotDetermined:
        //...
        break;
    default:
        break;
}
```

**Requesting Permission**

Put the following code in `NotDetermined` case:

**Swift**

```swift
eventStore.requestAccessToEntityType(EKEntityTypeEvent, completion: { [weak self]
(userGrantedAccess, _) -> Void in
    if userGrantedAccess{
        //access calendar
    }
}
```

# Section 138.3: Adding an event

**Creating the event object**
**Swift**

```swift
var event = EKEvent(eventStore: eventStore)
```

**Objective-C**

```objc
EKEvent *event = [EKEvent initWithEventStore:eventStore];
```

**Setting related calendar, title and dates**
**Swift**

```swift
event.calendar = calendar
event.title = "Event Title"
event.startDate = startDate //assuming startDate is a valid NSDate object
event.endDate = endDate //assuming endDate is a valid NSDate object
```

**Adding event to calendar**

**Swift**

```swift
try {
    do eventStore.saveEvent(event, span: EKSpan.ThisEvent)
} catch let error as NSError {
    //error
}
```

**Objective-C**

```objc
NSError *error;
BOOL *result = [eventStore saveEvent:event span:EKSpanThisEvent error:&error];
if (result == NO){
    //error
}
```

# Chapter 139: SiriKit

## Section 139.1: Adding Siri Extension to App

To integrate Siri capabilities in your app, you should add an extensions as you would do while creating an iOS 10 Widget (old Today View Extension) or a custom keyboard.

**Adding capability**

1- In the project settings, select your iOS app target and go to Capabilities tab

2- Enable the Siri capability

**Adding the extension**

1- Go to File -> New -> Target...

2- Select iOS -> Application Extension from the left pane

3- Double-click Intents Extension from right

> **According to Apple:**
>
> Intents Extension template builds an Intents extension that allows your app to handle intents issued by system services like Siri and Maps.
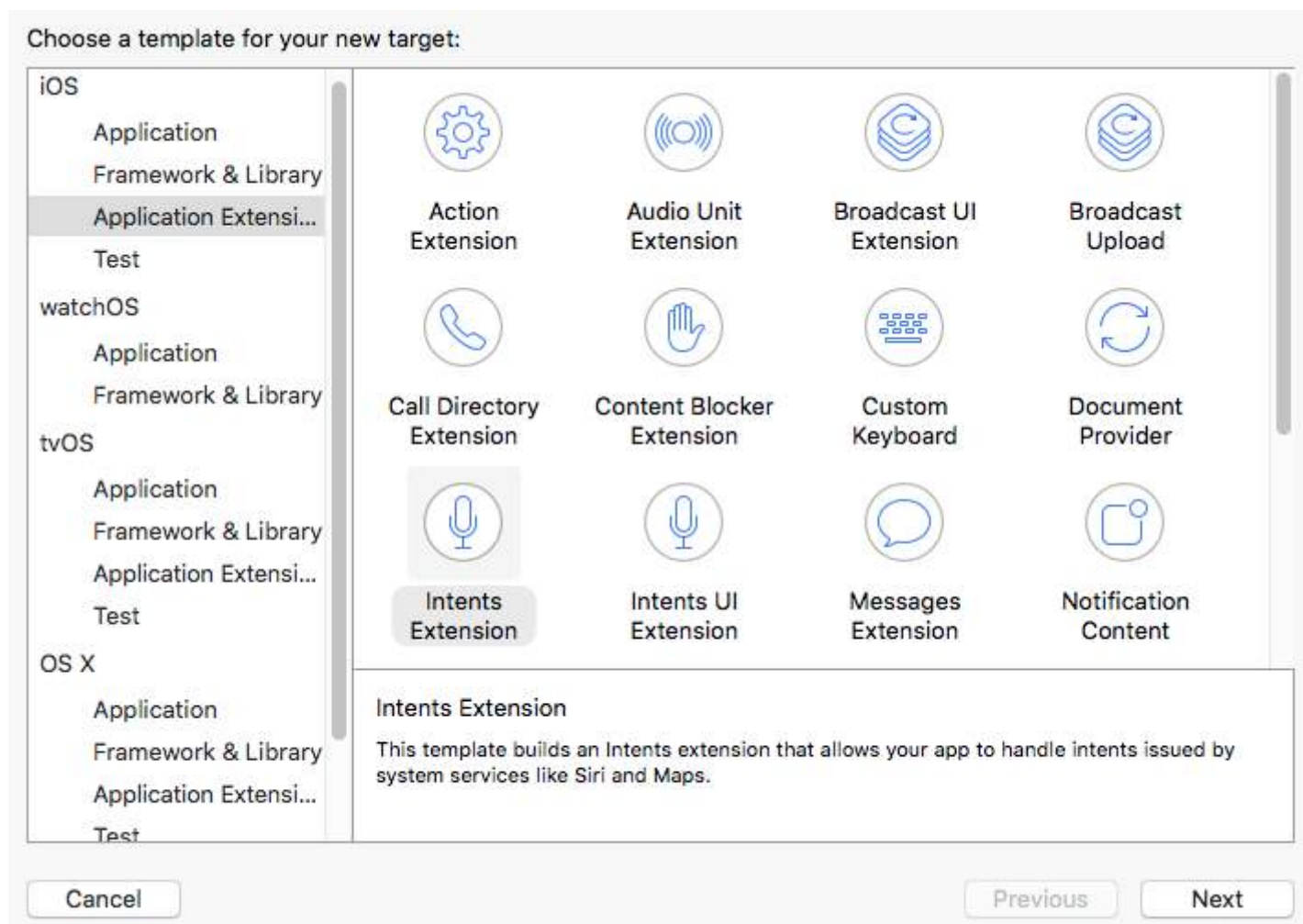
4- Choose a name, and be sure to check "Include UI Extension"



By doing this steps, two new targets (Intents Extension and UI Extension) are created, and by default they contain Workout Intent code. For different types of Siri requests, see Remarks.

> **Note**
>
> Anytime you want to debug your extension, just select the Intent scheme from the available schemes.

> **Note**
>
> You can't test SiriKit apps in the Simulator. Instead, you need a real device.

# Chapter 140: Contacts Framework

## Section 140.1: Adding a Contact

**Swift**

```swift
import Contacts

// Creating a mutable object to add to the contact
let contact = CNMutableContact()

contact.imageData = NSData() // The profile picture as a NSData object

contact.givenName = "John"
contact.familyName = "Appleseed"

let homeEmail = CNLabeledValue(label:CNLabelHome, value:"john@example.com")
let workEmail = CNLabeledValue(label:CNLabelWork, value:"j.appleseed@icloud.com")
contact.emailAddresses = [homeEmail, workEmail]

contact.phoneNumbers = [CNLabeledValue(
    label:CNLabelPhoneNumberiPhone,
    value:CNPhoneNumber(stringValue:"(408) 555-0126"))]

let homeAddress = CNMutablePostalAddress()
homeAddress.street = "1 Infinite Loop"
homeAddress.city = "Cupertino"
homeAddress.state = "CA"
homeAddress.postalCode = "95014"
contact.postalAddresses = [CNLabeledValue(label:CNLabelHome, value:homeAddress)]

let birthday = NSDateComponents()
birthday.day = 1
birthday.month = 4
birthday.year = 1988  // You can omit the year value for a yearless birthday
contact.birthday = birthday

// Saving the newly created contact
let store = CNContactStore()
let saveRequest = CNSaveRequest()
saveRequest.addContact(contact, toContainerWithIdentifier:nil)
try! store.executeSaveRequest(saveRequest)
```

## Section 140.2: Authorizing Contact Access

**Importing the framework**
**Swift**

```swift
import Contacts
```

**Objective-C**

```objc
#import <Contacts/Contacts.h>
```

**Checking accessibility**
**Swift**

```swift
switch CNContactStore.authorizationStatusForEntityType(CNEntityType.Contacts){
case .Authorized: //access contacts
case .Denied, .NotDetermined: //request permission
default: break
}
```

**Objective-C**

```objectivec
switch ([CNContactStore authorizationStatusForEntityType:CNEntityType.Contacts]){
case CNAuthorizationStatus.Authorized:
    //access contacts
    break;
case CNAuthorizationStatus.Denied:
    //request permission
    break;
case CNAuthorizationStatus.NotDetermined:
    //request permission
    break;
}
```

**Requesting Permission**

**Swift**

```swift
var contactStore = CKContactStore()
contactStore.requestAccessForEntityType(CKEntityType.Contacts, completionHandler: { (ok, _) -> Void
in
    if access{
        //access contacts
    }
}
```

# Section 140.3: Accessing Contacts

**Applying a filter**

To access contacts, we should apply a filter of type `NSPredicate` to our contactStore variable which we defined it in Authorizing Contact Access example. For example, here we want to sort out contacts with name matching with our own:

**Swift**

```swift
let predicate = CNContact.predicateForContactsMatchingName("Some Name")
```

**Objective-C**

```objectivec
NSPredicate *predicate = [CNContact predicateForContactsMatchingName:@"Some Name"];
```

**Specifying keys to fetch**

Here, we want to fetch the contact's first name, last name and profile image:

**Swift**

```swift
let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactImageDataKey]
```

**Fetching contacts**

**Swift**

```swift
do {
    let contacts = try contactStore.unifiedContactsMatchingPredicate(predicate, keysToFetch: keys)
} catch let error as NSError {
    //...
}
```

**Accessing contact details**

**Swift**

```swift
print(contacts[0].givenName)
print(contacts[1].familyName)
let image = contacts[2].imageData
```

# Chapter 141: iOS 10 Speech Recognition API

## Section 141.1: Speech to text: Recognize speech from a bundle contained audio recording

```swift
//import Speech
//import AVFoundation

// create a text field to show speech output
@IBOutlet weak var transcriptionTextField: UITextView!
// we need this audio player to play audio
var audioPlayer: AVAudioPlayer!

override func viewDidLoad()
{
    super.viewDidLoad()
}

// this function is required to stop audio on audio completion otherwise it will play same audio
again and again
func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
{
    player.stop()
}

// this function is required to get a speech recognizer and after that make and request to speech
recognizer
func requestSpeechAuth()
{
    SFSpeechRecognizer.requestAuthorization { authStatus in
        if authStatus == SFSpeechRecognizerAuthorizationStatus.authorized {
            if let path = Bundle.main.url(forResource: "mpthreetest", withExtension: "m4a") {
                do {
                    let sound = try AVAudioPlayer(contentsOf: path)
                    self.audioPlayer = sound
                    self.audioPlayer.delegate = self
                    sound.play()
                }   catch {
                    print("error")
                }

                let recognizer = SFSpeechRecognizer()
                let request = SFSpeechURLRecognitionRequest(url:path)
                recognizer?.recognitionTask(with: request) { (result, error) in
                    if let error = error {
                    print("there is a error\(error)")
                    } else {
// here you are printing out the audio output basically showing it on uitext field
                        self.transcriptionTextField.text =
result?.bestTranscription.formattedString
                    }
                }
            }
        }
    }
}

// here you are calling requestSpeechAuth function on UIButton press
@IBAction func playButtonPress(_ sender: AnyObject)
{
```

```
    requestSpeechAuth()
}
```

# Chapter 142: StoreKit

## Section 142.1: Get localized product information from the App Store

Get localized product information from a set of product identifier strings using SKProductsRequest:

```
import StoreKit

let productIdentifierSet = Set(["yellowSubmarine", "pennyLane"])
let productsRequest = SKProductsRequest(productIdentifiers: productIdentifierSet)
```

In order to process the products from the productsRequest, we need to assign a delegate to the request that handles the response. The delegate needs to conform to the SKProductsRequestDelegate protocol, which means that it must inherit from NSObject (i.e. any Foundation object) and implement the productsRequest method:

```
class PaymentManager: NSObject, SKProductsRequestDelegate {

    var products: [SKProduct] = []

    func productsRequest(request: SKProductsRequest,
                         didReceiveResponse response: SKProductsResponse) {

        products = response.products

    }

}
```

To initiate the productsRequest we assign PaymentManager as the products-request's delegate, and call the start() method on the request:

```
let paymentManager = PaymentManager()
productsRequest.delegate = paymentManager
productsRequest.start()
```

If the requests succeeds the products will be in paymentManager.products.

# Chapter 143: Code signing

## Section 143.1: Provisioning Profiles

In order to build an IPA file in XCode, you require to sign your application with a certificate and provisioning profile. These can be created at https://developer.apple.com/account/ios/profile/create

**Provisioning Profile Types**

Provisioning Profiles are split into two types, Development, and Distribution:

**Development**

- iOS App Development / tvOS App Development - used in development in order to install your app onto a test device.

**Distribution**

- App Store / tvOS App Store - used to sign your application for app store upload.
- In House - Used for Enterprise distribution of your app, to devices within your business.
- Ad Hoc / tvOS Ad Hoc - Used to distribute your app to a limited number of specific devices (e.g. you will need to know the UDIDs of the devices you want to install your app to).

# Chapter 144: Create .ipa File to upload on appstore with Applicationloader

## Section 144.1: create .ipa file to upload app to appstore with Application Loader

If you want to upload .ipa file to itunesconnect **without integrating developer account in Xcode** and you want to use **application loader**. then you can **generate .ipa with iTunes** .

**Step 1:** Select device inplace of simulator.
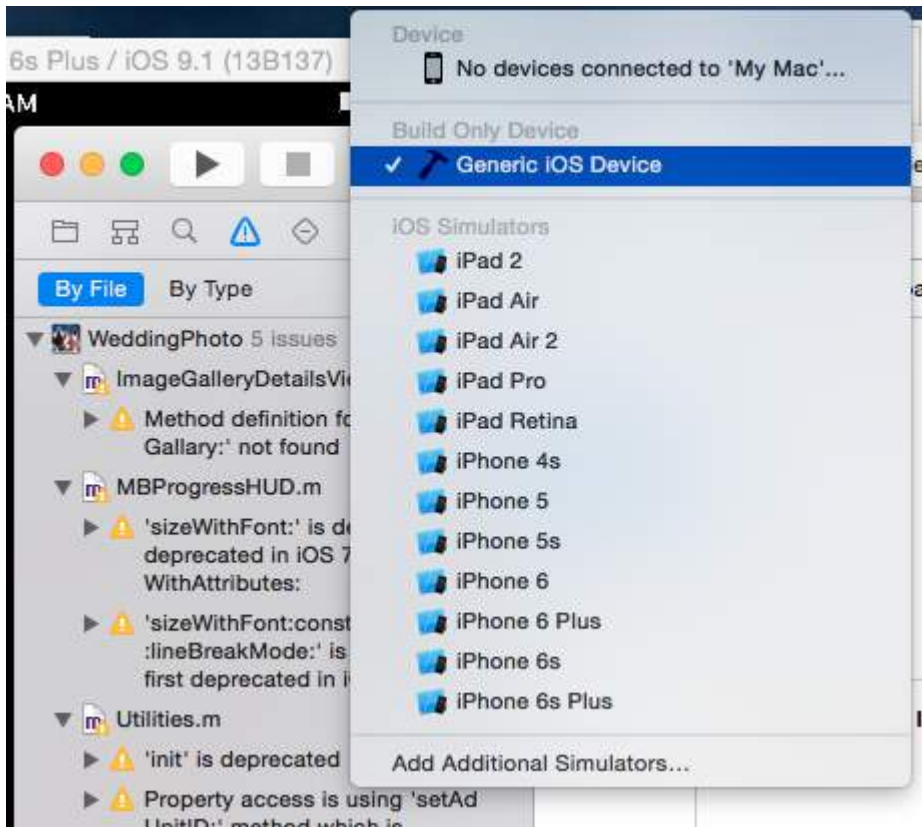


**Step 2:** Go to Product -> select Archive
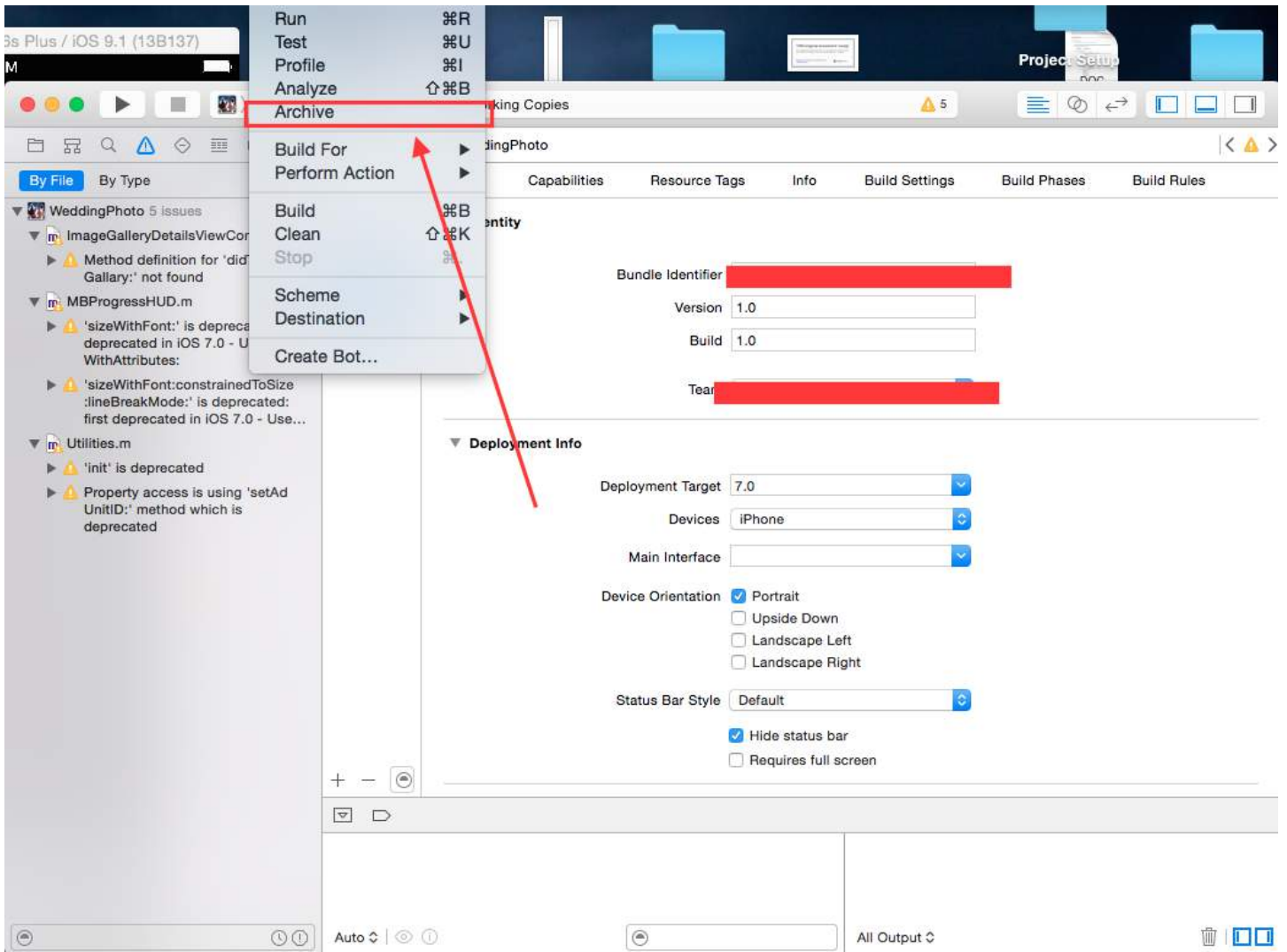
**Step 3:** After complited process right click to your Archive -> and select show in Finder
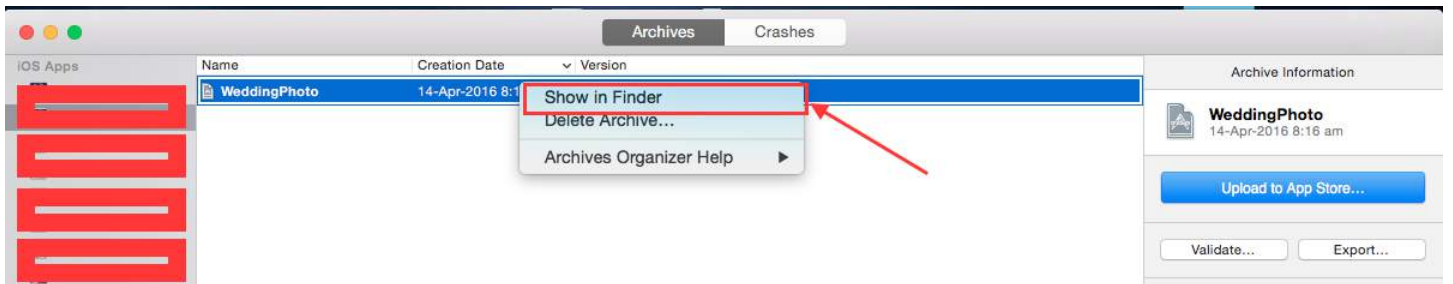


**Step 4:** when you click on show in finder you will redirect to Archive folder, looks like this



**Step 5:** Right click on .xarchive file -> select Show in finder option.
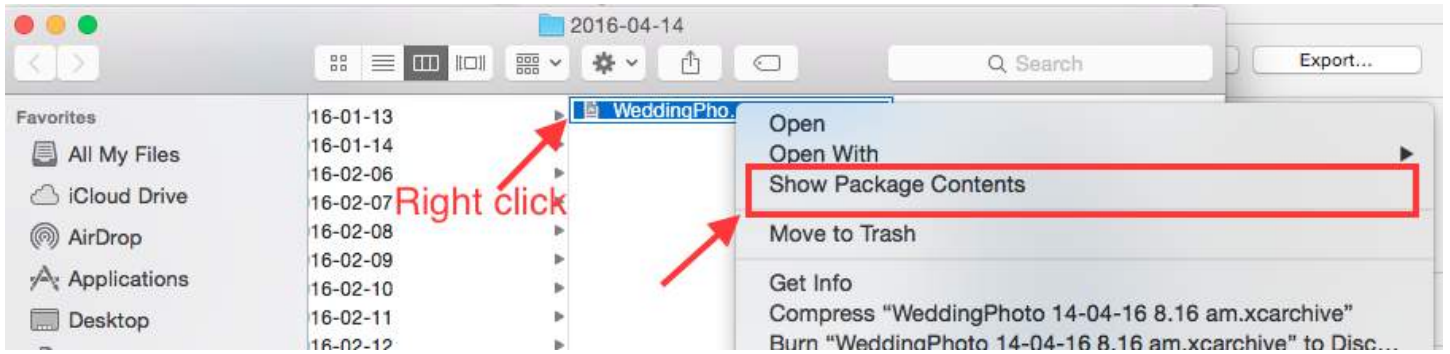
**Step 6:** Go to Product Folder -> Application Folder -> You will find yourprojectname.app
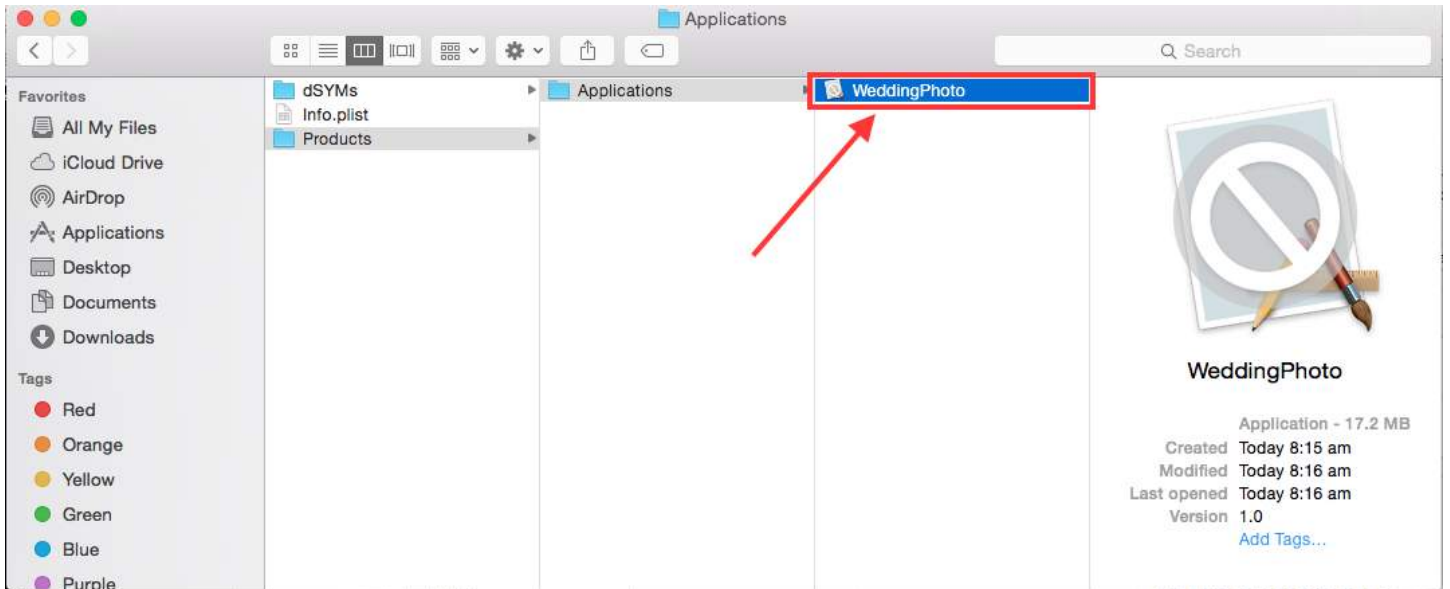


**Step 7:** Now to convert .app to .ipa just drag and drop into itunes . check below image ,
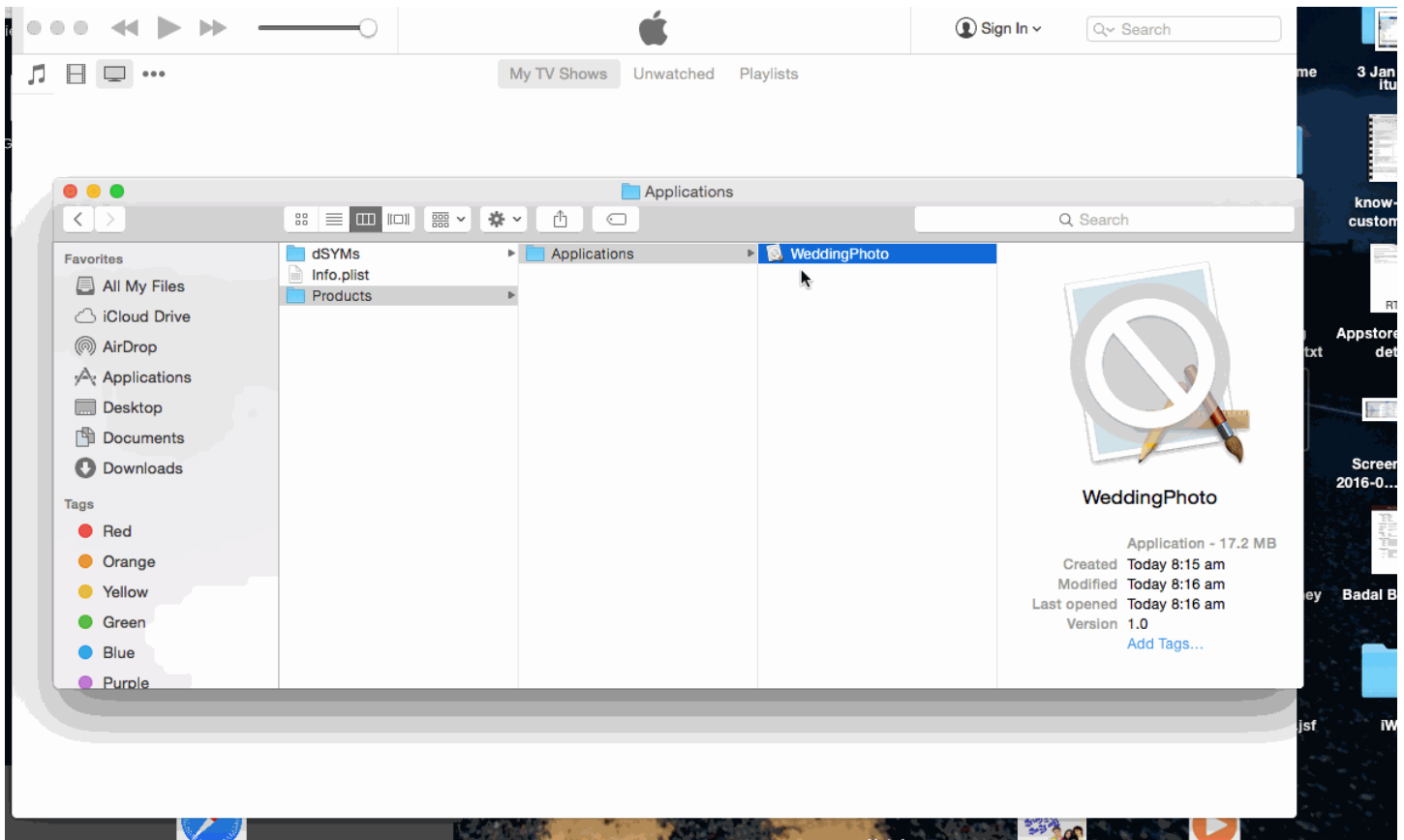
**Step 8:** Now put this .ipa file in safe place and use when upload with application loader .

**Note:** if you want to know how to upload app with application loader then check this ,

[Upload app with application Loader](#)

**EDIT:**

**WARNING:** Don't make .ipa with changing extension from .aap to .zip and .zip to .ipa.

I have seen in many answer that , they have suggest compress .app file and then change the extension from .zip to .ipa . It is not working now . By this method you will get Error like ,

> IPA is invalid, it does not include a payload directory.

# Chapter 145: Size Classes and Adaptivity

## Section 145.1: Size Classes and Adaptivity through Storyboard

We can add adaptivity to any subclass of `UIView` which we add on view controller in nib file.
Lets take an example of adding adaptivity using size classes to a view.

1. Add a view on view controller as:
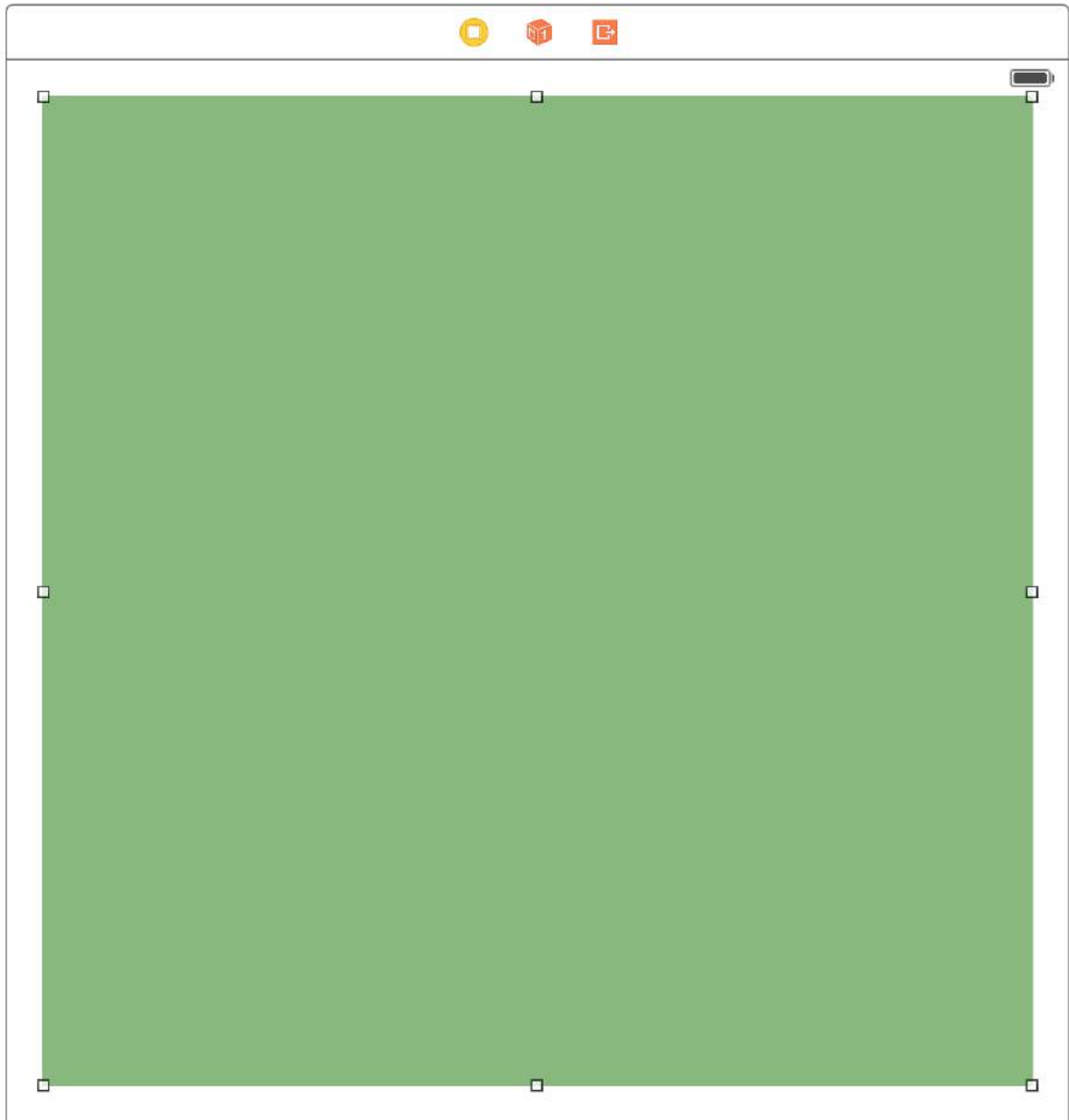


2. Now we need to pin this view to it's superview for fixing it's size and position using **constraints** as:
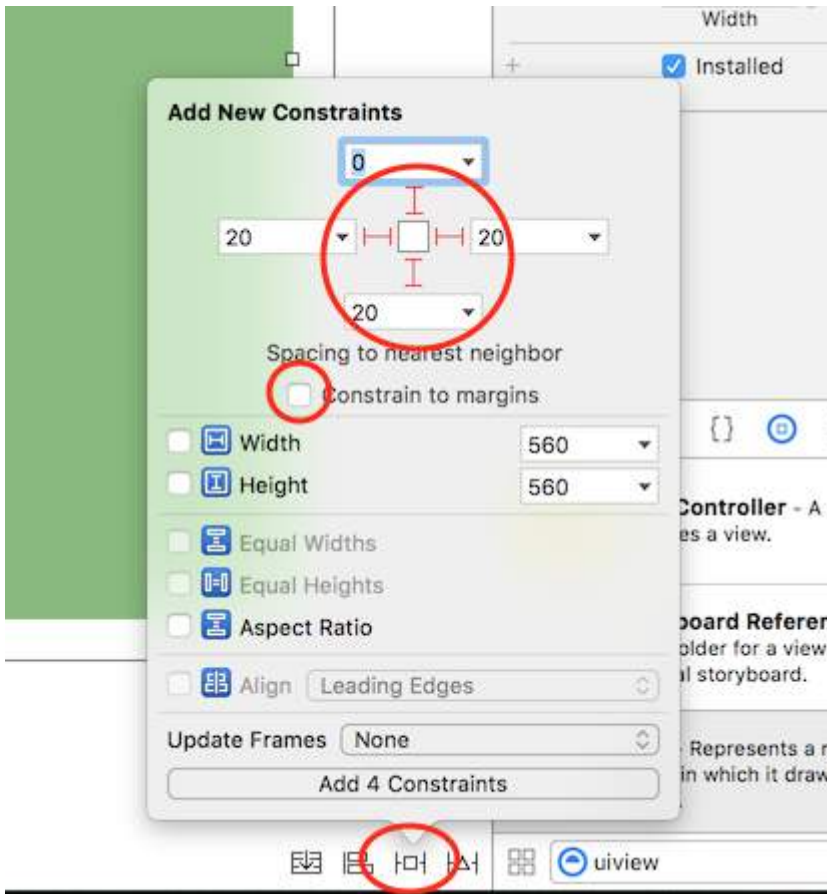
3. We can see the added constraints as:
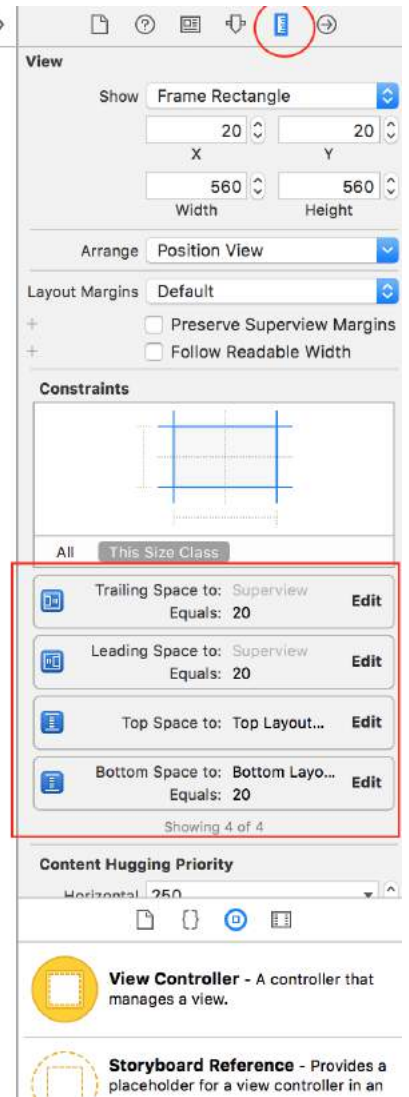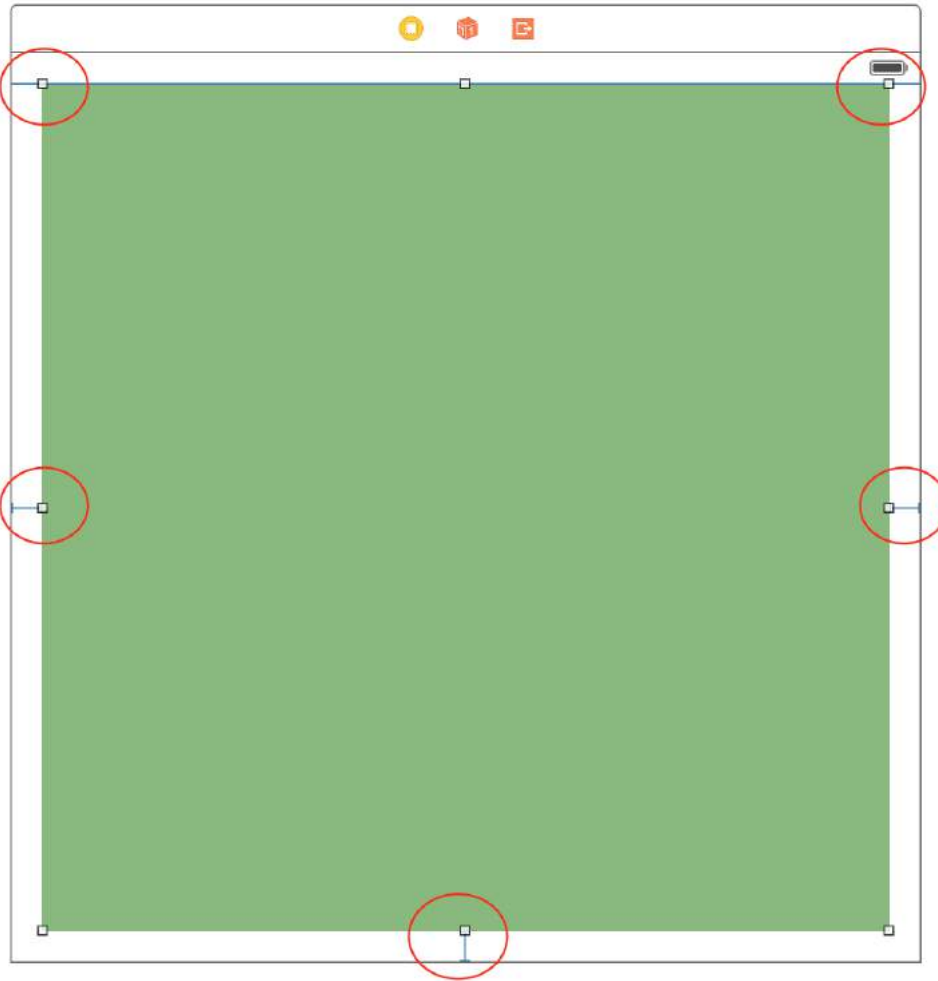
These constraints defines that the added view will be placed in it's superview as

```
CGRect(20, 0, superview.width - 20, superview.height - 20)
```

4. To see the preview on screen of these added constraints we can use **Assistant Editor** as;
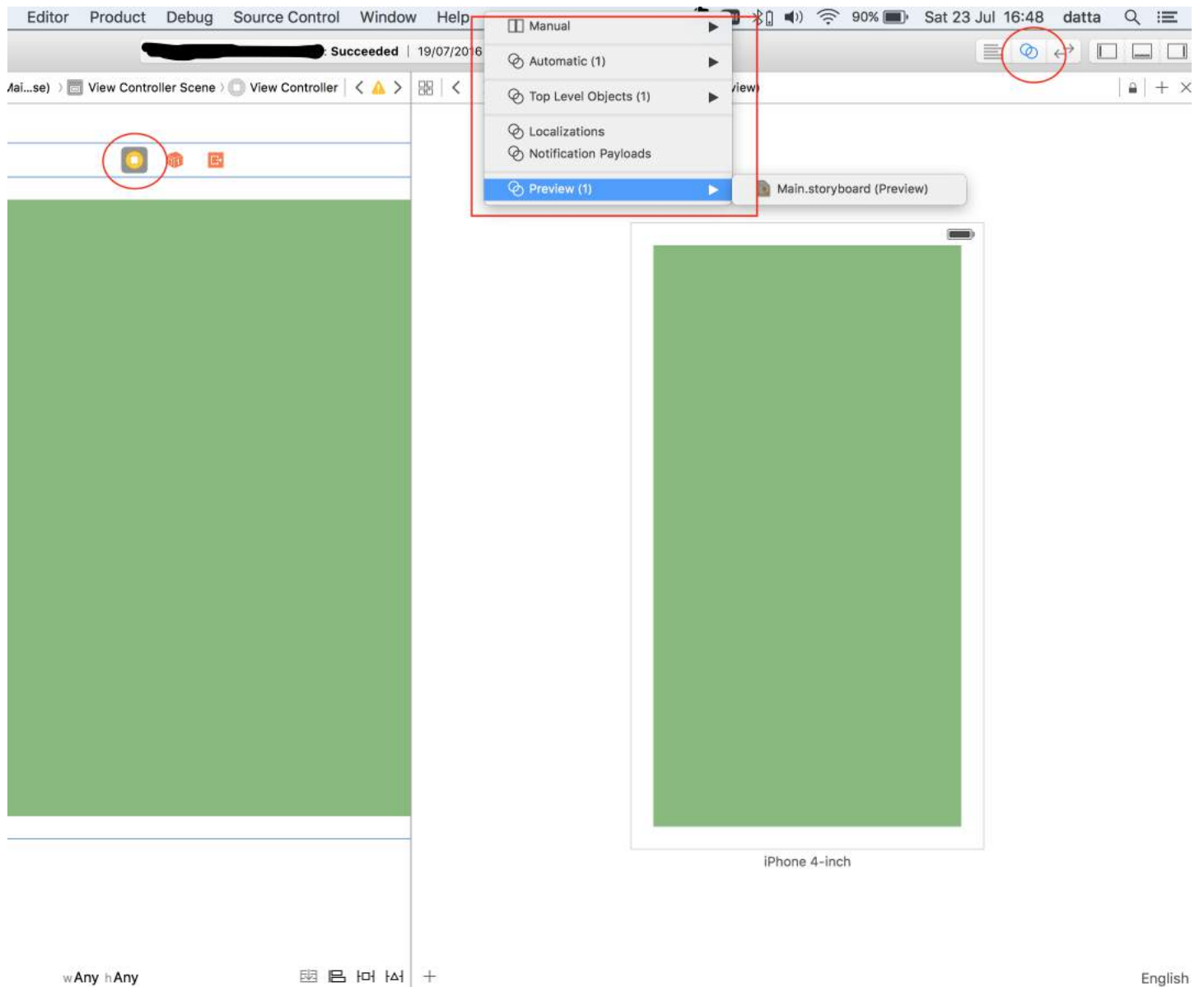
5. We can add more screen to see preview like:
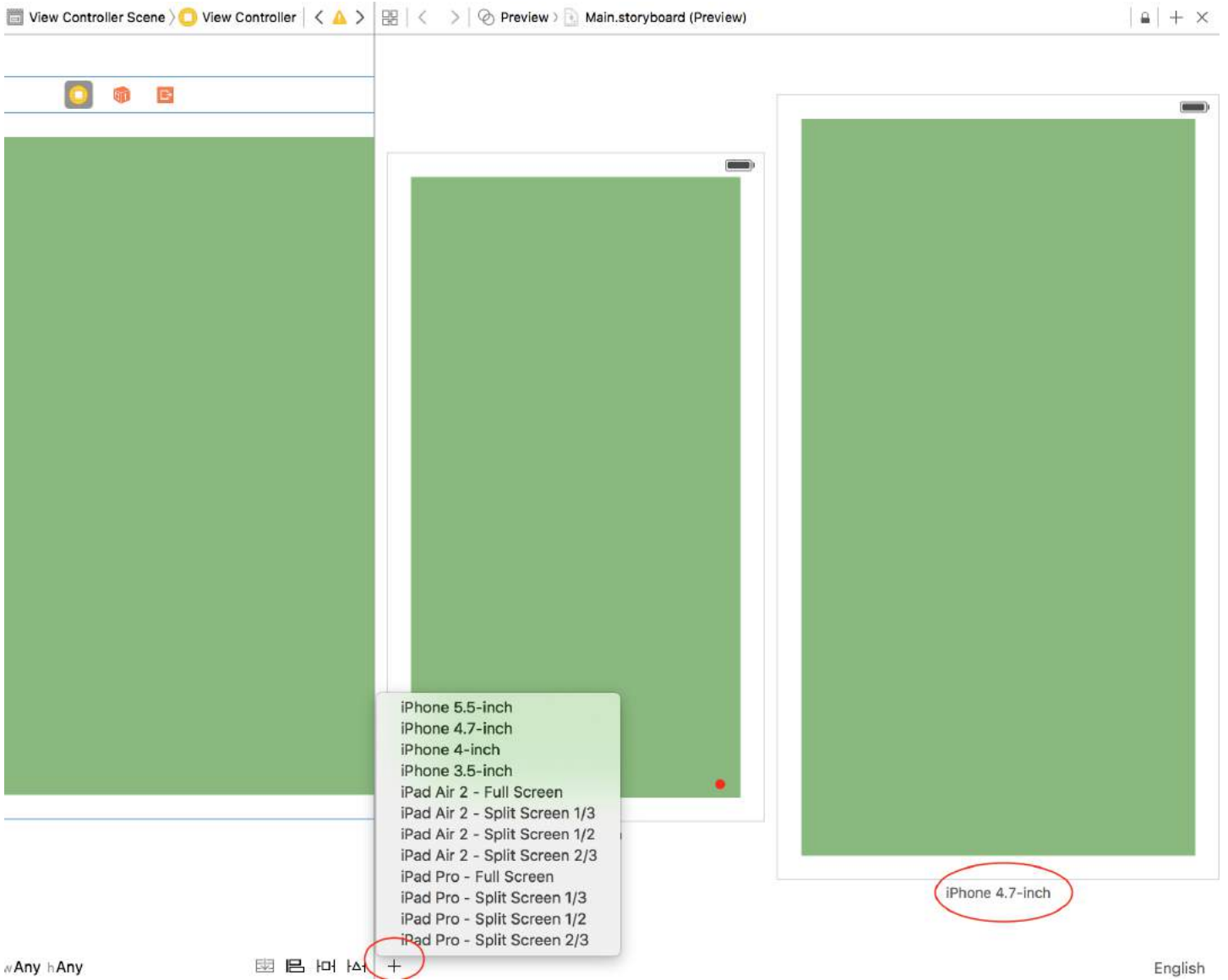
We can also see the preview with landscape mode by moving mouse on the name of device and clicking the rotation button as:
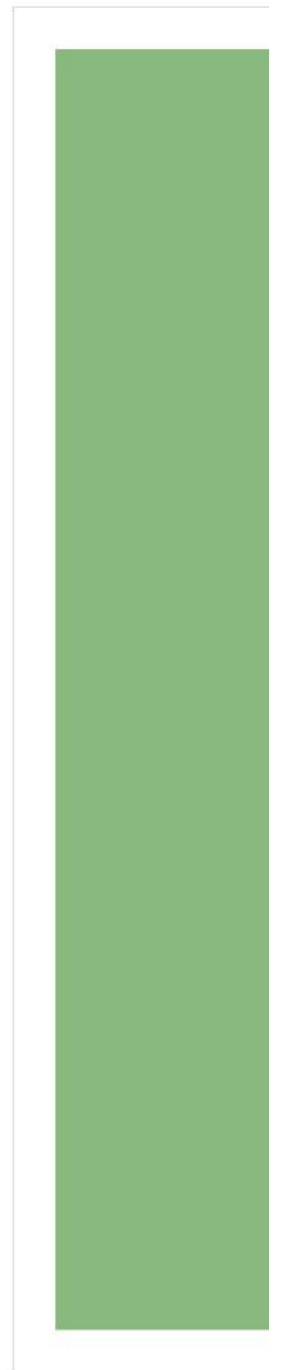
# Chapter 146: MKDistanceFormatter

## Section 146.1: String from distance

Given a `CLLocationDistance` (simply a `Double` representing meters), output a user-readable string:

```
let distance = CLLocationDistance(42)
let formatter = MKDistanceFormatter()
let answer = formatter.stringFromDistance(distance)
// answer = "150 feet"
```

**Objective-C**

```
CLLocationDistance distance=42;
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
NSString *answer=[formatter stringFromDistance:distance];
// answer = "150 feet"
```

By default, this respects the user's locale.

## Section 146.2: Distance units

`import` Mapkit Set units to one of `.Default`, `.Metric`, `.Imperial`, `.ImperialWithYards`:

```
formatter.units = .Metric
var answer = formatter.stringFromDistance(distance)
// "40 m"

formatter.units = .ImperialWithYards
answer = formatter.stringFromDistance(distance)
// "50 yards"
```

**Objective-C**

```
MKDistanceFormatter *formatter=[[MKDistanceFormatter alloc]init];
formatter.units=MKDistanceFormatterUnitsMetric;
NSString *answer=[formatter stringFromDistance:distance];
//40 m

formatter.units=MKDistanceFormatterUnitsImperialWithYards;
NSString *answer=[formatter stringFromDistance:distance];
//50 yards
```

## Section 146.3: Unit style

Set unitStyle to one of `.Default`, `.Abbreviated`, `.Full`:

```
formatter.unitStyle = .Full
var answer = formatter.stringFromDistance(distance)
// "150 feet"

formatter.unitStyle = .Abbreviated
answer = formatter.stringFromDistance(distance)
// "150 ft"
```

**Objective-C**

```
formatter.unitStyle=MKDistanceFormatterUnitStyleFull;
NSString *answer=[formatter stringFromDistance:distance];
// "150 feet"

 formatter.unitStyle=MKDistanceFormatterUnitStyleAbbreviated;
 NSString *answer=[formatter stringFromDistance:distance];
// "150 ft"
```

# Chapter 147: 3D Touch

## Section 147.1: 3D Touch with Swift

3D touch has been introduced with iPhone 6s Plus. There are two behaviors added with this new interface layer: Peek and Pop.

**Peek and Pop in a nutshell**

Peek - Press hard

Pop - Press really hard

**Checking for 3D support**

You should check if the device has a 3D touch support. You can do this by checking the value of *forceTouchCapability* property of a *UITraitCollection* object. UITraitCollection describes the iOS interface environment for your app.

```
if (traitCollection.forceTouchCapability == .Available) {
    registerForPreviewingWithDelegate(self, sourceView: view)
}
```

**Implementing the delegate**

You need to implement the two methods of *UIViewControllerPreviewingDelegate* in your class. One of the methods is for *peek* and the other one is for *pop* behavior.

The method to be implemented for the *peek* is *previewingContext*.

```
func previewingContext(previewingContext: UIViewControllerPreviewing, viewControllerForLocation
location: CGPoint) -> UIViewController? {

    guard let indexPath = self.tableView.indexPathForRowAtPoint(location), cell =
self.tableView.cellForRowAtIndexPath(indexPath) as? <YourTableViewCell> else {
        return nil
    }

    guard let datailVC =
storyboard?.instantiateViewControllerWithIdentifier("<YourViewControllerIdentifier>") as?
<YourViewController> else {
        return nil
    }

    datailVC.peekActive = true
    previewingContext.sourceRect = cell.frame

    // Do the stuff

    return datailVC

}
```

The method to be implemented for the *pop* is *previewingContext*. :)

```
func previewingContext(previewingContext: UIViewControllerPreviewing, commitViewController
viewControllerToCommit: UIViewController) {
```

```
    let balanceViewController = viewControllerToCommit as! <YourViewController>

    // Do the stuff

    navigationController?.pushViewController(balanceViewController, animated: true)

}
```

As you can see they are overloaded methods. You can use 3D touch in any way implementing these methods.

**Objective-C**

```
//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
            viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```

# Section 147.2: 3 D Touch Objective-C Example

**Objective-C**

```
//Checking for 3-D Touch availability
if ([self.traitCollection respondsToSelector:@selector(forceTouchCapability)] &&
        (self.traitCollection.forceTouchCapability == UIForceTouchCapabilityAvailable))
    {
        [self registerForPreviewingWithDelegate:self sourceView:self.view];
    }
//Peek
- (UIViewController *)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
            viewControllerForLocation:(CGPoint)location {

    NSIndexPath *indexPath = [self.tableView indexPathForRowAtPoint:location];
    Country *country = [self countryForIndexPath:indexPath];
    if (country) {
```

```objectivec
        CountryCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
        if (cell) {
            previewingContext.sourceRect = cell.frame;
            UINavigationController *navController = [self.storyboard
instantiateViewControllerWithIdentifier:@"UYLCountryNavController"];
            [self configureNavigationController:navController withCountry:country];
            return navController;
        }
    }
    return nil;
}
//Pop
- (void)previewingContext:(id<UIViewControllerPreviewing>)previewingContext
commitViewController:(UIViewController *)viewControllerToCommit {

    [self showDetailViewController:viewControllerToCommit sender:self];
}
```

# Chapter 148: GameCenter Leaderboards

## Section 148.1: GameCenter Leaderboards

**Prerequisites:**

1. Apple Developers Account
2. Setup GameCenter Leaderboards with iTunesConnect

**Setting up GameCenter Leaderboards:**

1. Sign in to *iTunesConnect*
2. Go to *My Apps*. Create an app for your project then go to *Features*.
3. Click on *Game Center*
4. Click the plus sign next to Leaderboards.
5. Choose *Single Leaderboard* for Leaderboard types.
6. Create a *Leaderboard Reference Name* for your reference.
7. Create a *Leaderboard ID* for your app to refer to when reporting scores.
8. Set score format to *Integer*
9. Score Submission will be *Best Score*
10. Click *Add language* and fill the entries.

Copy your `LeaderboardID` that you made and lets head over to Xcode.

Working with Xcode

There are 4 functions that we will be working with.

1. Importing the framework and setting up the protocols

2. Checking if the user is signed in to GameCenter

3. Reporting the scores to GameCenter

4. Viewing leaderboards

5. Import GameKit `import GameKit` Protocols `GKGameCenterControllerDelegate`

6. Now we want to check if the user is signed in to GameCenter

```
func authenticateLocalPlayer() {

    let localPlayer = GKLocalPlayer.localPlayer()
    localPlayer.authenticateHandler = { (viewController, error) -> Void in

        if viewController != nil {
            //If the user is not signed in to GameCenter, we make them sign in
            let vc:UIViewController = self.view!.window!.rootViewController!
            vc.presentViewController(viewController!, animated: true, completion: nil)

        } else {

            //Do something here if you want
        }
    }
}
```

3. Now the user is using the app and suddenly the user has a new high score, we report the high score by calling the function below.

The function below hols 2 parameters.

`Identifier` which is defined as a string and used to enter your leaderboardID that you made in iTunesConnect.

`score` which is defined as an Int which will be the users score to submit to iTunesConnect

```swift
func saveHighScore(identifier:String, score:Int) {

    if GKLocalPlayer.localPlayer().authenticated {

        let scoreReporter = GKScore(leaderboardIdentifier: identifier)

        scoreReporter.value = Int64(score)

        let scoreArray:[GKScore] = [scoreReporter]

        GKScore.reportScores(scoreArray, withCompletionHandler: {
            error -> Void in

            if error != nil {
                print("Error")
            } else {


            }
        })
    }
}
```

4. Now if the user wants to view leaderboards, call the function below

```swift
//This function will show GameCenter leaderboards and Achievements if you call this function.
    func showGameCenter() {

        let gameCenterViewController = GKGameCenterViewController()
        gameCenterViewController.gameCenterDelegate = self

        let vc:UIViewController = self.view!.window!.rootViewController!
        vc.presentViewController(gameCenterViewController, animated: true, completion:nil)

    }

    //This function closes gameCenter after showing.
    func gameCenterViewControllerDidFinish(gameCenterViewController: GKGameCenterViewController) {

        gameCenterViewController.dismissViewControllerAnimated(true, completion: nil)
        self.gameCenterAchievements.removeAll()

    }
```

# Chapter 149: Keychain

## Section 149.1: Adding a Password to the Keychain

Every Keychain Item is most often represented as a `CFDictionary`. You can, however, simply use `NSDictionary` in Objective-C and take advantage of bridging, or in Swift you may use `Dictionary` and explicitly cast to `CFDictionary`.

You could construct a password with the following dictionary:

**Swift**

```
var dict = [String : AnyObject]()
```

First, you need a key/value pair that lets the Keychain know this is a password. Note that because our dict key is a `String` we must cast any `CFString` to a `String` explicitly in Swift 3. CFString may not be used as the key to a Swift Dictionary because it is not Hashable.

**Swift**

```
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, our password may have a series of attributes to describe it and help us find it later. [Here's a list of attributes for generic passwords](#).

**Swift**

```
// The password will only be accessible when the device is unlocked
dict[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked
// Label may help you find it later
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
// Service name
dict[kSecAttrService as String] = "MyService" as CFString
```

Finally, we need our actual private data. Be sure not to keep this around in memory for too long. This must be `CFData`.

**Swift**

```
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Finally, the Keychain Services add function wants to know how it should return the newly constructed keychain item. Since you shouldn't be holding on to the data very long in memory, here's how you could only return the attributes:

**Swift**

```
dict[kSecReturnAttributes as String] = kCFBooleanTrue
```

Now we have constructed our item. Let's add it:

**Swift**

```
var result: AnyObject?
let status = withUnsafeMutablePointer(to: &result) {
    SecItemAdd(dict as CFDictionary, UnsafeMutablePointer($0))
}
let newAttributes = result as! Dictionary<String, AnyObject>
```

This places the new attributes dict inside `result`. `SecItemAdd` takes in the dictionary we constructed, as well as a pointer to where we would like our result. The function then returns an `OSStatus` indicating success or an error code. Result codes are described [here].

## Section 149.2: Keychain Access Control (TouchID with password fallback)

Keychain allows to save items with special SecAccessControl attribute which will allow to get item from Keychain only after user will be authenticated with Touch ID (or passcode if such fallback is allowed). App is only notified whether the authentication was successful or not, whole UI is managed by iOS.

First, SecAccessControl object should be created:

**Swift**

```swift
let error: Unmanaged<CFError>?

guard let accessControl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly, .userPresence, &error) else {
    fatalError("Something went wrong")
}
```

Next, add it to the dictionary with kSecAttrAccessControl key (which is mutually exclusive with kSecAttrAccessible key you've been using in other examples):

**Swift**

```swift
var dictionary = [String : Any]()

dictionary[kSecClass as String] = kSecClassGenericPassword
dictionary[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
dictionary[kSecAttrAccount as String] = "My Name" as CFString
dictionary[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
dictionary[kSecAttrAccessControl as String] = accessControl
```

And save it as you've done before:

**Swift**

```swift
let lastResultCode = SecItemAdd(query as CFDictionary, nil)
```

To access stored data, just query Keychain for a key. Keychain Services will present authentication dialog to the user and return data or nil depending on whether suitable fingerprint was provided or passcode matched.

Optionally, prompt string can be specified:

**Swift**

```swift
var query = [String: Any]()

query[kSecClass as String] = kSecClassGenericPassword
query[kSecReturnData as String] = kCFBooleanTrue
query[kSecAttrAccount as String] = "My Name" as CFString
query[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
query[kSecUseOperationPrompt as String] = "Please put your fingers on that button" as CFString

var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
```

```
}
```

Pay attention that `status` will be `err` if user declined, canceled or failed authorization.

**Swift**

```swift
if status == noErr {
    let password = String(data: queryResult as! Data, encoding: .utf8)!
    print("Password: \(password)")
} else {
    print("Authorization not passed")
}
```

# Section 149.3: Finding a Password in the Keychain

To construct a query, we need to represent it as a `CFDictionary`. You may also use `NSDictionary` in Objective-C or `Dictionary` in Swift and cast to `CFDictionary`.

We need a class key:

**Swift**

```swift
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

Next, we can specify attributes to narrow down our search:

**Swift**

```swift
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
// Service name
dict[kSecAttrService as String] = "MyService" as CFString
```

We can also specify special search modifier keys described [here](#).

Finally, we need to say how we'd like our data returned. Below, we'll request that just the private password itself be returned as a `CFData` object:

**Swift**

```swift
dict[kSecReturnData as String] = kCFBooleanTrue
```

Now, let's search:

**Swift**

```swift
var queryResult: AnyObject?
let status = withUnsafeMutablePointer(to: &queryResult) {
    SecItemCopyMatching(dict as CFDictionary, UnsafeMutablePointer($0))
}
// Don't keep this in memory for long!!
let password = String(data: queryResult as! Data, encoding: .utf8)!
```

Here, `SecItemCopyMatching` takes in a query dictionary and a pointer to where you'd like the result to go. It returns an `OSStatus` with a result codes. [Here](#) are the possibilities.

# Section 149.4: Updating a Password in the Keychain

As usual, we first need a `CFDictionary` to represent the item we want to update. This must contain all of the old values for the item, including the old private data. Then it takes a `CFDictionary` of any attributes or the data itself that you would like to change.

So first, let's construct a class key and a list of attributes. These attributes can narrow our search but you must include any attributes and there old values if you will be changing them.

**Swift**

```swift
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

Now we must add the old data:

**Swift**

```swift
dict[kSecValueData as String] = "my_password!!".data(using: .utf8) as! CFData
```

Now let's create the same attributes but a different password:

**Swift**

```swift
var newDict = [String : AnyObject]()
newDict[kSecClass as String] = kSecClassGenericPassword
// Label
newDict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
newDict[kSecAttrAccount as String] = "My Name" as CFString
// New password
newDict[kSecValueData as String] = "new_password!!".data(using: .utf8) as! CFData
```

Now, we just pass it to Keychain Services:

**Swift**

```swift
let status = SecItemUpdate(dict as CFDictionary, newDict as CFDictionary)
```

SecItemUpdate returns a status code. Results are described [here](#).

# Section 149.5: Removing a Password from the Keychain

We need only one thing in order to delete an item from the Keychain: a `CFDictionary` with attributes describing the items to be deleted. Any items that match the query dictionary will be deleted permanently, so if you are only intending to delete a single item be sure to be specific with your query. As always, we can use an `NSDictionary` in Objective-C or in Swift we can use a `Dictionary` and then cast to `CFDictionary`.

A query dictionary, in this context exclusively includes a class key to describe what the item is and attributes to describe information about the item. Inclusion of search restrictions such as `kSecMatchCaseInsensitive` is not allowed.

**Swift**

```swift
var dict = [String : AnyObject]()
dict[kSecClass as String] = kSecClassGenericPassword
```

```
// Label
dict[kSecAttrLabel as String] = "com.me.myapp.myaccountpassword" as CFString
// Username
dict[kSecAttrAccount as String] = "My Name" as CFString
```

And now we can simply remove it:

**Swift**

```
let status = SecItemDelete(dict as CFDictionary)
```

SecItemDelete returns an OSStatus. Result codes are described [here](here).

# Section 149.6: Keychain Add, Update, Remove and Find operations using one file

**Keychain.h**

```
#import <Foundation/Foundation.h>
typedef void (^KeychainOperationBlock)(BOOL successfulOperation, NSData *data, OSStatus status);

@interface Keychain : NSObject

-(id) initWithService:(NSString *) service_ withGroup:(NSString*)group_;

-(void)insertKey:(NSString *)key withData:(NSData *)data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)updateKey:(NSString*)key withData:(NSData*) data
withCompletion:(KeychainOperationBlock)completionBlock;
-(void)removeDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;
-(void)findDataForKey:(NSString*)key withCompletionBlock:(KeychainOperationBlock)completionBlock;

@end
```

Keychain.m

```
#import "Keychain.h"
#import <Security/Security.h>

@implementation Keychain

{
    NSString * keychainService;
    NSString * keychainGroup;
}

-(id) initWithService:(NSString *)service withGroup:(NSString*)group
{
    self =[super init];
    if(self) {
        keychainService = [NSString stringWithString:service];
        if(group) {
            keychainGroup = [NSString stringWithString:group];
        }
    }

    return  self;
}

-(void)insertKey:(NSString *)key
```

```
            withData:(NSData *)data
   withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dict =[self prepareDict:key];
    [dict setObject:data forKey:(__bridge id)kSecValueData];
    [dict setObject:keychainService forKey:(id)kSecAttrService];

    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dict, NULL);
    if(errSecSuccess != status) {
        DLog(@"Unable add item with key =%@ error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    }
    if (status == errSecDuplicateItem) {
        [self updateKey:key withData:data withCompletion:^(BOOL successfulOperation, NSData
*updateData, OSStatus updateStatus) {
            if (completionBlock) {
                completionBlock(successfulOperation, updateData, updateStatus);
            }
            DLog(@"Found duplication item -- updating key with data");
        }];
    }
}

-(void)findDataForKey:(NSString *)key
   withCompletionBlock:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary *dict = [self prepareDict:key];
    [dict setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [dict setObject:keychainService forKey:(id)kSecAttrService];
    [dict setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFTypeRef result = NULL;
    OSStatus status = SecItemCopyMatching((__bridge CFDictionaryRef)dict,&result);

    if( status != errSecSuccess) {
        DLog(@"Unable to fetch item for key %@ with error:%d",key,(int)status);
        if (completionBlock) {
            completionBlock(errSecSuccess == status, nil, status);
        }
    } else {
        if (completionBlock) {
            completionBlock(errSecSuccess == status, (__bridge NSData *)result, status);
        }
    }
}

-(void)updateKey:(NSString *)key
         withData:(NSData *)data
   withCompletion:(KeychainOperationBlock)completionBlock
{
    NSMutableDictionary * dictKey =[self prepareDict:key];

    NSMutableDictionary * dictUpdate =[[NSMutableDictionary alloc] init];
    [dictUpdate setObject:data forKey:(__bridge id)kSecValueData];
    [dictUpdate setObject:keychainService forKey:(id)kSecAttrService];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)dictKey, (__bridge
CFDictionaryRef)dictUpdate);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
```

```objc
        completionBlock(errSecSuccess == status, nil, status);
    }
}

-(void)removeDataForKey:(NSString *)key
    withCompletionBlock:(KeychainOperationBlock)completionBlock {
    NSMutableDictionary *dict = [self prepareDict:key];
    OSStatus status = SecItemDelete((__bridge CFDictionaryRef)dict);
    if( status != errSecSuccess) {
        DLog(@"Unable to remove item for key %@ with error:%d",key,(int)status);
    }
    if (completionBlock) {
        completionBlock(errSecSuccess == status, nil, status);
    }
}

#pragma mark Internal methods

-(NSMutableDictionary*) prepareDict:(NSString *) key {

    NSMutableDictionary *dict = [[NSMutableDictionary alloc] init];
    [dict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];

    NSData *encodedKey = [key dataUsingEncoding:NSUTF8StringEncoding];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrGeneric];
    [dict setObject:encodedKey forKey:(__bridge id)kSecAttrAccount];
    [dict setObject:keychainService forKey:(__bridge id)kSecAttrService];
    [dict setObject:(__bridge id)kSecAttrAccessibleAlwaysThisDeviceOnly forKey:(__bridge
id)kSecAttrAccessible];

    //This is for sharing data across apps
    if(keychainGroup != nil) {
        [dict setObject:keychainGroup forKey:(__bridge id)kSecAttrAccessGroup];
    }

    return  dict;
}

@end
```

# Chapter 150: Handle Multiple Environment using Macro

## Section 150.1: Handle multiple environment using multiple target and macro

For example, we have two environments: CI - Staging and want to add some customizations for each environment. Here I will try to customize server URL, app name.

First, we create two targets for 2 environments by duplicating the main target:
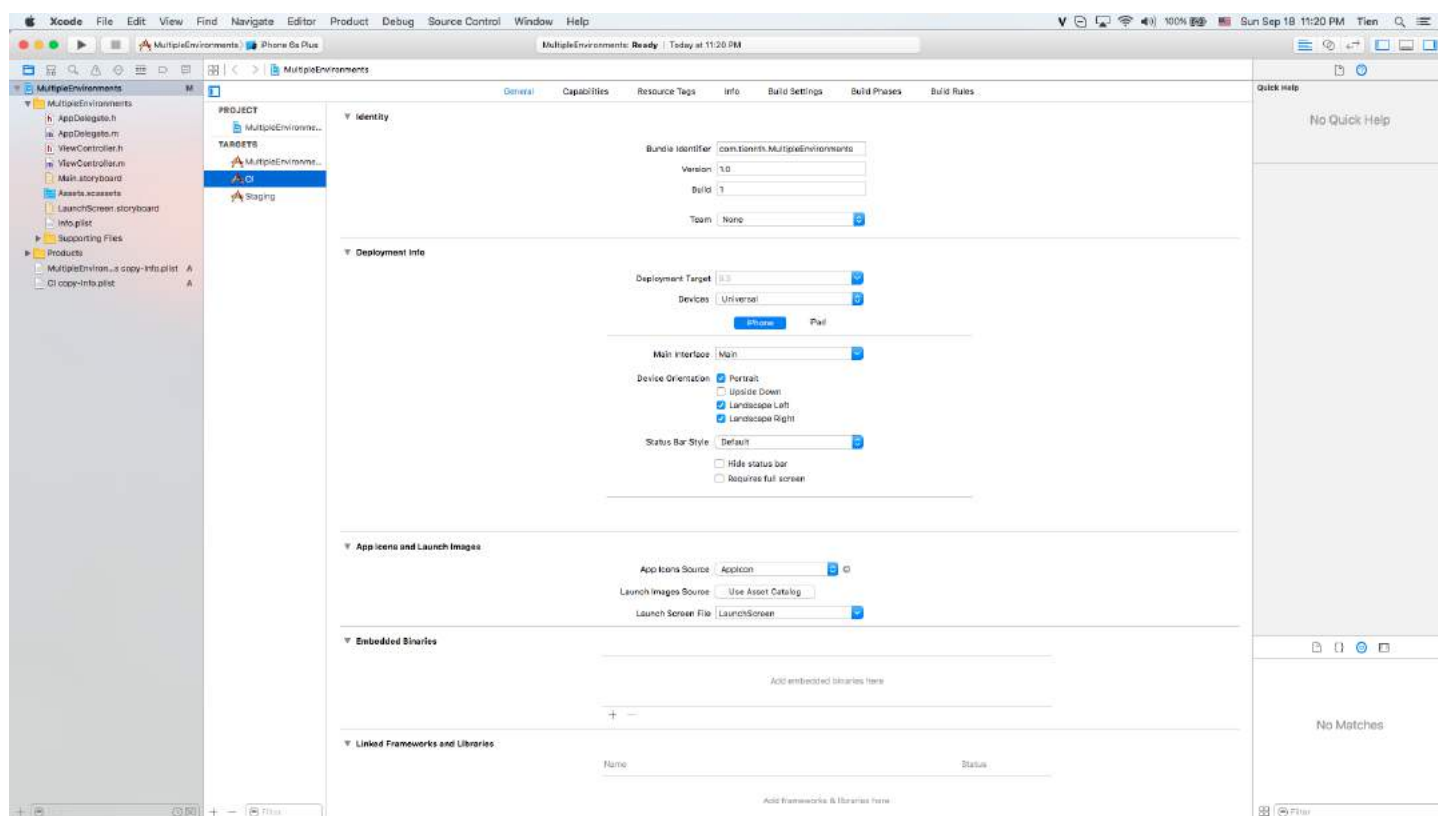


For each target, we will define a custom macro. Here I will define macro named "CI" in build settings of target CI, macro named "STAGING" for target Staging.

The development target (MultipleEnvironment target):
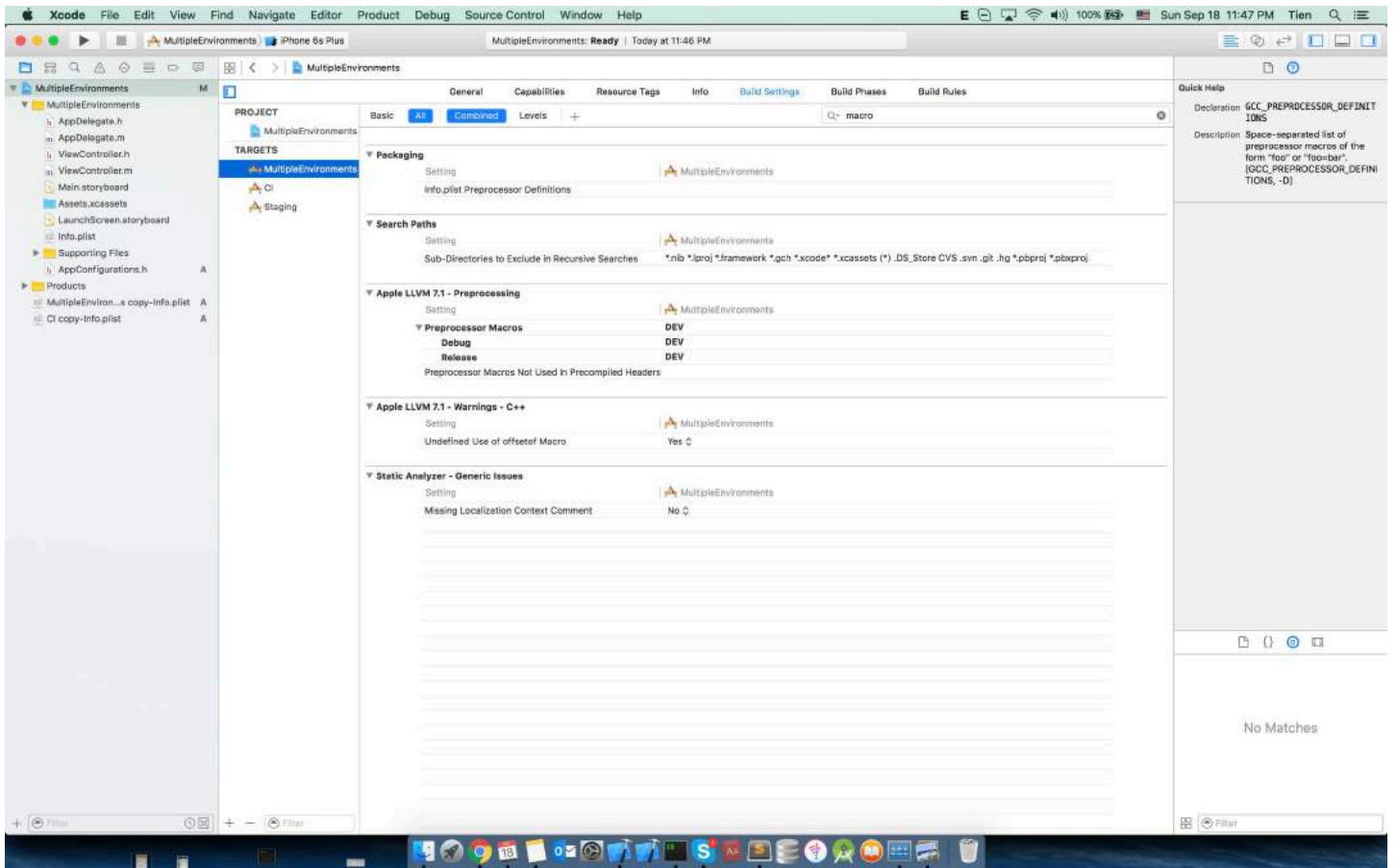
Target CI:
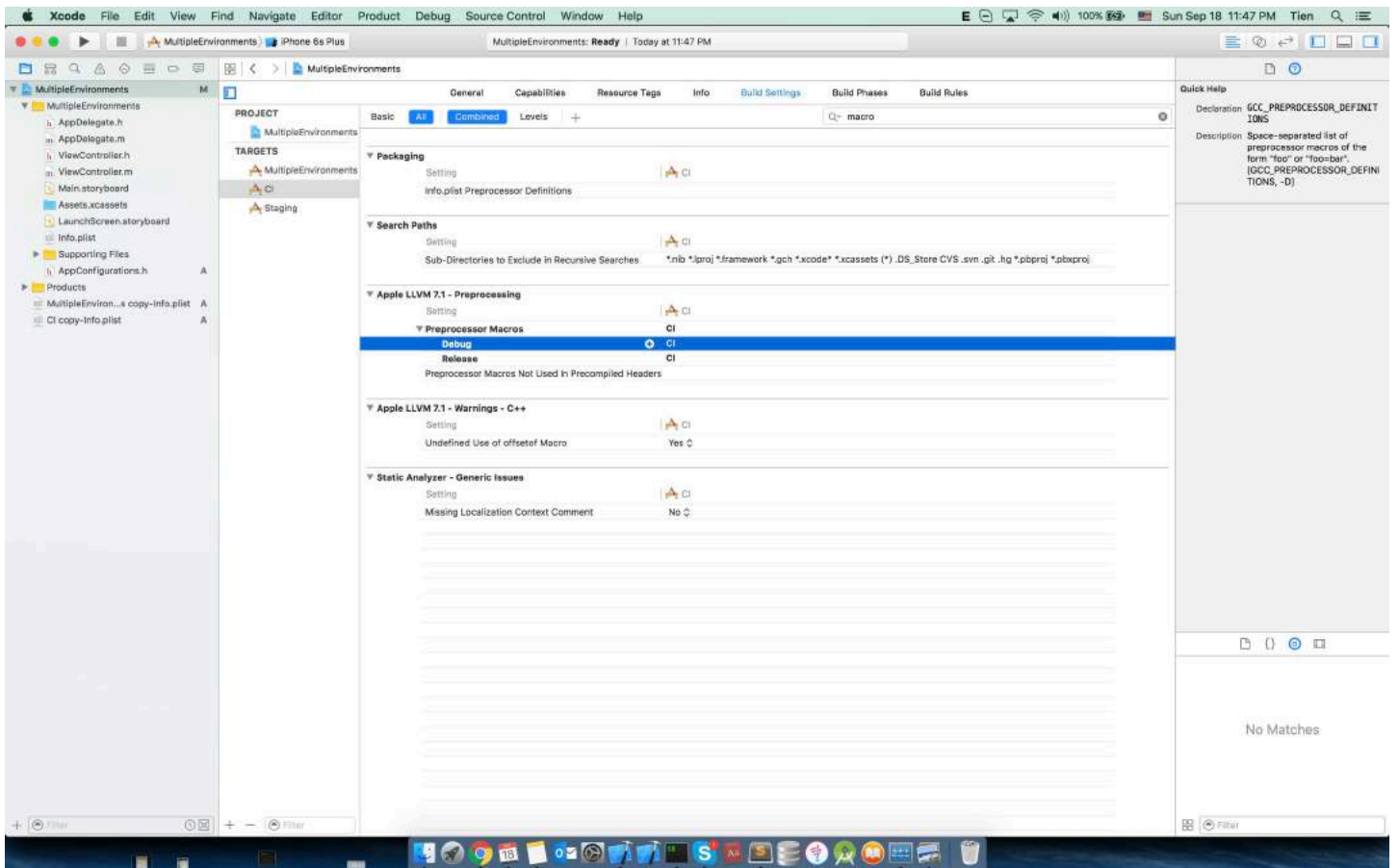


Target Staging:
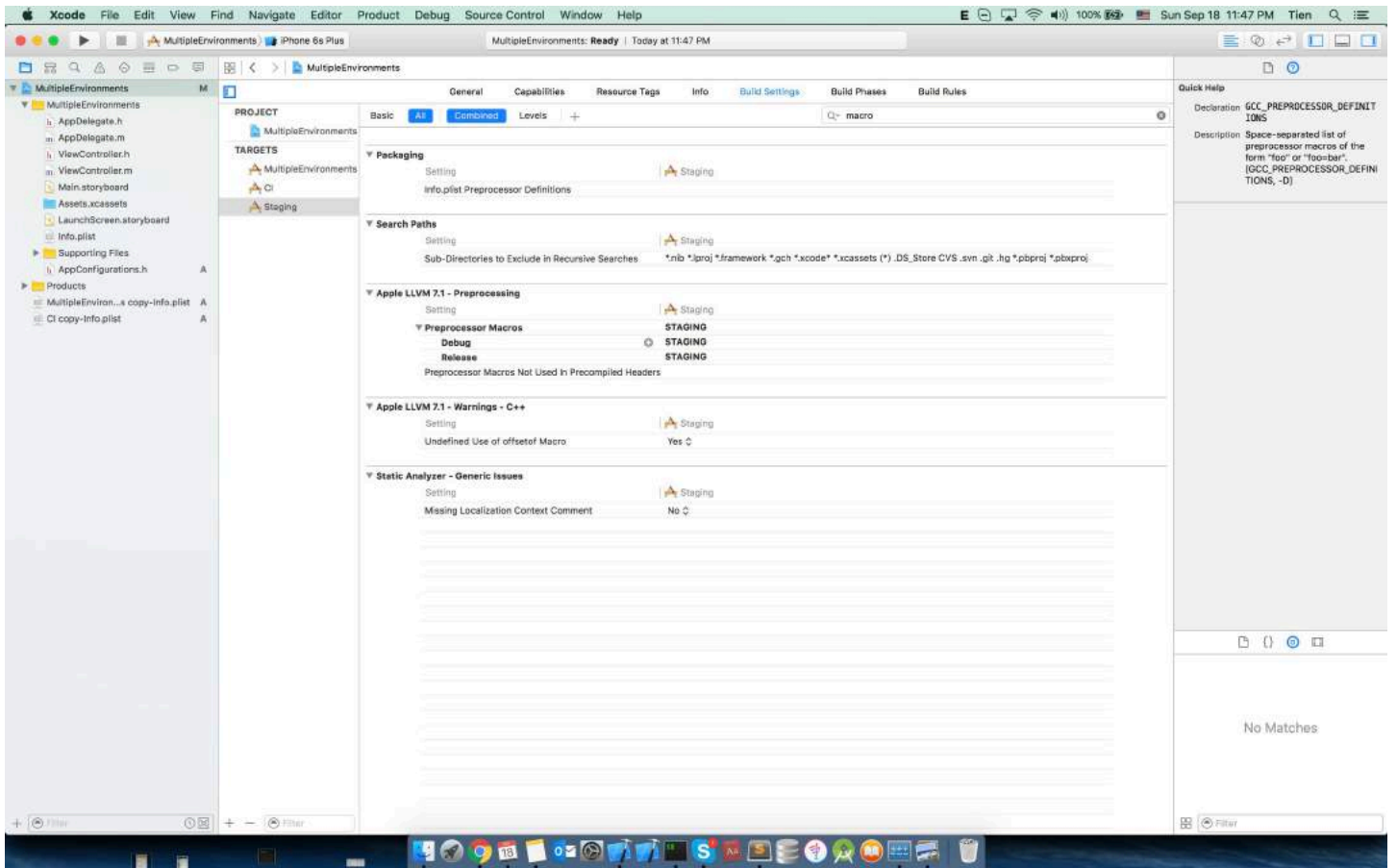
Create scheme for each target:
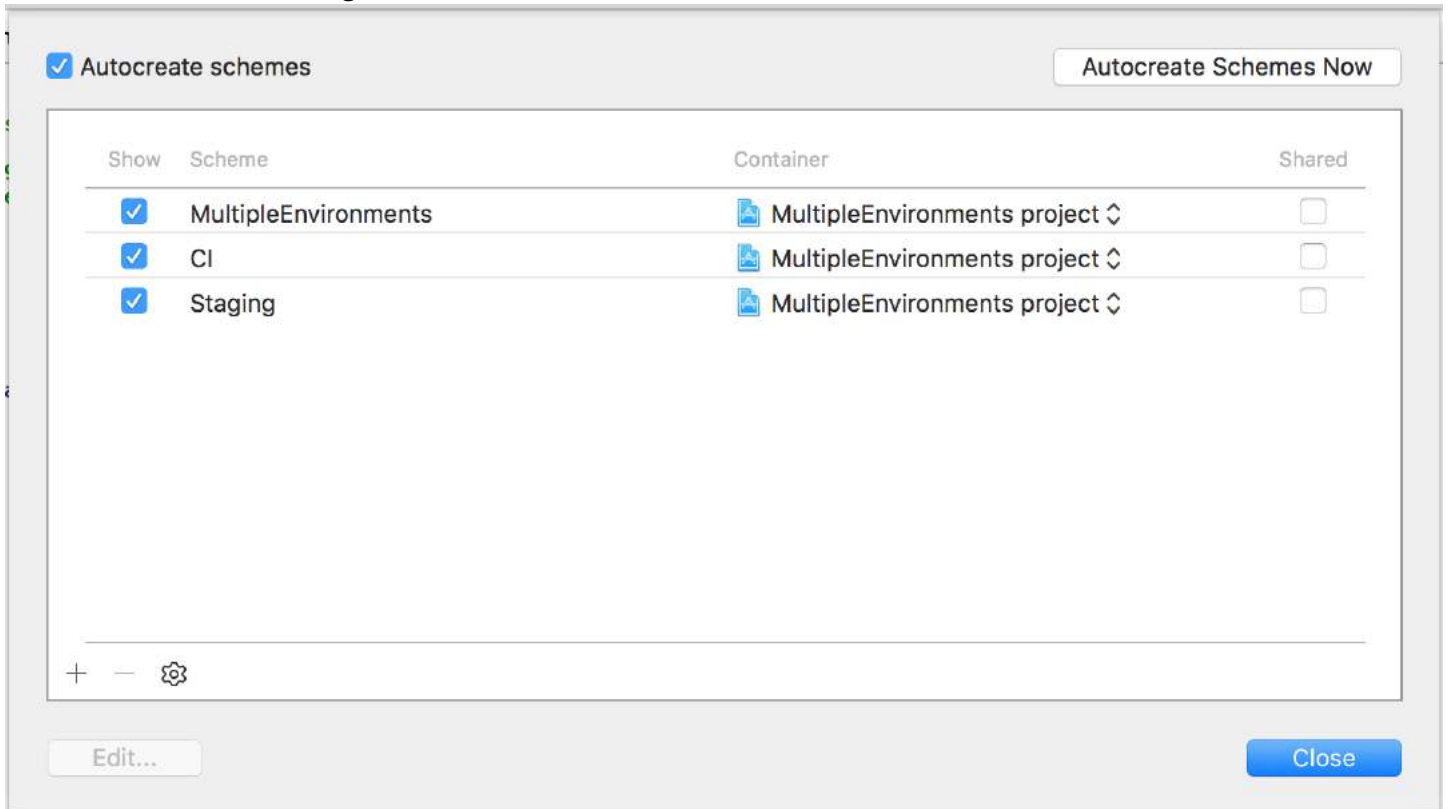


We will create a header file to define SERVER URL as below:
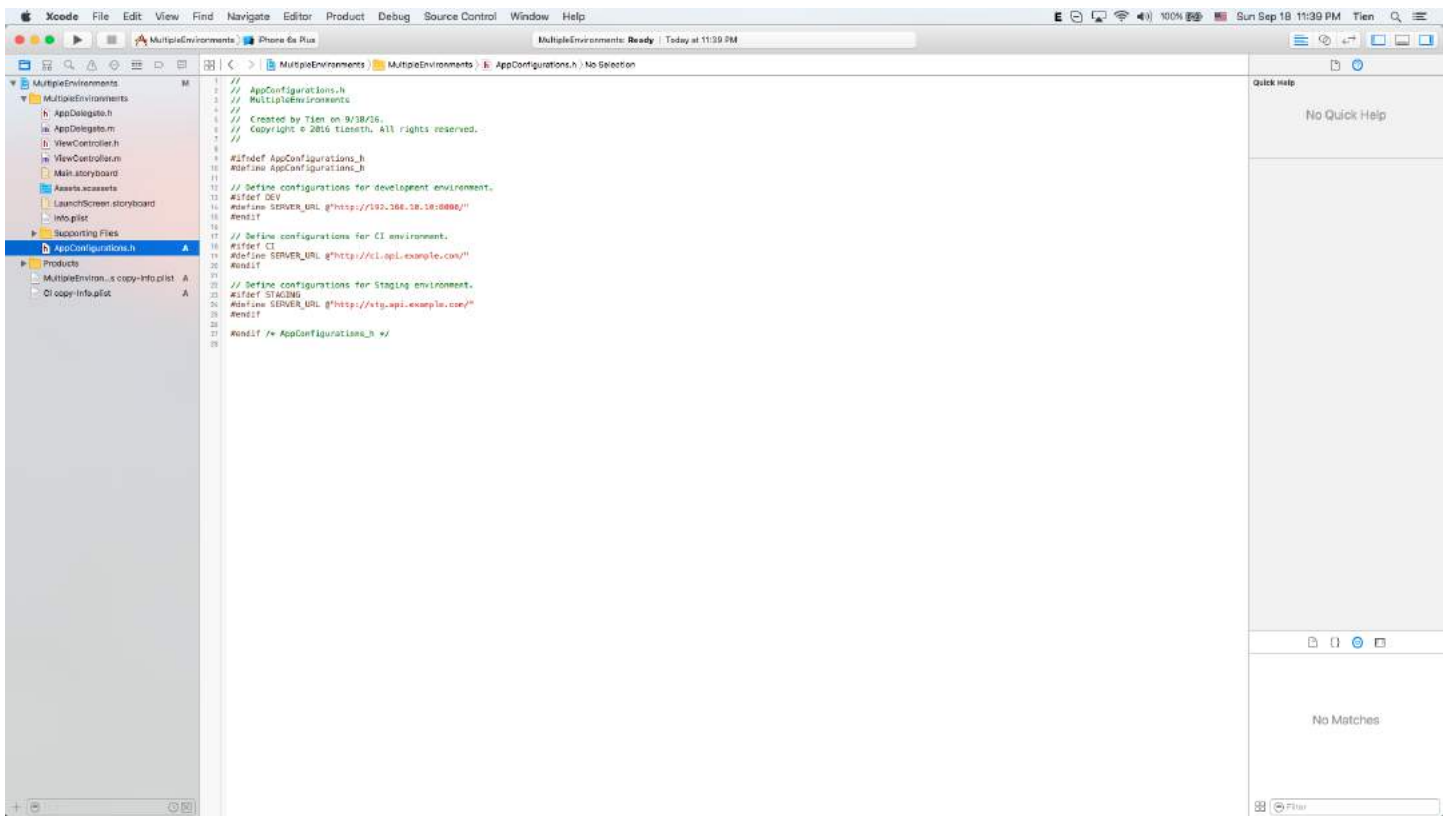
It means,

- If we run/archive using the default target (MultipleEnvironment), the SERVER_URL is
  http://192.168.10.10:8080/
- If we run/archive using CI target, the SERVER_URL is http://ci.api.example.com/
- If we run/archive using STAGING target, the SERVER_URL is http://stg.api.example.com/

If you want to do more customize, for example: Change app name for each target:
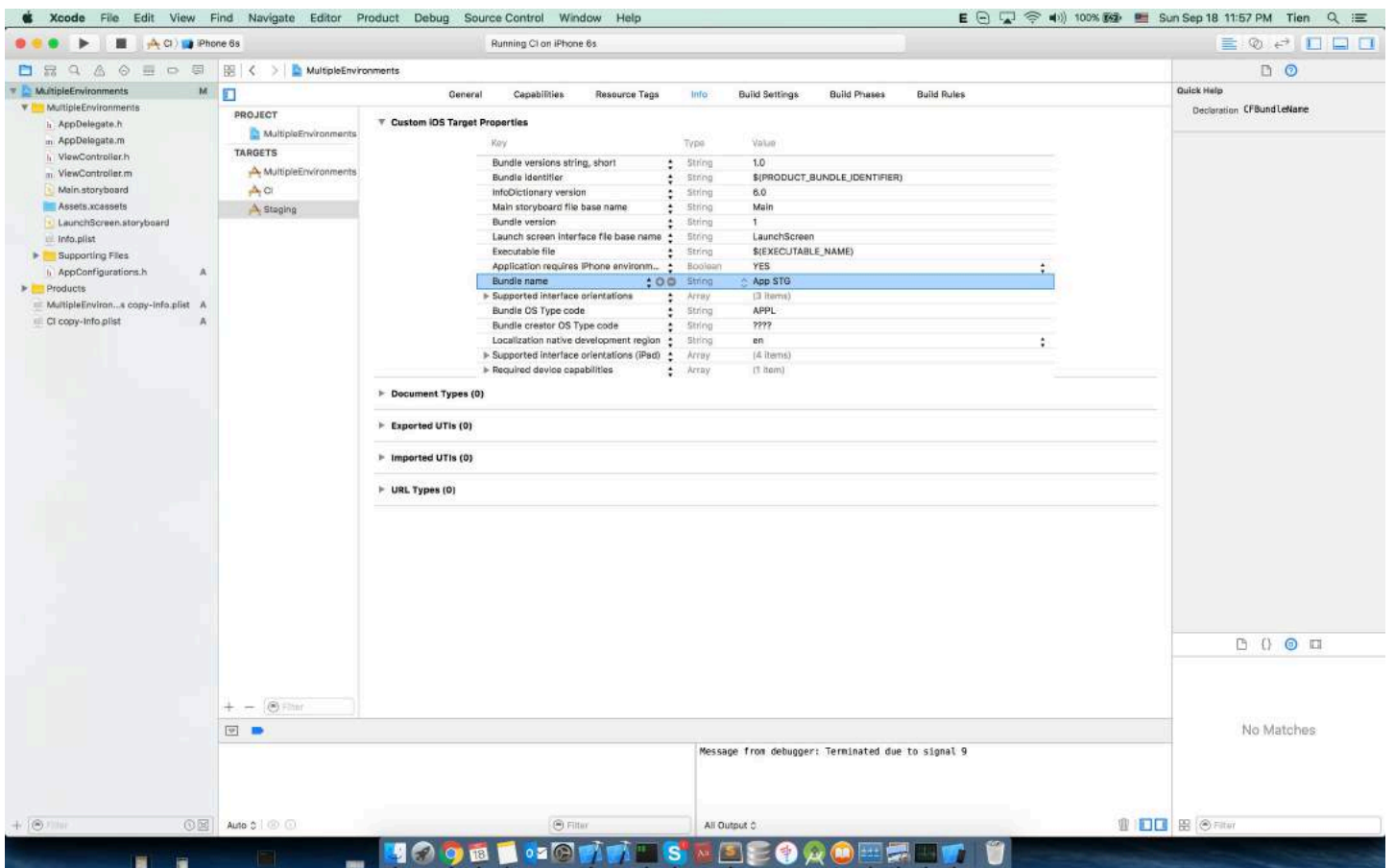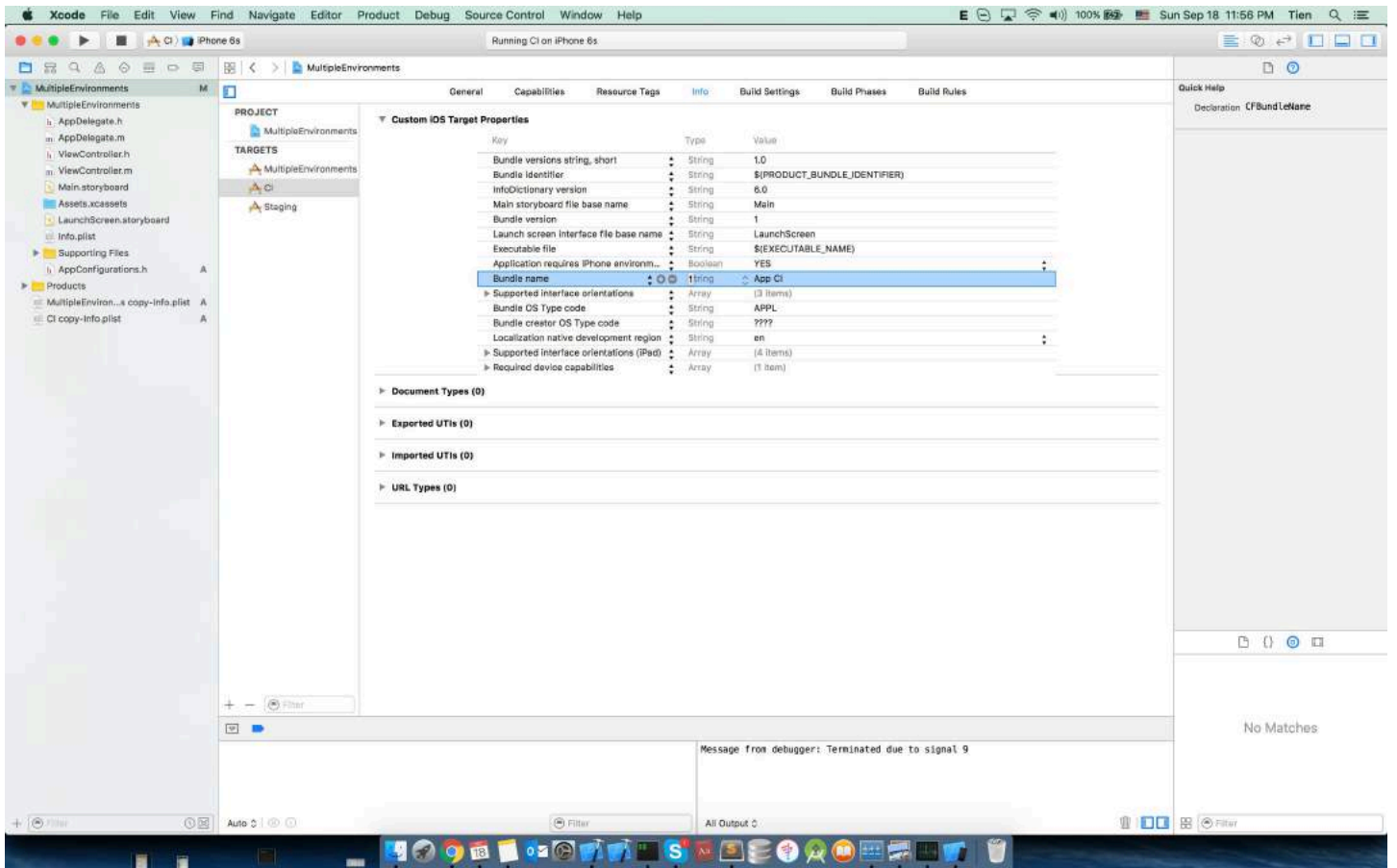
Almost done. Now we want to show current SERVER_URL to main screen:
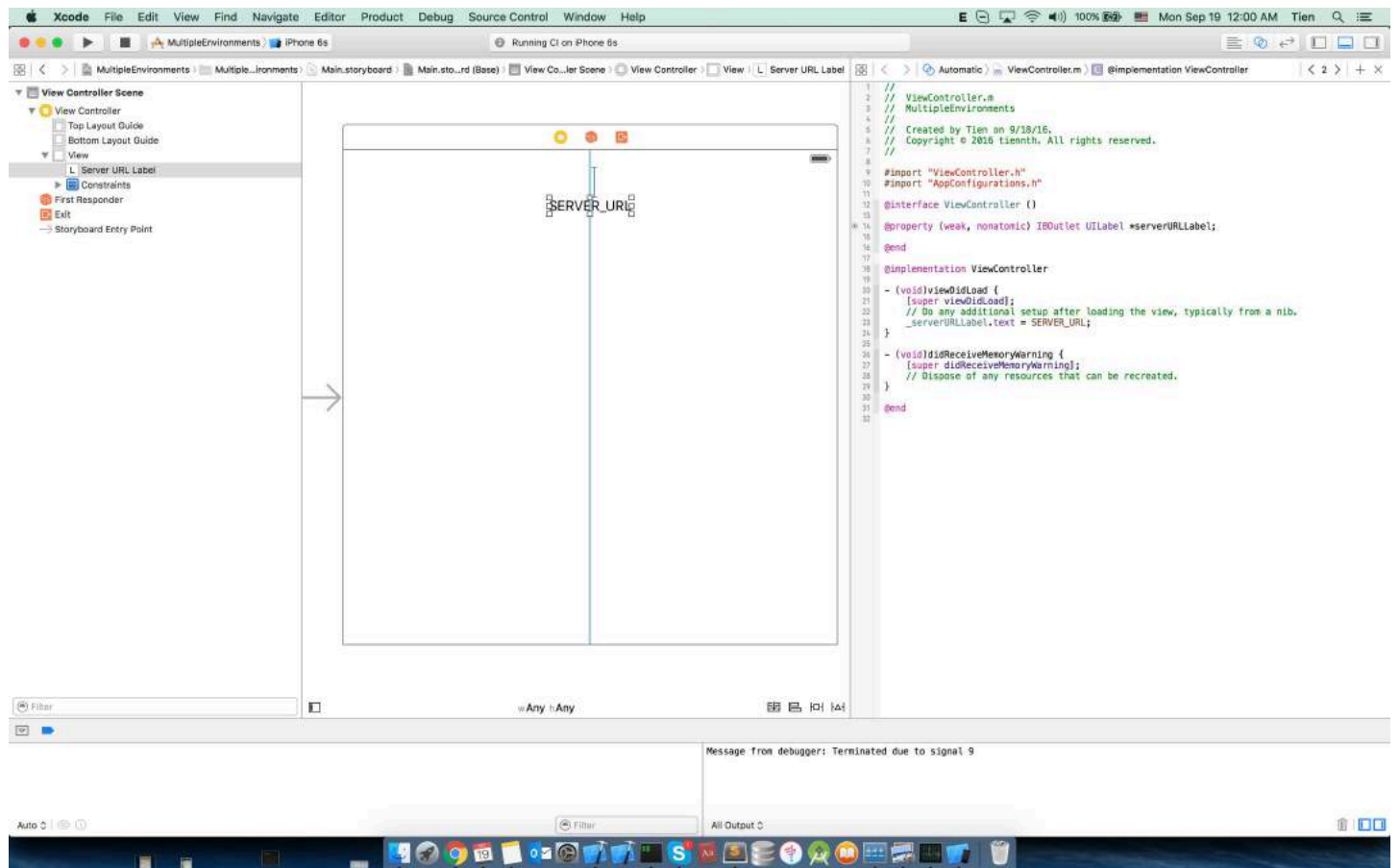
Now, let's see if we run the app with the default target (MultipleEnvironment)

Carrier 🔅     **12:03 AM**     🔋

# http://192.168.10.10:8080/

MultipleEn...

Safari

**CI target**:

http://ci.api.example.com/

App CI

Q • • • • •

Safari

**Staging target**:

http://stg.api.example.com/

As you can see, value of SERVER_URL and app name is changed for each target :)

# Chapter 151: Set View Background

## Section 151.1: Fill background Image of a UIView

**Objective-C**

```
UIGraphicsBeginImageContext(self.view.frame.size);
[[UIImage imageNamed:@"image.png"] drawInRect:self.view.bounds];
UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
self.view.backgroundColor = [UIColor colorWithPatternImage:image];
```

## Section 151.2: Creating a gradient background view

To create a background with a gradient you can use the CAGradientLayer class:

Swift 3.1:

```
func createGradient() {
    let caLayer = CAGradientLayer()
    caLayer.colors = [UIColor.white, UIColor.green, UIColor.blue]
    caLayer.locations = [0, 0.5, 1]
    caLayer.bounds = self.bounds
    self.layer.addSublayer(caLayer)
}
```

This can be called on viewDidLoad() like so:

```
override func viewDidLoad() {
    super.viewDidLoad()
    createGradient()
}
```

The CAGradientLayer locations and bounds variables can take multiple values to create a gradient layer with how ever many colors you desire. From the documentation:

> By default, the colors are spread uniformly across the layer, but you can optionally specify locations for control over the color positions through the gradient.

## Section 151.3: Set View backround with image

```
self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage imageNamed:@"Background.png"]];
```

## Section 151.4: Set View background

Objective C:

```
view.backgroundColor = [UIColor redColor];
```

Swift:

```
view.backgroundColor! = UIColor.redColor()
```

Swift 3

```
view.backgroundColor = UIColor.redColor
```

# Chapter 152: Block

## Section 152.1: Custom completion block for Custom Methods

**1- Define Your own custom Block**

```
typedef void(^myCustomCompletion)(BOOL);
```

**2- Create custom method which takes your custom completion block as a parameter.**

```
-(void) customMethodName:(myCustomCompletion) compblock{
    //do stuff
    // check if completion block exist; if we do not check it will throw an exception
    if(complblock)
        compblock(YES);
  }
```

3- How to use block in your Method

```
[self customMethodName:^(BOOL finished) {
if(finished){
    NSLog(@"success");
}
}];
```

## Section 152.2: UIView Animations

```
[UIView animateWithDuration:1.0
    animations:^{
        someView.alpha = 0;
        otherView.alpha = 1;
    }
    completion:^(BOOL finished) {
    [someView removeFromSuperview];
}];
```

The carat "^" character defines a block. For example, ^{ … } is a block. More specifically, it is a block that returns "void" and accepts no arguments. It is equivalent to a method such like: "- (void)something;" but there is no inherent name associated with the code block.

Define a block that can accept arguments work very similarly. To supply an argument to a block, you define the block like so: *^(BOOL someArg, NSString someStr) { … }*. When you use API calls that support blocks, you'll be writing blocks that look similar to this, especially for animation blocks or NSURLConnection blocks as shown in the above example.

## Section 152.3: Modify captured variable

Block will capture variables that appeared in the same lexical scope. Normally these variables are captured as "const" value:

```
int val = 10;
void (^blk)(void) = ^{
    val = 20; // Error! val is a constant value and cannot be modified!
};
```

In order to modify the variable, you need to use the __block storage type modifier.

```
__block int val = 10;
void (^blk)(void) = ^{
    val = 20; // Correct! val now can be modified as an ordinary variable.
};
```

# Chapter 153: Content Hugging/Content Compression in Autolayout

## Section 153.1: Definition: Intrinsic Content Size

Before Auto Layout, you always had to tell buttons and other controls how big they should be, either by setting their frame or bounds properties or by resizing them in Interface Builder. But it turns out that most controls are perfectly capable of determining how much space they need, based on their content.

A **label** knows how wide and tall it is because it knows the length of the text that has been set on it, as well as the font size for that text. Likewise for a **button**, which might combine the text with a background image and some padding.

The same is true for segmented controls, progress bars, and most other controls, although some may only have a predetermined height but an unknown width.

This is known as the intrinsic content size, and it is an important concept in Auto Layout. Auto Layout asks your controls how big they need to be and lays out the screen based on that information.

Usually you want to use the `intrinsic content size`, but there are some cases where you may not want to do that. You can prevent this by setting an explicit Width or Height constraint on a control.

Imagine what happens when you set an image on a UIImageView if that image is much larger than the screen. You usually want to give image views a fixed width and height and scale the content, unless you want the view to resize to the dimensions of the image.

Reference: https://www.raywenderlich.com/115444/auto-layout-tutorial-in-ios-9-part-2-constraints

# Chapter 154: iOS Google Places API

## Section 154.1: Getting Nearby Places from Current Location

Prerequisites

1. Install pods in your project
2. Install the GooglePlaces SDK
3. Enable location services

First we need to get the users location by getting their current longitude and latitude.

1. Import GooglePlaces and GooglePlacePicker

```
import GooglePlaces
import GooglePlacePicker
```

2. Add the `CLLOcationManagerDelegate` protocol

```
class ViewController: UIViewController, CLLocationManagerDelegate {

}
```

3. create your CLLocationManager()

```
var currentLocation = CLLocationManager()
```

4. Request authorization

```
currentLocation = CLLocationManager()
currentLocation.requetAlwayAuthorization()
```

5. Create a button to call the GooglePlacePicker method

@IBAction func placePickerAction(sender: AnyObject) {

```
if CLLOcationManager.authorizationStatues() == .AuthorizedAlways {

        let center = CLLocationCoordinate2DMake((currentLocation.location?.coordinate.latitude)!,
(currentLocation.location?.coordinate.longitude)!)
        let northEast = CLLocationCoordinate2DMake(center.latitude + 0.001, center.longitude +
0.001)
        let southWest = CLLocationCoordinate2DMake(center.latitude - 0.001, center.longitude -
0.001)
        let viewport = GMSCoordinateBounds(coordinate: northEast, coordinate: southWest)
        let config = GMSPlacePickerConfig(viewport: viewport)
        placePicker = GMSPlacePicker(config: config)

        placePicker?.pickPlaceWithCallback({ (place: GMSPlace?, error: NSError?) -> Void in
            if let error = error {
                print("Pick Place error: \(error.localizedDescription)")
                return
            }

            if let place = place {
                print("Place name: \(place.name)")
                 print("Address: \(place.formattedAddress)")
```

```
        } else {
            print("Place name: nil")
             print("Address: nil")
        }
    })
    }
}
```

# Chapter 155: Navigation Bar

## Section 155.1: SWIFT Example

```swift
navigationController?.navigationBar.titleTextAttributes = [NSForegroundColorAttributeName:
UIColor.white, NSFontAttributeName:UIFont(name: "HelveticaNeue-CondensedBold", size: 17)!,]
navigationController?.navigationBar.tintColor = .white
navigationController?.navigationBar.barTintColor = .red
navigationController?.navigationBar.isTranslucent = false
navigationController?.navigationBar.barStyle = .black
```

## Section 155.2: Customize default navigation bar appearance

```objc
// Default UINavigationBar appearance throughout the app
[[UINavigationBar appearance] setTitleTextAttributes:@{NSForegroundColorAttributeName: [UIColor
whiteColor],

                                                       NSFontAttributeName : [UIFont
fontWithName:@"HelveticaNeue-CondensedBold" size:17],
                                                       }];

[[UINavigationBar appearance] setTintColor:[UIColor whiteColor]];
[[UINavigationBar appearance] setBarTintColor:[UIColor KNGRed]];
[[UINavigationBar appearance] setTranslucent:NO];
[[UINavigationBar appearance] setBarStyle:UIBarStyleBlack];
[[UIBarButtonItem appearanceWhenContainedIn: [UISearchBar class], nil] setTintColor:[UIColor
KNGGray]];
```

# Chapter 156: App wide operations

## Section 156.1: Get the top most UIViewController

A common approach to get the top most `UIViewController` is to get the RootViewController of your active `UIWindow`. I wrote an extension for this:

```
extension UIApplication {

func topViewController(_ base: UIViewController? =
UIApplication.shared.keyWindow?.rootViewController) -> UIViewController {

    if let nav = base as? UINavigationController {
        return topViewController(nav.visibleViewController)
    }

    if let tab = base as? UITabBarController {
        if let selected = tab.selectedViewController {
            return topViewController(selected)
        }
    }

    if let presented = base?.presentedViewController {
        return topViewController(presented)
    }

    return base!
}
```

## Section 156.2: Intercept System Events

Using the NotificationCenter of iOS, which can be very powerful, you are able to intercept certain app-wide events:

```
NotificationCenter.default.addObserver(
        self,
        selector: #selector(ViewController.do(_:)),
        name: NSNotification.Name.UIApplicationDidBecomeActive,
        object: nil)
```

You can register for a lot of more events, just take a look at
https://developer.apple.com/reference/foundation/nsnotification.name.

# Chapter 157: CoreImage Filters

## Section 157.1: Core Image Filter Example

**Objective-C**

Just log this see how to use a particular filter

```
NSArray *properties = [CIFilter filterNamesInCategory:kCICategoryBuiltIn];

for (NSString *filterName in properties)
{
CIFilter *fltr = [CIFilter filterWithName:filterName];
NSLog(@"%@", [fltr attributes]);
}
```

In case of CISepiaTone the system log is as follows

```
CIAttributeFilterDisplayName = "Sepia Tone";
CIAttributeFilterName = CISepiaTone;
    CIAttributeReferenceDocumentation =
"http://developer.apple.com/cgi-bin/apple_ref.cgi?apple_ref=//apple_ref/doc/filter/ci/CISepiaTone";
    inputImage =     {
        CIAttributeClass = CIImage;
        CIAttributeDescription = "The image to use as an input image. For filters that also use a
background image, this is the foreground image.";
        CIAttributeDisplayName = Image;
        CIAttributeType = CIAttributeTypeImage;
    };
    inputIntensity =     {
        CIAttributeClass = NSNumber;
        CIAttributeDefault = 1;
        CIAttributeDescription = "The intensity of the sepia effect. A value of 1.0 creates a
monochrome sepia image. A value of 0.0 has no effect on the image.";
        CIAttributeDisplayName = Intensity;
        CIAttributeIdentity = 0;
        CIAttributeMin = 0;
        CIAttributeSliderMax = 1;
        CIAttributeSliderMin = 0;
        CIAttributeType = CIAttributeTypeScalar;
    };
}
```

Using the above system log we set up the filter as below:

```
    CIImage *beginImage = [CIImage imageWithCGImage:[myImageView.image CGImage]];
    CIContext *context = [CIContext contextWithOptions:nil];
    //select Filter Name and Intensity
    CIFilter *filter = [CIFilter filterWithName:@"CISepiaTone" keysAndValues: kCIInputImageKey,
beginImage, @"inputIntensity", [NSNumber numberWithFloat:0.8], nil];
    CIImage *outputImage = [filter outputImage];

    CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
    UIImage *newImg = [UIImage imageWithCGImage:cgimg];

    [myImageView1 setImage:newImg];

    CGImageRelease(cgimg);
```

Image generated by the above code



An alternative way to set up a filter

```objc
UIImageView *imageView1=[[UIImageView alloc]initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height/2)];
UIImageView *imageView2=[[UIImageView alloc]initWithFrame:CGRectMake(0,
self.view.frame.size.height/2, self.view.frame.size.width, self.view.frame.size.height/2)];
imageView1.image=[UIImage imageNamed:@"image.png"];

CIImage *beginImage = [CIImage imageWithCGImage:[imageView1.image CGImage]];
CIContext *context = [CIContext contextWithOptions:nil];
//select Filter Name and Intensity

CIFilter *filter = [CIFilter filterWithName:@"CIColorPosterize"];
[filter setValue:beginImage forKey:kCIInputImageKey];
[filter setValue:[NSNumber numberWithFloat:8.0] forKey:@"inputLevels"];
CIImage *outputImage = [filter outputImage];

CGImageRef cgimg = [context createCGImage:outputImage fromRect:[outputImage extent]];
UIImage *newImg = [UIImage imageWithCGImage:cgimg];

[imageView2 setImage:newImg];

CGImageRelease(cgimg);
[self.view addSubview:imageView1];
```

```
[self.view addSubview:imageView2];
```

Image generated from this code
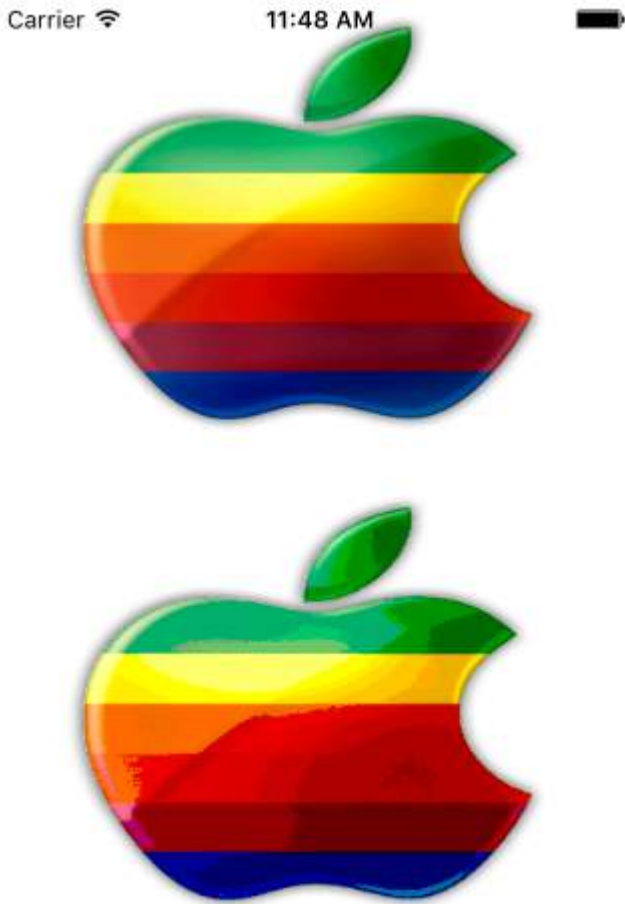


All Available Filters are as below

```
/* CIAccordionFoldTransition,
    CIAdditionCompositing,
    CIAffineClamp,
    CIAffineTile,
    CIAffineTransform,
    CIAreaAverage,
    CIAreaHistogram,
    CIAreaMaximum,
    CIAreaMaximumAlpha,
    CIAreaMinimum,
    CIAreaMinimumAlpha,
    CIAztecCodeGenerator,
    CIBarsSwipeTransition,
    CIBlendWithAlphaMask,
    CIBlendWithMask,
    CIBloom,
    CIBoxBlur,
    CIBumpDistortion,
    CIBumpDistortionLinear,
    CICheckerboardGenerator,
    CICircleSplashDistortion,
    CICircularScreen,
    CICircularWrap,
    CICMYKHalftone,
    CICode128BarcodeGenerator,
```

```
CIColorBlendMode,
CIColorBurnBlendMode,
CIColorClamp,
CIColorControls,
CIColorCrossPolynomial,
CIColorCube,
CIColorCubeWithColorSpace,
CIColorDodgeBlendMode,
CIColorInvert,
CIColorMap,
CIColorMatrix,
CIColorMonochrome,
CIColorPolynomial,
CIColorPosterize,
CIColumnAverage,
CIComicEffect,
CIConstantColorGenerator,
CIConvolution3X3,
CIConvolution5X5,
CIConvolution7X7,
CIConvolution9Horizontal,
CIConvolution9Vertical,
CICopyMachineTransition,
CICrop,
CICrystallize,
CIDarkenBlendMode,
CIDepthOfField,
CIDifferenceBlendMode,
CIDiscBlur,
CIDisintegrateWithMaskTransition,
CIDisplacementDistortion,
CIDissolveTransition,
CIDivideBlendMode,
CIDotScreen,
CIDroste,
CIEdges,
CIEdgeWork,
CIEightfoldReflectedTile,
CIExclusionBlendMode,
CIExposureAdjust,
CIFalseColor,
CIFlashTransition,
CIFourfoldReflectedTile,
CIFourfoldRotatedTile,
CIFourfoldTranslatedTile,
CIGammaAdjust,
CIGaussianBlur,
CIGaussianGradient,
CIGlassDistortion,
CIGlassLozenge,
CIGlideReflectedTile,
CIGloom,
CIHardLightBlendMode,
CIHatchedScreen,
CIHeightFieldFromMask,
CIHexagonalPixellate,
CIHighlightShadowAdjust,
CIHistogramDisplayFilter,
CIHoleDistortion,
CIHueAdjust,
CIHueBlendMode,
CIKaleidoscope,
```

```
CILanczosScaleTransform,
CILenticularHaloGenerator,
CILightenBlendMode,
CILightTunnel,
CILinearBurnBlendMode,
CILinearDodgeBlendMode,
CILinearGradient,
CILinearToSRGBToneCurve,
CILineOverlay,
CILineScreen,
CILuminosityBlendMode,
CIMaskedVariableBlur,
CIMaskToAlpha,
CIMaximumComponent,
CIMaximumCompositing,
CIMedianFilter,
CIMinimumComponent,
CIMinimumCompositing,
CIModTransition,
CIMotionBlur,
CIMultiplyBlendMode,
CIMultiplyCompositing,
CINoiseReduction,
CIOpTile,
CIOverlayBlendMode,
CIPageCurlTransition,
CIPageCurlWithShadowTransition,
CIParallelogramTile,
CIPDF417BarcodeGenerator,
CIPerspectiveCorrection,
CIPerspectiveTile,
CIPerspectiveTransform,
CIPerspectiveTransformWithExtent,
CIPhotoEffectChrome,
CIPhotoEffectFade,
CIPhotoEffectInstant,
CIPhotoEffectMono,
CIPhotoEffectNoir,
CIPhotoEffectProcess,
CIPhotoEffectTonal,
CIPhotoEffectTransfer,
CIPinchDistortion,
CIPinLightBlendMode,
CIPixellate,
CIPointillize,
CIQRCodeGenerator,
CIRadialGradient,
CIRandomGenerator,
CIRippleTransition,
CIRowAverage,
CISaturationBlendMode,
CIScreenBlendMode,
CISepiaTone,
CIShadedMaterial,
CISharpenLuminance,
CISixfoldReflectedTile,
CISixfoldRotatedTile,
CISmoothLinearGradient,
CISoftLightBlendMode,
CISourceAtopCompositing,
CISourceInCompositing,
CISourceOutCompositing,
```

```
        CISourceOverCompositing,
        CISpotColor,
        CISpotLight,
        CISRGBToneCurveToLinear,
        CIStarShineGenerator,
        CIStraightenFilter,
        CIStretchCrop,
        CIStripesGenerator,
        CISubtractBlendMode,
        CISunbeamsGenerator,
        CISwipeTransition,
        CITemperatureAndTint,
        CIToneCurve,
        CITorusLensDistortion,
        CITriangleKaleidoscope,
        CITriangleTile,
        CITwelvefoldReflectedTile,
        CITwirlDistortion,
        CIUnsharpMask,
        CIVibrance,
        CIVignette,
        CIVignetteEffect,
        CIVortexDistortion,
        CIWhitePointAdjust,
        CIZoomBlur*/
```

# Chapter 158: Face Detection Using CoreImage/OpenCV

## Section 158.1: Face and Feature Detection

**Objective-C**

Import the following to your ViewController

```
#import <CoreImage/CoreImage.h>
#import <CoreImage/CoreImage.h>
#import <QuartzCore/QuartzCore.h>
```

Call the function

```
[self faceDetector];
```

Function definition:

```
-(void)faceDetector
{
    // Load the picture for face detection
    UIImageView* image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"download.jpeg"]];

    // Draw the face detection image
    [self.view addSubview:image];

    // Execute the method used to markFaces in background
    [self performSelectorInBackground:@selector(markFaces:) withObject:image];

    // flip image on y-axis to match coordinate system used by core image
    [image setTransform:CGAffineTransformMakeScale(1, -1)];

    // flip the entire window to make everything right side up
    [self.view setTransform:CGAffineTransformMakeScale(1, -1)];


}
```

Mark Face Function

```
//Adds face squares and color masks to eyes and mouth
-(void)markFaces:(UIImageView *)facePicture
{
    // draw a CI image with the previously loaded face detection picture
    CIImage* image = [CIImage imageWithCGImage:facePicture.image.CGImage];

    // create a face detector - since speed is not an issue we'll use a high accuracy
    // detector
    CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                              context:nil options:[NSDictionary
dictionaryWithObject:CIDetectorAccuracyHigh forKey:CIDetectorAccuracy]];

    // create an array containing all the detected faces from the detector
    NSArray* features = [detector featuresInImage:image];
    NSLog(@"Number of faces %d",[features count]);
```

```objc
    // we'll iterate through every detected face.  CIFaceFeature provides us
    // with the width for the entire face, and the coordinates of each eye
    // and the mouth if detected.  Also provided are BOOL's for the eye's and
    // mouth so we can check if they already exist.
    //    for (features in image)
    //    {
    for(CIFaceFeature* faceFeature in features)
    {
        // get the width of the face
        CGFloat faceWidth = faceFeature.bounds.size.width;

        // create a UIView using the bounds of the face
        UIView* faceView = [[UIView alloc] initWithFrame:faceFeature.bounds];

        // add a border around the newly created UIView
        faceView.layer.borderWidth = 1;
        faceView.layer.borderColor = [[UIColor redColor] CGColor];

        // add the new view to create a box around the face
        [self.view addSubview:faceView];

        if(faceFeature.hasLeftEyePosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* leftEyeView = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.leftEyePosition.x-faceWidth*0.15,
faceFeature.leftEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // change the background color of the eye view
            [leftEyeView setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // set the position of the leftEyeView based on the face
            [leftEyeView setCenter:faceFeature.leftEyePosition];
            // round the corners
            leftEyeView.layer.cornerRadius = faceWidth*0.15;
            // add the view to the window
            [self.view addSubview:leftEyeView];
        }

        if(faceFeature.hasRightEyePosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* leftEye = [[UIView alloc]
initWithFrame:CGRectMake(faceFeature.rightEyePosition.x-faceWidth*0.15,
faceFeature.rightEyePosition.y-faceWidth*0.15, faceWidth*0.3, faceWidth*0.3)];
            // change the background color of the eye view
            [leftEye setBackgroundColor:[[UIColor blueColor] colorWithAlphaComponent:0.3]];
            // set the position of the rightEyeView based on the face
            [leftEye setCenter:faceFeature.rightEyePosition];
            // round the corners
            leftEye.layer.cornerRadius = faceWidth*0.15;
            // add the new view to the window
            [self.view addSubview:leftEye];
        }

        if(faceFeature.hasMouthPosition)
        {
            // create a UIView with a size based on the width of the face
            UIView* mouth = [[UIView alloc] initWithFrame:CGRectMake(faceFeature.mouthPosition.x-
faceWidth*0.2, faceFeature.mouthPosition.y-faceWidth*0.2, faceWidth*0.4, faceWidth*0.4)];
            // change the background color for the mouth to green
            [mouth setBackgroundColor:[[UIColor greenColor] colorWithAlphaComponent:0.3]];
            // set the position of the mouthView based on the face
            [mouth setCenter:faceFeature.mouthPosition];
```

```
            // round the corners
            mouth.layer.cornerRadius = faceWidth*0.2;
            // add the new view to the window
            [self.view addSubview:mouth];
        }
    }

    // }
}
```

The Simulator ScreenShot for the Function

# Chapter 159: MPMediaPickerDelegate

## Section 159.1: Load music with MPMediaPickerControllerDelegate and play it with AVAudioPlayer

**Go through the steps:**

- Add 'NSAppleMusicUsageDescription' to your Info.plist for the privacy authority.
- Make sure your music is available in your iPhone. It will not work in the simulator.

Version ≥ iOS 10.0.1

```swift
import UIKit
import AVFoundation
import MediaPlayer

class ViewController: UIViewController, MPMediaPickerControllerDelegate {

    var avMusicPlayer: AVAudioPlayer!
    var mpMediapicker: MPMediaPickerController!
    var mediaItems = [MPMediaItem]()
    let currentIndex = 0

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool){
        //What to do?
    }

    func mediaPicker(_ mediaPicker: MPMediaPickerController, didPickMediaItems mediaItemCollection:
MPMediaItemCollection) {
        mediaItems = mediaItemCollection.items
        updatePlayer()
        self.dismiss(animated: true, completion: nil)
    }

    func updatePlayer(){
        let item = mediaItems[currentIndex]
        // DO-TRY-CATCH try to setup AVAudioPlayer with the path, if successful, sets up the
AVMusicPlayer, and song values.
        if let path: NSURL = item.assetURL as NSURL? {
            do
            {
                avMusicPlayer = try AVAudioPlayer(contentsOf: path as URL)
                avMusicPlayer.enableRate = true
                avMusicPlayer.rate = 1.0
                avMusicPlayer.numberOfLoops = 0
                avMusicPlayer.currentTime = 0
            }
            catch
            {
                avMusicPlayer = nil
            }
        }
    }

    @IBAction func Play(_ sender: AnyObject) {
```

```
        //AVMusicPlayer.deviceCurrentTime
        avMusicPlayer.play()
    }

    @IBAction func Stop(_ sender: AnyObject) {
        avMusicPlayer.stop()
    }

    @IBAction func picker(_ sender: AnyObject) {
        mpMediapicker = MPMediaPickerController.self(mediaTypes:MPMediaType.music)
        mpMediapicker.allowsPickingMultipleItems = false
        mpMediapicker.delegate = self
        self.present(mpMediapicker, animated: true, completion: nil)
    }

}
```

# Chapter 160: Graph (Coreplot)

## Section 160.1: Making graphs with CorePlot

Core Plot provides a podspec, so you can use cocoapods as your library manager which should make installing and updating much simpler

Install cocoapods on your system

In the project directory add a text file to your project called Podfile by typing pod `init` in your project directory

In the Podfile add the line pod 'CorePlot', '~> 1.6'

In the terminal, cd to your project directory and run pod install

Cocoapods will generate a xcworkspace file, which you should use for launching your project (the .xcodeproj file will not include the pod libraries)

Open the .xcworkspace generated by CocoaPods

In the ViewController.h File

```
#import <CorePlot/ios/CorePlot.h>
//#import "CorePlot-CocoaTouch.h" or the above import statement
@interface ViewController : UIViewController<CPTPlotDataSource>
```

In the ViewController.m File

```
-(void)loadView
{
    [super loadView];
    // We need a hostview, you can create one in IB (and create an outlet) or just do this:
    CPTGraphHostingView* hostView = [[CPTGraphHostingView alloc] initWithFrame:CGRectMake(10, 40,
300, 400)];
    hostView.backgroundColor=[UIColor whiteColor];
    self.view.backgroundColor=[UIColor blackColor];
    [self.view addSubview: hostView];
    // Create a CPTGraph object and add to hostView
    CPTGraph* graph = [[CPTXYGraph alloc] initWithFrame:CGRectMake(10, 40, 300, 400)];
    hostView.hostedGraph = graph;
    // Get the (default) plotspace from the graph so we can set its x/y ranges
    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace *) graph.defaultPlotSpace;
    // Note that these CPTPlotRange are defined by START and LENGTH (not START and END) !!
    [plotSpace setYRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( 0 )
length:CPTDecimalFromFloat( 20 )]];
    [plotSpace setXRange: [CPTPlotRange plotRangeWithLocation:CPTDecimalFromFloat( -4 )
length:CPTDecimalFromFloat( 8 )]];
    // Create the plot (we do not define actual x/y values yet, these will be supplied by the
datasource...)
    CPTScatterPlot* plot = [[CPTScatterPlot alloc] initWithFrame:CGRectZero];
    // Let's keep it simple and let this class act as datasource (therefore we implemtn
<CPTPlotDataSource>)
    plot.dataSource = self;
    // Finally, add the created plot to the default plot space of the CPTGraph object we created
before
    [graph addPlot:plot toPlotSpace:graph.defaultPlotSpace];
}
// This method is here because this class also functions as datasource for our graph
```

```
// Therefore this class implements the CPTPlotDataSource protocol
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plotnumberOfRecords
{
        return 9; // Our sample graph contains 9 'points'
}
// This method is here because this class also functions as datasource for our graph
// Therefore this class implements the CPTPlotDataSource protocol
-(NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index
{
    // We need to provide an X or Y (this method will be called for each) value for every index
    int x = index - 4;
    // This method is actually called twice per point in the plot, one for the X and one for the Y
value
    if(fieldEnum == CPTScatterPlotFieldX)
    {
    // Return x value, which will, depending on index, be between -4 to 4
    return [NSNumber numberWithInt: x];
    } else
    {
    // Return y value, for this example we'll be plotting y = x * x
        return [NSNumber numberWithInt: x * x];
    }
}
```

The generated Output is as given below:
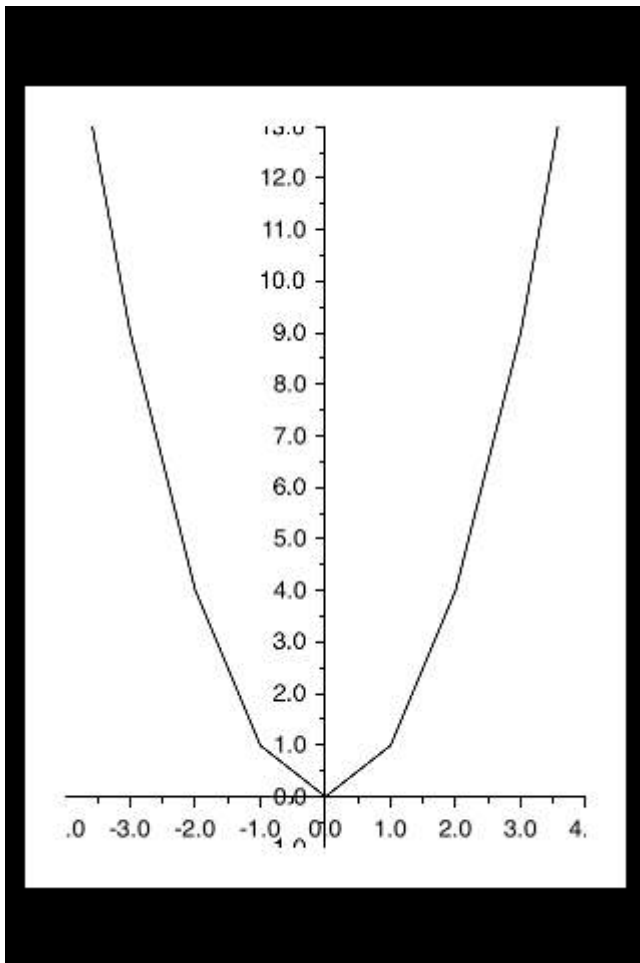
# Chapter 161: FCM Messaging in Swift

## Section 161.1: Initialize FCM in Swift

follow below step to add FCM in your swift Project

1- If you don't have an Xcode project yet, create one now. Create a Podfile if you don't have one:

> $ cd your-project directory
> $ pod init

2- Add the pods that you want to install. You can include a Pod in your Podfile like this:

> pod 'Firebase/Core'
> pod 'Firebase/Messaging'

3- Install the pods and open the .xcworkspace file to see the project in Xcode.

> $ pod install
> $ open your-project.xcworkspace

4- Download a GoogleService-Info.plist file from plist and include it in your app.

5- Upload APNs certificate to Firebase. APN Cert

6- add "import Firebase" in your appDelegate file of project

7- add this "FIRApp.configure()" in your "application:didFinishLaunchingWithOptions"

8- register for remote notification

```
if #available(iOS 10.0, *) {
let authOptions : UNAuthorizationOptions = [.Alert, .Badge, .Sound]
UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(
  authOptions,
  completionHandler: {_,_ in })

// For iOS 10 display notification (sent via APNS)
UNUserNotificationCenter.currentNotificationCenter().delegate = self
// For iOS 10 data message (sent via FCM)
FIRMessaging.messaging().remoteMessageDelegate = self

} else {
  let settings: UIUserNotificationSettings =
  UIUserNotificationSettings(forTypes: [.Alert, .Badge, .Sound], categories: nil)
  application.registerUserNotificationSettings(settings)
}

application.registerForRemoteNotifications()
```

9- to get register token use

```
let token = FIRInstanceID.instanceID().token()!
```

10- and if u want monitor for token change use below code in appDelegate file

```
func tokenRefreshNotification(notification: NSNotification) {
if let refreshedToken = FIRInstanceID.instanceID().token() {
    print("InstanceID token: \(refreshedToken)")
  }

  // Connect to FCM since connection may have failed when attempted before having a token.
  connectToFcm()
}
```

11- to recieve message from fcm add below code in appDelegate

```
func connectToFcm() {
  FIRMessaging.messaging().connectWithCompletion { (error) in
    if (error != nil) {
      print("Unable to connect with FCM. \(error)")
    } else {
      print("Connected to FCM.")
    }
  }
}
```

12- and for disconnect use

```
func applicationDidEnterBackground(application: UIApplication) {
  FIRMessaging.messaging().disconnect()
  print("Disconnected from FCM.")
}
```

in your appDelegate.

the initialization complete and client ready to recieve message from fcm panel or send by token from third party server

# Chapter 162: Create a Custom framework in iOS

## Section 162.1: Create Framework in Swift

follow these step to creating Custom Framework in Swift-IOS:

1. Create a new project. In Xcode
2. Choose iOS/Framework & Library/Cocoa Touch Framework to create a new framework
3. click next and set the productName
4. click next and choose directory to create Project there
5. add code and resources to created project

Framework created successfully

to add created framework to another project, first you should create a workspace
add "target project" and "framework project" to workspace, then :

1. go to the general tab of target project
2. drag the "*.framework" file in product folder of framework project to "Embedded Binaries" section
3. to use in any ViewController or class just import framework at each file

# Chapter 163: Custom Keyboard

## Section 163.1: Custom KeyBoard Example

**Objective-C and Xib**

Add a target to an existing XCode project
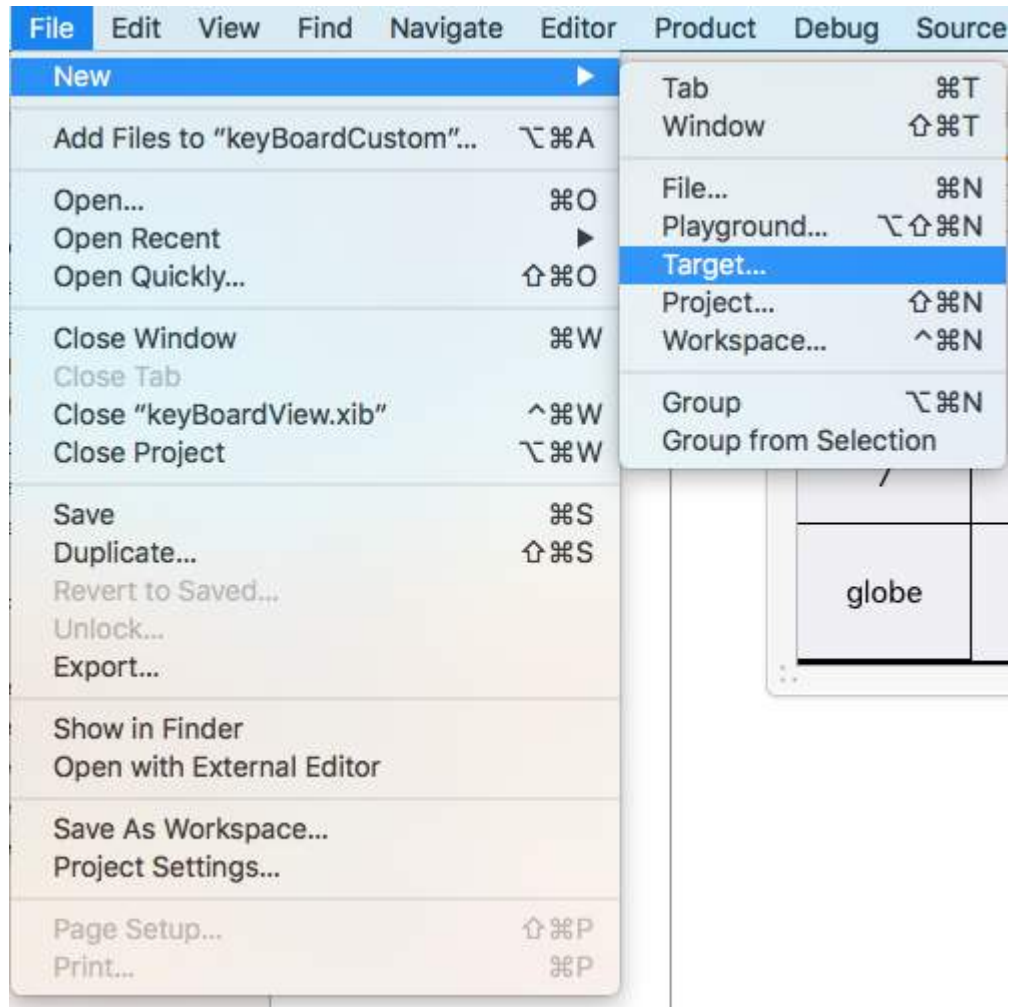


In the Add Target select Custom KeyBoard
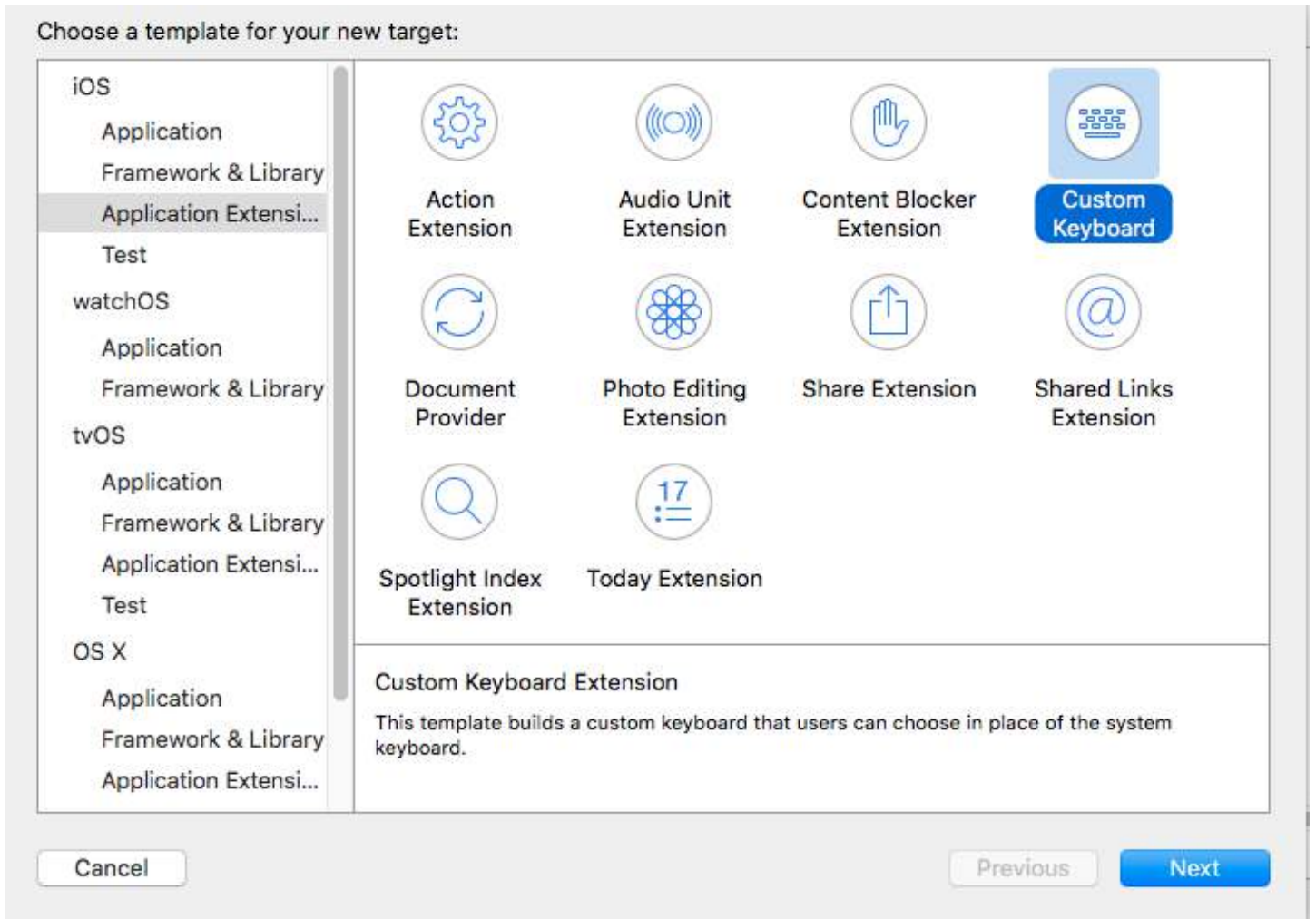
Add the target like this:
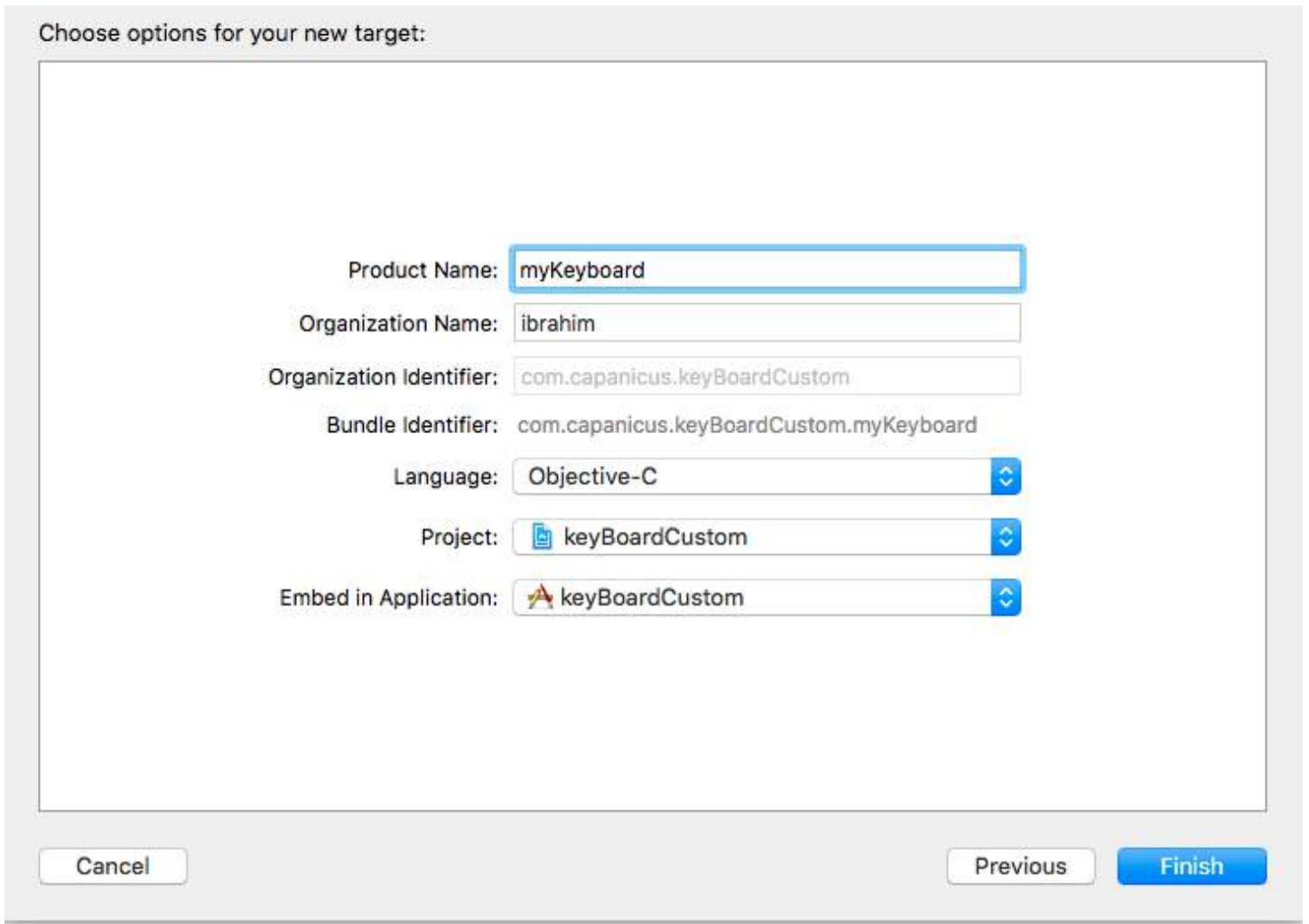
Your project file directory should look something like this
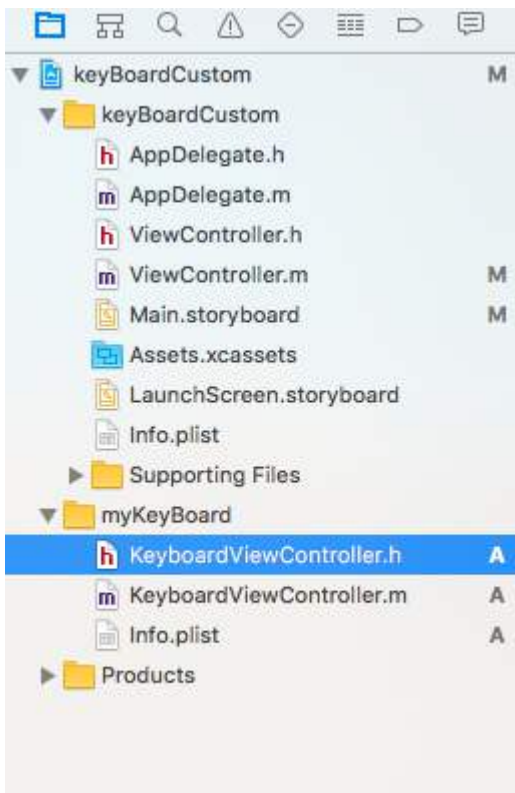


Here myKeyBoard is the name of the added Target

Add new Cocoatouch file of type of type UIView and add an interface file
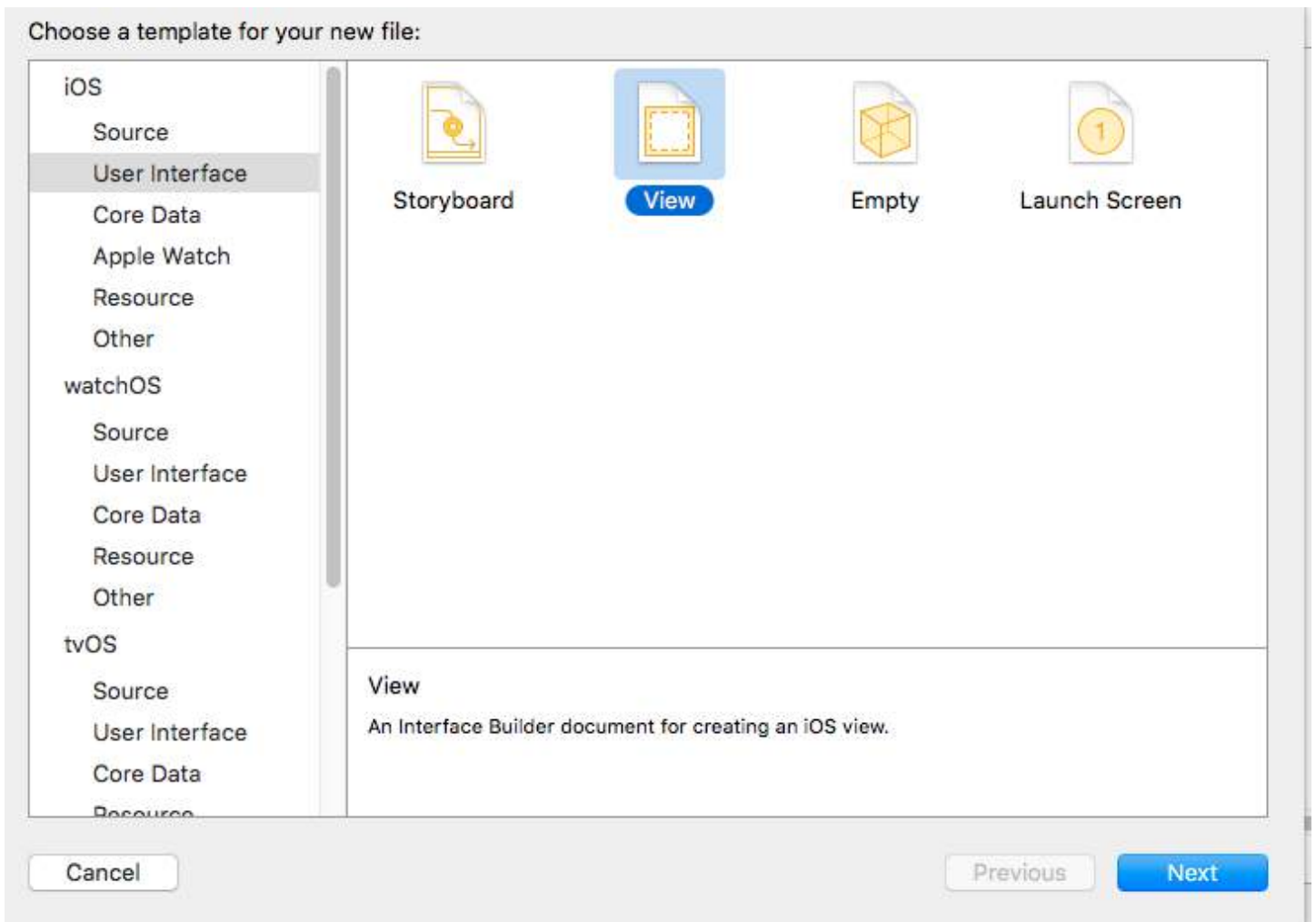
Finally your project directory should look like this
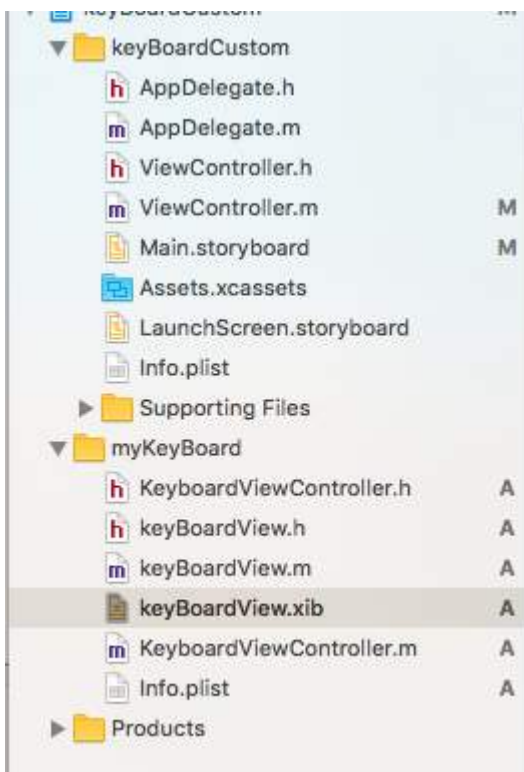


make the `keyBoardView.xib` a subclass of `keyBoardView`
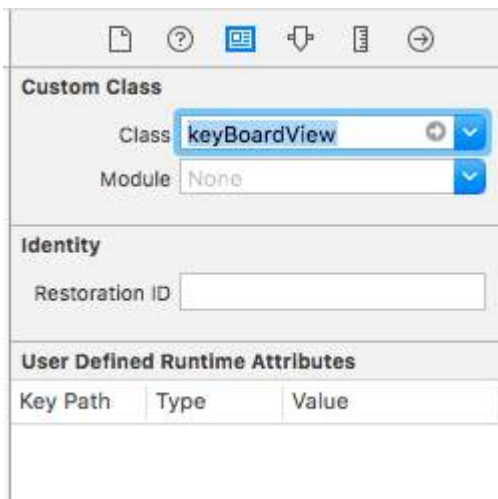
Make interface in the `keyBoardView.xib` file
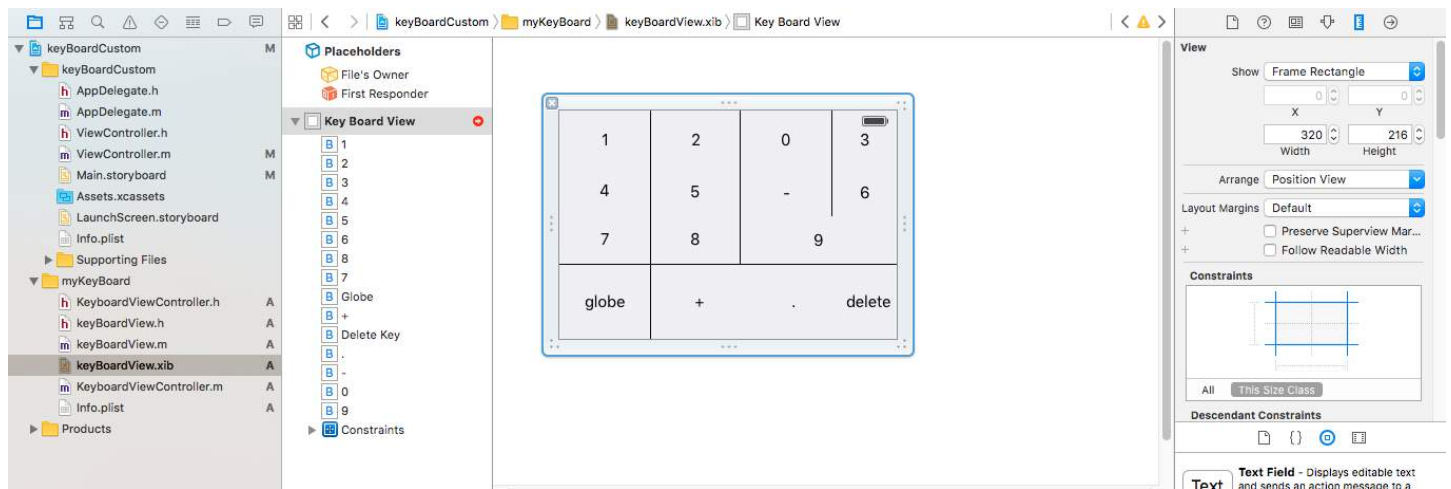


Make connections to from the `keyBoardView.xib` to `keyBoardView.h` file

`keyBoardView.h` should look like

```
#import <UIKit/UIKit.h>

@interface keyBoardView : UIView

@property (weak, nonatomic) IBOutlet UIButton *deleteKey;
//IBOutlet for the delete Key
@property (weak, nonatomic) IBOutlet UIButton *globe;
//Outlet for the key with title globe which changes the keyboard type
@property (strong, nonatomic) IBOutletCollection(UIButton) NSArray *keys;
//Contains a colloection of all the keys '0 to 9' '+' '-' and '.'

@end
```

In the `keyBoardViewController.h` file import `#import "keyBoardView.h"`

Declare a property for keyboard `@property (strong, nonatomic)keyBoardView *keyboard;`

Comment out the

```
@property (nonatomic, strong) UIButton *nextKeyboardButton and all the code associated with it
```

The KeyboardViewController.m file's viewDidLoad() function should look like this

```objc
- (void)viewDidLoad {
    [super viewDidLoad];
    self.keyboard=[[[NSBundle mainBundle]loadNibNamed:@"keyBoardView" owner:nil
options:nil]objectAtIndex:0];
        self.inputView=self.keyboard;
    [self addGestureToKeyboard];


    // Perform custom UI setup here
//    self.nextKeyboardButton = [UIButton buttonWithType:UIButtonTypeSystem];
//
//    [self.nextKeyboardButton setTitle:NSLocalizedString(@"Next Keyboard", @"Title for 'Next
Keyboard' button") forState:UIControlStateNormal];
//    [self.nextKeyboardButton sizeToFit];
//    self.nextKeyboardButton.translatesAutoresizingMaskIntoConstraints = NO;
//
//    [self.nextKeyboardButton addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];
//
//    [self.view addSubview:self.nextKeyboardButton];
//
//    [self.nextKeyboardButton.leftAnchor constraintEqualToAnchor:self.view.leftAnchor].active =
YES;
//    [self.nextKeyboardButton.bottomAnchor constraintEqualToAnchor:self.view.bottomAnchor].active
= YES;
}
```

The functions `addGestureToKeyboard`, `pressDeleteKey`, `keyPressed` are defined below

```objc
-(void) addGestureToKeyboard
{
    [self.keyboard.deleteKey addTarget:self action:@selector(pressDeleteKey)
forControlEvents:UIControlEventTouchUpInside];
    [self.keyboard.globe addTarget:self action:@selector(advanceToNextInputMode)
forControlEvents:UIControlEventTouchUpInside];

    for (UIButton *key in self.keyboard.keys)
    {
        [key addTarget:self action:@selector(keyPressed:)
forControlEvents:UIControlEventTouchUpInside];
    }


}
-(void) pressDeleteKey
{
    [self.textDocumentProxy deleteBackward];
}

-(void)keyPressed:(UIButton *)key
{
    [self.textDocumentProxy insertText:[key currentTitle]];
}
```

Run the Main Application and go to Settings->General->Keyboard->Add New Keyboard-> and add keyboard from the third party keyboard section (The displayed keyboardName would be keyBoardCustom)

The keyboard name can be changed by adding a key called `Bundle display name` and in the Value String Value enter the desired name for the keyboard of the main Project.

---

You can also watch this [Youtube Video](#)

# Chapter 164: AirDrop

## Section 164.1: AirDrop

**Objective-C**

Airdrop can be used from `UIActivityViewController`. The `UIActivityViewController` class is a standard view controller that provides several standard services, such as copying items to the clipboard, sharing content to social media sites, sending items via Messages, AirDrop and some third party applications.

In this case we would be sending an image via `UIActivityViewController`

```objc
UIImage *hatImage = [UIImage imageNamed:@"logo.png"];
if (hatImage)//checks if the image file is not nil
{
//Initialise a UIActivityViewController
UIActivityViewController *controller = [[UIActivityViewController alloc]
initWithActivityItems:@[hatImage] applicationActivities:nil];
//Excludes following options from the UIActivityViewController menu
NSArray *excludeActivities = @[UIActivityTypePostToWeibo,UIActivityTypePrint,
UIActivityTypeMail,UIActivityTypeMessage,UIActivityTypePostToTwitter,UIActivityTypePostToFacebook,

UIActivityTypeCopyToPasteboard,UIActivityTypeAssignToContact,

UIActivityTypeSaveToCameraRoll,UIActivityTypeAddToReadingList,

UIActivityTypePostToFlickr,UIActivityTypePostToVimeo,
                                            UIActivityTypePostToTencentWeibo];
controller.excludedActivityTypes = excludeActivities;
[self presentViewController:controller animated:YES completion:nil];
}
```

**Swift**

```swift
if ((newImage) != nil)
        {
            let activityVC = UIActivityViewController(activityItems: [newImage],
applicationActivities: nil)
            activityVC.excludedActivityTypes =[UIActivityTypeAddToReadingList]
            self.presentViewController(activityVC, animated: true, completion: nil)

        }
```

# Chapter 165: SLComposeViewController

## Section 165.1: SLComposeViewController for Twitter, facebook, SinaWeIbo and TencentWeIbo

**Objective-C**

First add the `Social Framework` to the XCode project.

Import the `#import "Social/Social.h"` class to the required ViewController

**Twitter with text, image and link**

```
//- - To Share text on twitter - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTwitter])
{
        //Tweet
        SLComposeViewController *twitterVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTwitter];
        //To send link together with text
        [twitterVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
        //To add a photo to a link
        [twitterVC addImage:[UIImage imageNamed:@"image"]];
        //Sending link and Image with the tweet
        [twitterVC setInitialText:text];
        /* While adding link and images in a tweet the effective length of a tweet i.e.
        the number of characters which can be entered by the user decreases.
        The default maximum length of a tweet is 140 characters*/
        [self presentViewController:twitterVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
        UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
}
```

**Facebook with Text, Image and Link**

```
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeFacebook])
{
    SLComposeViewController *fbVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeFacebook];
    [fbVC setInitialText:text];
    //To send link together with text
    [fbVC addURL:[NSURL URLWithString:@"https://twitter.com/IbrahimH_ss_n"]];
    //To add a photo to a link
    [fbVC addImage:[UIImage imageNamed:@"image"]];
    [self presentViewController:fbVC animated:YES completion:nil];
}
else
{//Shows alert if twitter is not signed in
        UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to twitter."preferredStyle:UIAlertControllerStyleAlert];
```

```
        [self presentViewController:alertCont animated:YES completion:nil];
        UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
        [alertCont addAction:okay];
}
```

## SinaWeibo

```
//- - SinaWeibo - -
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeSinaWeibo]){

    SLComposeViewController *SinaWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeSinaWeibo];
    [SinaWeiboVC setInitialText:text];

    [self presentViewController:SinaWeiboVC animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```

## TencentWeibo

```
//- -TencentWeibo text share
if([SLComposeViewController isAvailableForServiceType:SLServiceTypeTencentWeibo])
{
    SLComposeViewController *tencentWeiboVC=[SLComposeViewController
composeViewControllerForServiceType:SLServiceTypeTencentWeibo];
    [tencentWeibo setInitialText:text];
    [self presentViewController:tencentWeibo animated:YES completion:nil];
}
else
{
    UIAlertController *alertCont=[UIAlertController alertControllerWithTitle:@"SocialShare"
message:@"You are not signed in to SinaWeibo."preferredStyle:UIAlertControllerStyleAlert];
    [self presentViewController:alertCont animated:YES completion:nil];
    UIAlertAction *okay=[UIAlertAction actionWithTitle:@"Okay" style:UIAlertActionStyleDefault
handler:nil];
    [alertCont addAction:okay];
}
```

# Chapter 166: AirPrint tutorial in iOS

## Section 166.1: AirPrint printing Banner Text

**Objective-C**

Add the delegate and a text-formatter to the `ViewController.h` file

```objc
@interface ViewController : UIViewController <UIPrintInteractionControllerDelegate> {
    UISimpleTextPrintFormatter *_textFormatter;
}
```

In the `ViewController.m` file define the following constants

```objc
#define DefaultFontSize 48
#define PaddingFactor 0.1f
```

The function which prints the text is as follows:

```objc
-(IBAction)print:(id)sender;
{
    /* Get the UIPrintInteractionController, which is a shared object */
    UIPrintInteractionController *controller = [UIPrintInteractionController
sharedPrintController];
    if(!controller){
        NSLog(@"Couldn't get shared UIPrintInteractionController!");
        return;
    }

    /* Set this object as delegate so you can  use the
printInteractionController:cutLengthForPaper: delegate */
    controller.delegate = self;

    UIPrintInfo *printInfo = [UIPrintInfo printInfo];
    printInfo.outputType = UIPrintInfoOutputGeneral;

    /* Use landscape orientation for a banner so the text  print along the long side of the paper.
*/
    printInfo.orientation = UIPrintInfoOrientationLandscape;

    printInfo.jobName = self.textField.text;
    controller.printInfo = printInfo;

    /* Create the UISimpleTextPrintFormatter with the text supplied by the user in the text field
*/
    _textFormatter = [[UISimpleTextPrintFormatter alloc] initWithText:self.textField.text];

    /* Set the text formatter's color and font properties based on what the user chose */
    _textFormatter.color = [self chosenColor];
    _textFormatter.font = [self chosenFontWithSize:DefaultFontSize];

    /* Set this UISimpleTextPrintFormatter on the controller */
    controller.printFormatter = _textFormatter;

    /* Set up a completion handler block.  If the print job has an error before spooling, this is
where it's handled. */
    void (^completionHandler)(UIPrintInteractionController *, BOOL, NSError *) =
^(UIPrintInteractionController *printController, BOOL completed, NSError *error) {
```

```
        if(completed && error)
            NSLog( @"Printing failed due to error in domain %@ with error code %lu. Localized
description: %@, and failure reason: %@", error.domain, (long)error.code,
error.localizedDescription, error.localizedFailureReason );
    };

    if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
        [controller presentFromRect:self.printButton.frame inView:self.view animated:YES
completionHandler:completionHandler];
    else
        [controller presentAnimated:YES completionHandler:completionHandler];  // iPhone
}
```

The delegate function which sets up the print page:

```
- (CGFloat)printInteractionController:(UIPrintInteractionController *)printInteractionController
cutLengthForPaper:(UIPrintPaper *)paper {

    /* Create a font with arbitrary size so that you can calculate the approximate
        font points per screen point for the height of the text. */
    UIFont *font = _textFormatter.font;
    CGSize size = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

    float approximateFontPointPerScreenPoint = font.pointSize / size.height;

    /* Create a new font using a size  that will fill the width of the paper */
    font = [self chosenFontWithSize: paper.printableRect.size.width *
approximateFontPointPerScreenPoint];

    /* Calculate the height and width of the text with the final font size */
    CGSize finalTextSize = [self.textField.text sizeWithAttributes:@{NSFontAttributeName: font}];

    /* Set the UISimpleTextFormatter font to the font with the size calculated */
    _textFormatter.font = font;

    /* Calculate the margins of the roll. Roll printers may have unprintable areas
        before and after the cut.  We must add this to our cut length to ensure the
        printable area has enough room for our text. */
    CGFloat lengthOfMargins = paper.paperSize.height - paper.printableRect.size.height;

    /* The cut length is the width of the text, plus margins, plus some padding */
    return finalTextSize.width + lengthOfMargins + paper.printableRect.size.width * PaddingFactor;
}
```

# Chapter 167: Carthage iOS Setup

## Section 167.1: Carthage Installation Mac

**Carthage Setup**

Download the latest version of Carthage from the given link <u>Download Link</u>

Down in the Download section download the **Carthage.pkg** file.

Once the download is complete install it by double clinking on the download pkg file.

To check for successful download run the following command in your Terminal carthage version This should give the version installed like `0.18-19-g743fa0f`

# Chapter 168: Healthkit

## Section 168.1: HealthKit

**Objective-C**

First go the `Target->Capabilities` and enable `HealthKit`. This would setup the info.plist entry.

Make a new `CocoaClass` of type `NSObject` The filename I gave is `GSHealthKitManager` and the header file is as shown below

**GSHealthKitManager.h**

```objectivec
#import <Foundation/Foundation.h>
#import <HealthKit/HealthKit.h>
@interface GSHealthKitManager : NSObject

+ (GSHealthKitManager *)sharedManager;

- (void)requestAuthorization;

- (NSDate *)readBirthDate;
- (void)writeWeightSample:(double)weight;
- (NSString *)readGender;

@end
```

**GSHealthKitManager.m**

```objectivec
#import "GSHealthKitManager.h"
#import <HealthKit/HealthKit.h>


@interface GSHealthKitManager ()

@property (nonatomic, retain) HKHealthStore *healthStore;

@end


@implementation GSHealthKitManager

+ (GSHealthKitManager *)sharedManager {
    static dispatch_once_t pred = 0;
    static GSHealthKitManager *instance = nil;
    dispatch_once(&pred, ^{
        instance = [[GSHealthKitManager alloc] init];
        instance.healthStore = [[HKHealthStore alloc] init];
    });
    return instance;
}

- (void)requestAuthorization {

    if ([HKHealthStore isHealthDataAvailable] == NO) {
        // If our device doesn't support HealthKit -> return.
        return;
    }
```

```
    NSArray *readTypes = @[[HKObjectType
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierDateOfBirth],[HKObjectType
characteristicTypeForIdentifier:HKCharacteristicTypeIdentifierBiologicalSex]];


    [self.healthStore requestAuthorizationToShareTypes:nil readTypes:[NSSet setWithArray:readTypes]
completion:nil];
}

- (NSDate *)readBirthDate {
    NSError *error;
    NSDate *dateOfBirth = [self.healthStore dateOfBirthWithError:&error];   // Convenience method
of HKHealthStore to get date of birth directly.

    if (!dateOfBirth) {
        NSLog(@"Either an error occurred fetching the user's age information or none has been
stored yet. In your app, try to handle this gracefully.");
    }

    return dateOfBirth;
}

- (NSString *)readGender
{
    NSError *error;
    HKBiologicalSexObject *gen=[self.healthStore biologicalSexWithError:&error];
    if (gen.biologicalSex==HKBiologicalSexMale)
    {
        return(@"Male");
    }
    else if (gen.biologicalSex==HKBiologicalSexFemale)
    {
        return (@"Female");
    }
    else if (gen.biologicalSex==HKBiologicalSexOther)
    {
        return (@"Other");
    }
    else{
        return (@"Not Set");
    }
}



@end
```

Calling from ViewController

```
- (IBAction)pressed:(id)sender {

    [[GSHealthKitManager sharedManager] requestAuthorization];
    NSDate *birthDate = [[GSHealthKitManager sharedManager] readBirthDate];
        NSLog(@"birthdate %@", birthDate);
        NSLog(@"gender 2131321 %@", [[GSHealthKitManager sharedManager] readGender]);


}
```

**Log Output**

```
2016-10-13 14:41:39.568 random[778:26371] birthdate 1992-11-29 18:30:00 +0000
2016-10-13 14:41:39.570 random[778:26371] gender 2131321 Male
```

# Chapter 169: Core SpotLight in iOS

## Section 169.1: Core-Spotlight

**Objective-C**

1. Create a new iOS project and add *CoreSpotlight* and *MobileCoreServices* framework to your project.
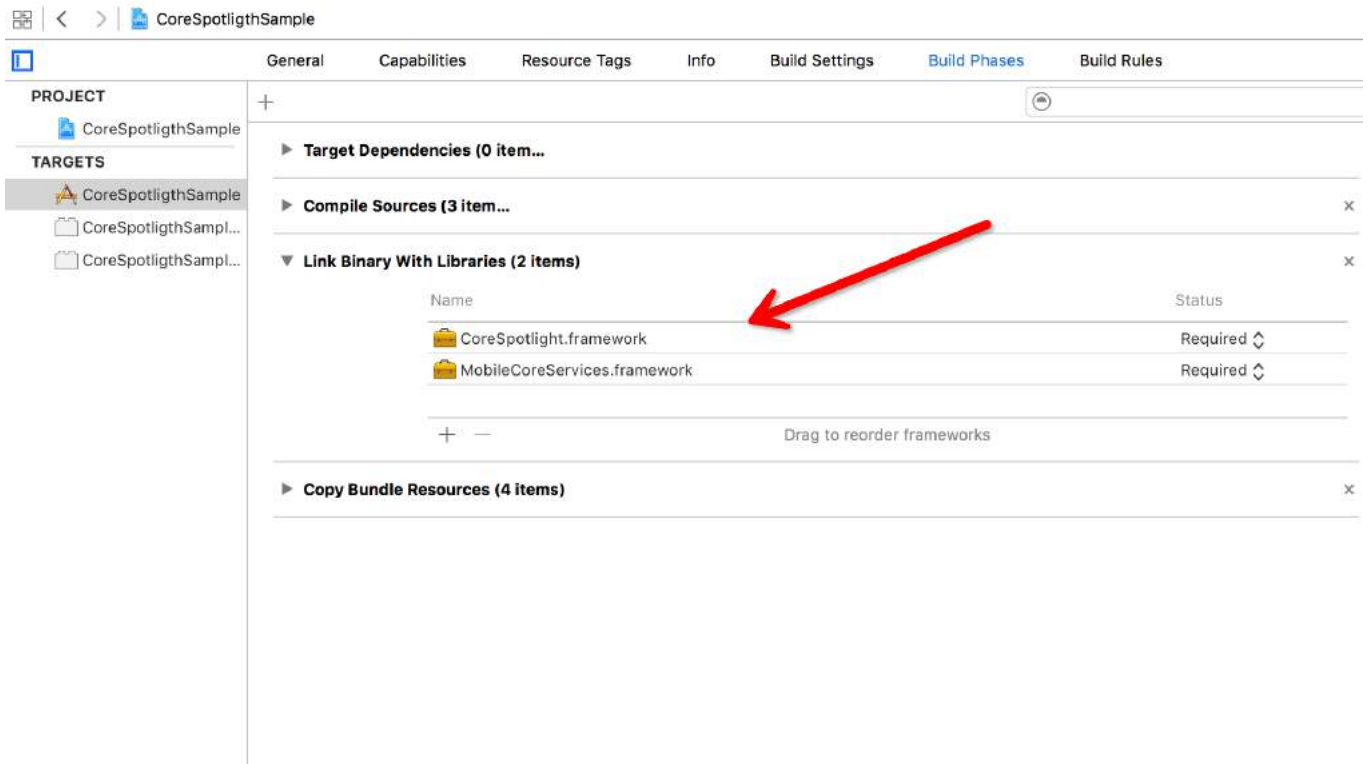


2. Create the actual CSSearchableItem and associating the uniqueIdentifier, domainIdentifier and the attributeSet. Finally index the CSSearchableItem using [[CSSearchableIndex defaultSearchableIndex]...] as show below.

```
 4  //
 5  //  Created by Mayqiyue on 7/10/15.
 6  //  Copyright © 2015 mayqiyue. All rights reserved.
 7  //
 8
 9  #import "ViewController.h"
10  #import <CoreSpotlight/CoreSpotlight.h>
11  #import <MobileCoreServices/MobileCoreServices.h>
12
13  @interface ViewController ()
14
15  @end
16
17  @implementation ViewController
18
19  - (void)viewDidLoad {
20      [super viewDidLoad];
21      [self setupCoreSpotlightSearch];
22  }
23
24  - (void)setupCoreSpotlightSearch {
25      CSSearchableItemAttributeSet *attributeSet = [[CSSearchableItemAttributeSet alloc] initWithItemContentType:
              (NSString *)kUTTypeImage];
26      attributeSet.title = @"My First Spotlight Search";
27      attributeSet.contentDescription = @"";
28      attributeSet.keywords = [NSArray arrayWithObjects:@"Hello", @"Welcome", @"Spotlight", nil];
29      UIImage *image = [UIImage imageNamed:@"l"];
30      NSData *imageData = [NSData dataWithData:UIImagePNGRepresentation(image)];
31      attributeSet.thumbnailData = imageData;
32
33      CSSearchableItem *item = [[CSSearchableItem alloc]initWithUniqueIdentifier:@"com.deeplink"
              domainIdentifier:@"spotlight.sample" attributeSet:attributeSet];
34
35      [[CSSearchableIndex defaultSearchableIndex] indexSearchableItems:@[item] completionHandler: ^(NSError * __nullable
              error) {
36          if (!error)
37              NSLog(@"Search item indexed");
38      }];
39  }
40
41  - (void)didReceiveMemoryWarning {
42      [super didReceiveMemoryWarning];
43  }
```

3. OK!Test the index!
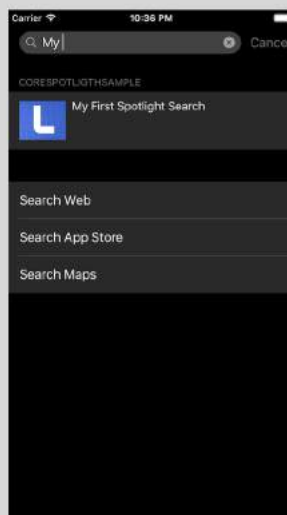
# Chapter 170: Core Motion

## Section 170.1: Accessing Barometer to get relative altitude

**Swift**

Import the Core Motion library:

```
import CoreMotion
```

Next, we need to create a `CMAltimeter` object, but a common pitfall is to create it in the `viewDidLoad()`. If done that way, the altimeter won't be accessible when we need to call a method on it. Nevertheless, go ahead and create your `CMAltimeter` object just before the `viewDidLoad()`:

```
let altimeter = CMAltimeter()
```

**Now:**

1. We need to check if `relativeAltitude` is even available with the following method:
   `CMAltimeter.isRelativeAltitudeAvailable`.

2. If that returns `true`, you can then begin monitoring altitude change with
   `startRelativeAltitudeUpdatesToQueue`

3. If there are no errors, you should be able to retrieve data from the `relativeAltitude` and pressure
   properties.

Given below is the definition of a button action to begin monitoring with our barometer.

```
@IBAction func start(sender: AnyObject){
if CMAltimeter.isRelativeAltitudeAvailable() {
    // 2
    altimeter.startRelativeAltitudeUpdatesToQueue(NSOperationQueue.mainQueue(), withHandler: {
data, error in
        // 3
        if (error == nil) {
            println("Relative Altitude: \(data.relativeAltitude)")
            println("Pressure: \(data.pressure)")
        }
    })
}
}
```

# Chapter 171: QR Code Scanner

QR (Quick Response) codes are two-dimensional barcodes which are widely used on machine-readable optical labels. iOS do provide a way to read the QR codes by using `AVFoundation` framework from iOS 7 onwards. This framework provides set of API's to setup/open the camera and read QR codes from the camera feed.

## Section 171.1: Scanning QR code with AVFoudation framework

Prior to iOS 7 when you want to scan a QR code, we might need to rely on third party frameworks or libraries like zBar or zXing. But Apple introduced `AVCaptureMetaDataOutput` from iOS 7 for reading barcodes.

To read QR code using `AVFoundation` we need to setup/create `AVCaptureSession` and use `captureOutput:didOutputMetadataObjects:fromConnection:` delegate method.

**Step 1**

Import `AVFoundation` framework and confirm to `AVCaptureMetadataOutputObjectsDelegate` protocol

```
import AVFoundation
class ViewController: UIViewController, AVCaptureMetadataOutputObjectsDelegate
```

**Step 2**

QR code reading is totally based on video capture. So to capture continuous video create an `AVCaptureSession` and set up device input and output. Add the below code in view controller `viewDidLoad` method

```
// Create an instance of the AVCaptureDevice and provide the video as the media type parameter.
let captureDevice = AVCaptureDevice.defaultDevice(withMediaType: AVMediaTypeVideo)

do {
    // Create an instance of the AVCaptureDeviceInput class using the device object and intialise
capture session
    let input = try AVCaptureDeviceInput(device: captureDevice)
    captureSession = AVCaptureSession()
    captureSession?.addInput(input)

    // Create a instance of AVCaptureMetadataOutput object and set it as the output device the
capture session.
    let captureMetadataOutput = AVCaptureMetadataOutput()
    captureSession?.addOutput(captureMetadataOutput)
    // Set delegate with a default dispatch queue
    captureMetadataOutput.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
    //set meta data object type as QR code, here we can add more then one type as well
    captureMetadataOutput.metadataObjectTypes = [AVMetadataObjectTypeQRCode]


    // Initialize the video preview layer and add it as a sublayer to the viewcontroller view's
layer.
    videoPreviewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    videoPreviewLayer?.videoGravity = AVLayerVideoGravityResizeAspectFill
    videoPreviewLayer?.frame = view.layer.bounds
    view.layer.addSublayer(videoPreviewLayer!)

    // Start capture session.
    captureSession?.startRunning()
} catch {
    // If any error occurs, let the user know. For the example purpose just print out the error
```

```
    print(error)
    return
}
```

**Step 3**

Implement `AVCaptureMetadataOutputObjectsDelegate` delegate method to read the QR code

```
func captureOutput(_ captureOutput: AVCaptureOutput!, didOutputMetadataObjects metadataObjects:
[Any]!, from connection: AVCaptureConnection!) {

    // Check if the metadataObjects array contains at least one object. If not no QR code is in our
video capture
    if metadataObjects == nil || metadataObjects.count == 0 {
        // NO QR code is being detected.
        return
    }

    // Get the metadata object and cast it to `AVMetadataMachineReadableCodeObject`
    let metadataObj = metadataObjects[0] as! AVMetadataMachineReadableCodeObject

    if metadataObj.type == AVMetadataObjectTypeQRCode {
        // If the found metadata is equal to the QR code metadata then get the string value from
meta data
        let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)

        if metadataObj.stringValue != nil {
            // metadataObj.stringValue is our QR code
        }
    }
}
```

here metadata object can give you the bounds of the QR code read on the camera feed as well. To get the bounds simply pass the metadata object to `videoPreviewLayer`'s `transformedMetadataObject` method like below.

```
let barCodeObject = videoPreviewLayer?.transformedMetadataObject(for: metadataObj)
        qrCodeFrameView?.frame = barCodeObject!.bounds
```

# Section 171.2: UIViewController scanning for QR and displaying video input

```
import AVFoundation
class QRScannerViewController: UIViewController,
    AVCaptureMetadataOutputObjectsDelegate {

    func viewDidLoad() {
        self.initCaptureSession()
    }

    private func initCaptureSession() {
        let captureDevice = AVCaptureDevice
            .defaultDevice(withMediaType: AVMediaTypeVideo)
        do {
            let input = try AVCaptureDeviceInput(device: captureDevice)
            let captureMetadataOutput = AVCaptureMetadataOutput()
            self.captureSession?.addOutput(captureMetadataOutput)
            captureMetadataOutput.setMetadataObjectsDelegate(self,
                queue: DispatchQueue.main)
            captureMetadataOutput
                .metadataObjectTypes = [AVMetadataObjectTypeQRCode]
```

```swift
            self.videoPreviewLayer =
                AVCaptureVideoPreviewLayer(session: self.captureSession)
            self.videoPreviewLayer?
                .videoGravity = AVLayerVideoGravityResizeAspectFill
            self.videoPreviewLayer?.frame =
                self.view.layer.bounds

            self._viewController?.view.layer
                .addSublayer(videoPreviewLayer!)
            self.captureSession?.startRunning()
        } catch {
            //TODO: handle input open error
        }
    }
    private func dismissCaptureSession() {
        if let running = self.captureSession?.isRunning, running {
            self.captureSession?.stopRunning()
        }
        self.captureSession = nil
        self.videoPreviewLayer?.removeFromSuperLayer()
        self.videoPreviewLayer = nil
    }

    func captureOutput(_ captureOutput: AVCaptureOutput,
        didOutputMetadataObjects metadataObjects: [Any]!,
        from connection: AVCaptureConnection) {
        guard metadataObjects != nil && metadataObjects.count != 0 else {
            //Nothing captured
            return
        }

        if let metadataObj =
            metadataObjects[0] as? AVMetadataMachineReadableCodeObject {
            guard metadataObj.type == AVMetadataObjectTypeQRCode else {
                return
            }

            let barCodeObject = videoPreviewLayer?
                .transformedMetadataObject(for:
                    metadataObj as AVMetadataMachineReadableCodeObject)
                 as! AVMetadataMachineReadableCodeObject

            if let qrValue = metadataObj.stringValue {
                self.handleQRRead(value: qrValue)
            }
        }
    }

    private handleQRRead(value: String) {
        //TODO: Handle the read qr
    }
    private captureSession: AVCaptureSession?
    private videoPreviewLayer: AVCaptureVideo
}
```

handleQRRead - will be called on a successful scan `initCaptureSession` - initialize scanning for QR and camera input dismissCaptureSession - hide the camera input and stop scanning

# Chapter 172: plist iOS

Plist is used for storage of data in iOS app. Plist save data in form of Array and Dictionaries. In plist we can save data as: 1. Static data to be used in app. 2. Data that will be coming from server.

## Section 172.1: Example:

**1. Static data to be used in app.**

To save static data in plist follow these methods:

**a) Add a new file**
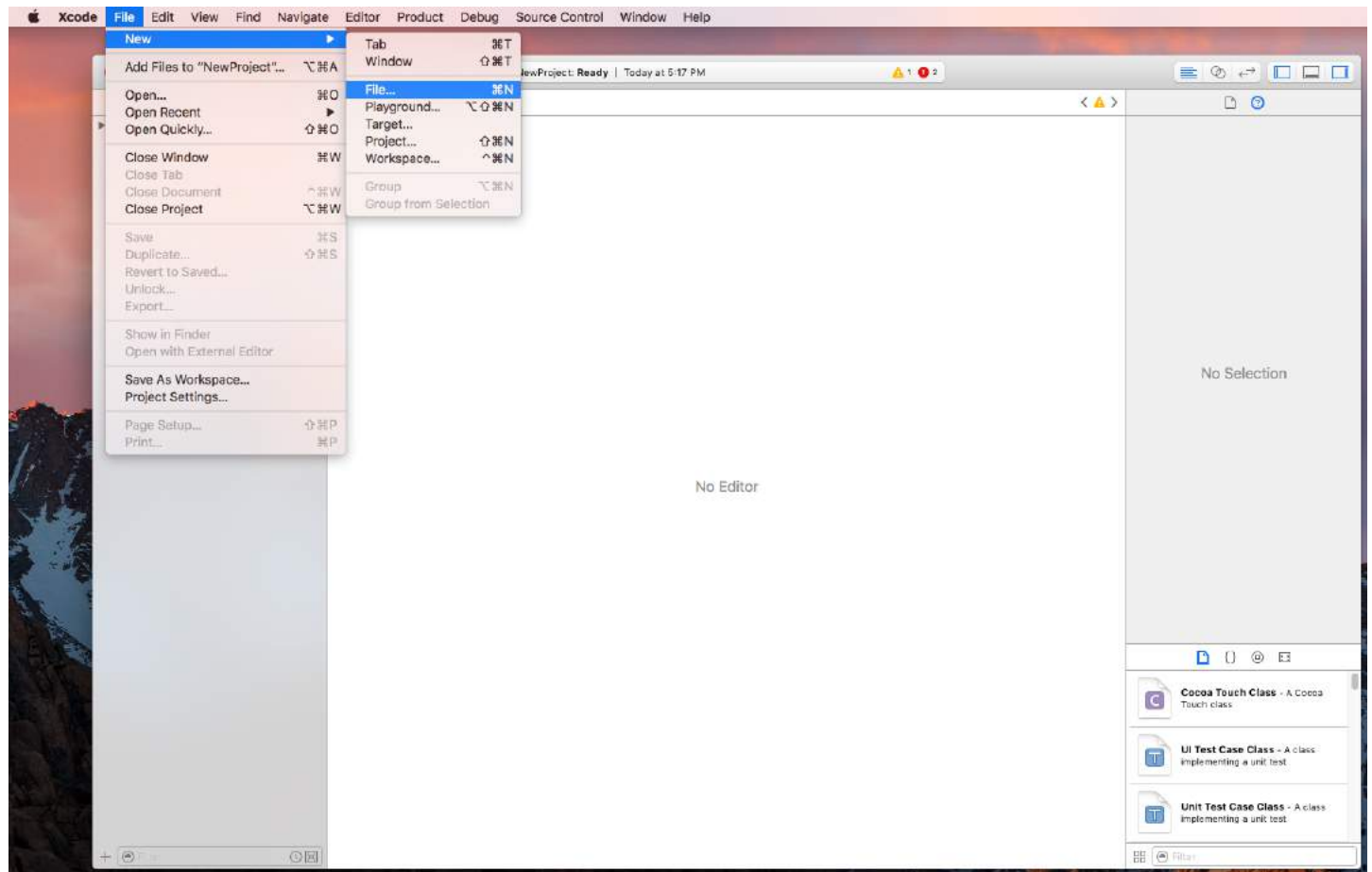


**b) Click Property list in Resources**
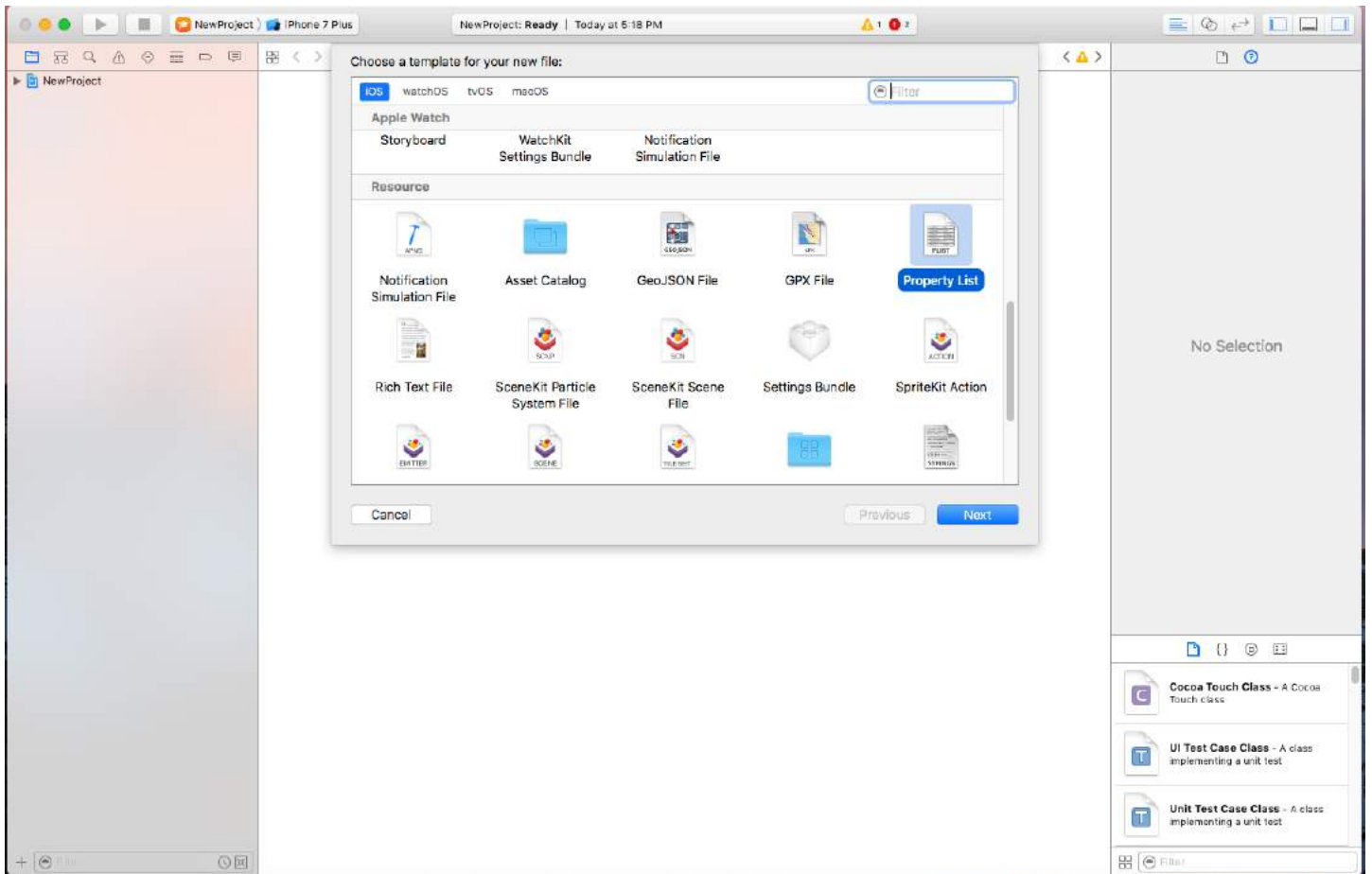
**c) Name the propertylist and a file will be created as(data.plist here)**
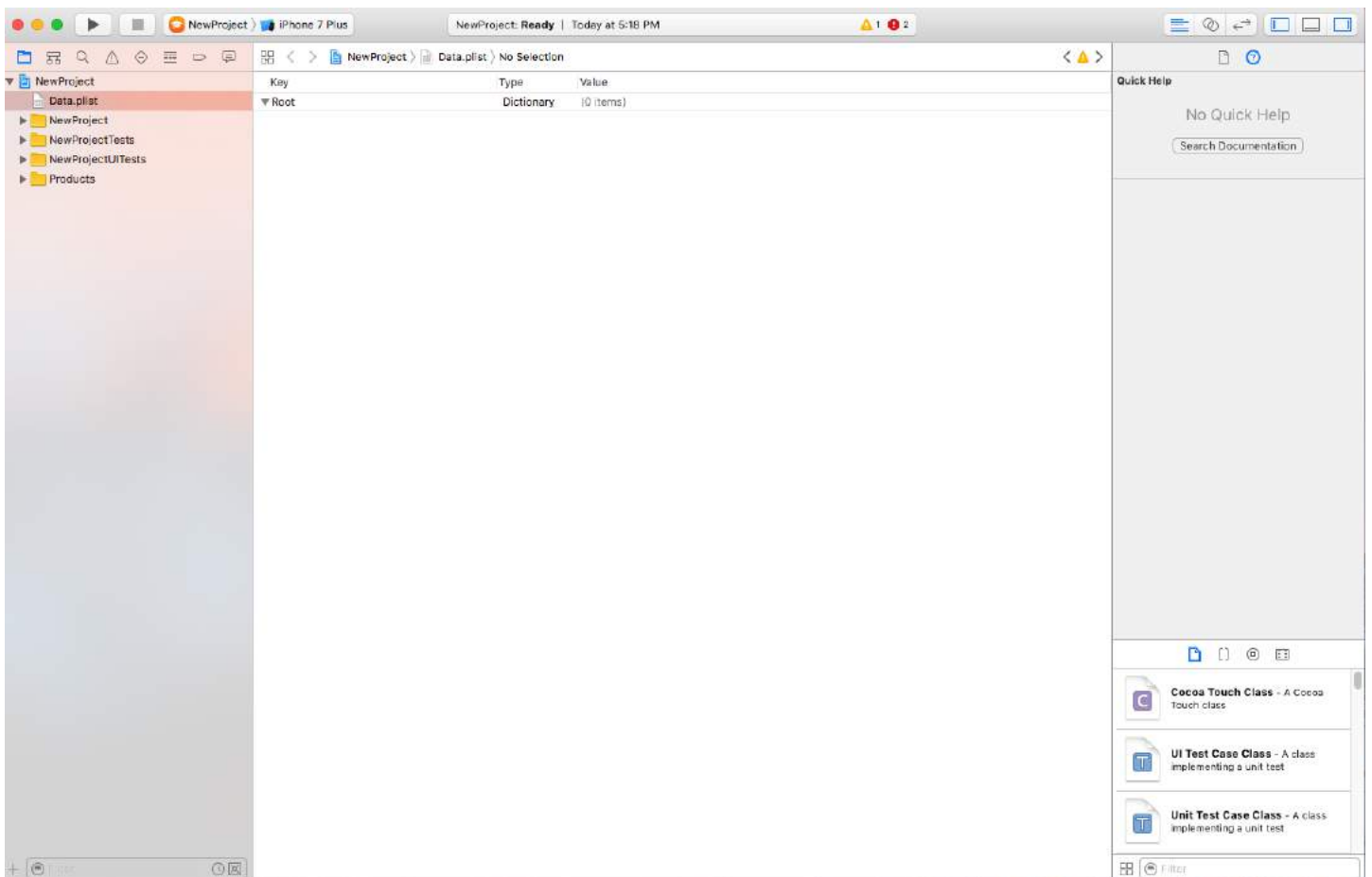


d) You can create a plist of Arrays and Dictionaries as:
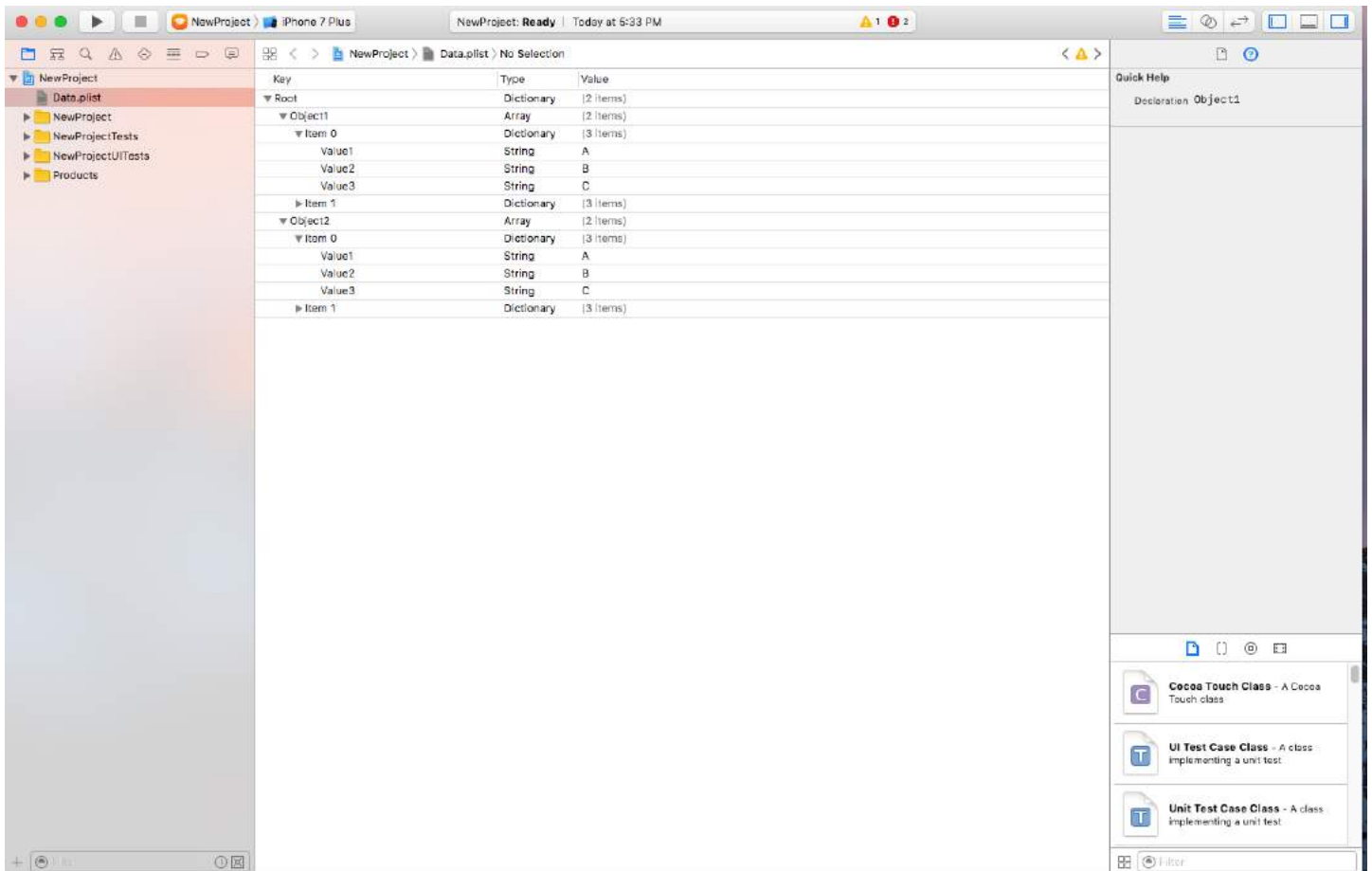
// Read plist from bundle and get Root Dictionary out of it

```objc
NSDictionary *dictRoot = [NSDictionary dictionaryWithContentsOfFile:[[NSBundle mainBundle]
pathForResource:@"Data" ofType:@"plist"]];
```

// Your dictionary contains an array of dictionary // Now pull an Array out of it.

```objc
NSArray *arrayList = [NSArray arrayWithArray:[dictRoot objectForKey:@"Object1"]];

for(int i=0; i< [arrayList count]; i++)
{
    NSMutableDictionary *details=[arrayList objectAtIndex:i];
}
```

# Section 172.2: Save and edit/delete data from Plist

You have already created a plist. This plist will remain same in app. If you want to edit the data in this plist, add new data in plist or remove data from plist, you can't make changes in this file.

For this purpose you will have to store your plist in Document Directory. You can edit your plist saved in document directory.

**Save plist in document directory as:**

```objc
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"Data" ofType:@"plist"];

NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:filePath];

NSDictionary *plistDict = dict;
```

```
NSFileManager *fileManager = [NSFileManager defaultManager];

NSString *error = nil;

NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:plistDict
format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

if (![fileManager fileExistsAtPath: plistPath]) {

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }
}
else
{

}
```

**Retrieve data from Plist as:**

```
    NSArray *paths = NSSearchPathForDirectoriesInDomains (NSDocumentDirectory, NSUserDomainMask,
YES);
    NSString *documentsPath = [paths objectAtIndex:0];
    NSString *plistPath = [documentsPath stringByAppendingPathComponent:@"Data.plist"];
    NSDictionary *dict = [[NSDictionary alloc] initWithContentsOfFile:plistPath];

    NSArray *usersArray = [dict objectForKey:@"Object1"];
```

You can edit remove, add new data as per your requirement and save the plist again to Document Directory.

# Chapter 173: WCSessionDelegate

WCSessionDelegate works with watch OS2 + using WatchConnectivity. var watchSession : WCSession? func
startWatchSession(){ if(WCSession.isSupported()){ watchSession = WCSession.default() watchSession!.delegate = self
watchSession!.activate() } } Implement the required method:- `didReceiveApplicationContext`

## Section 173.1: Watch kit controller (WKInterfaceController)

```
import WatchConnectivity

var watchSession : WCSession?

    override func awake(withContext context: Any?) {
        super.awake(withContext: context)
        // Configure interface objects here.
        startWatchSession()
    }

func startWatchSession(){

    if(WCSession.isSupported()){
        watchSession = WCSession.default()
        watchSession!.delegate = self
        watchSession!.activate()
    }
}

//Callback in below delegate method when iOS app triggers event
func session(_ session: WCSession, didReceiveApplicationContext applicationContext: [String : Any])
{
    print("did ReceiveApplicationContext at watch")
}
```

# Chapter 174: AppDelegate

**AppDelegate** is a protocol which defines methods that are called by the singleton UIApplication object in response to important events in the lifetime of an app.

Normally used to perform tasks on application startup (setup app environment, analitycs (ex.: Mixpanel/GoogleAnalytics/Crashlitics), DB stack etc.) and shutdown (ex.: save DB context), handling URL open requests and similar application-wide tasks.

## Section 174.1: All States of Application through AppDelegate methods

For Getting updated or to do something before app goes live to user you can use below method.

**AppDidFinishLaunching**

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    // Write your code before app launch
    return YES;
}
```

**While App Enter in Foreground:**

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the active state; here you can undo
many of the changes made on entering the background.
}
```

**When App Launching and also background to Foreground hit below method:**

```
- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was inactive.
If the application was previously in the background, optionally refresh the user interface.
}
```

**While App Enter in background:**

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store
enough application state information to restore your application to its current state in case it is
terminated later.
    // If your application supports background execution, this method is called instead of
applicationWillTerminate: when the user quits.
}
```

**While app resign active**

```
- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for
certain types of temporary interruptions (such as an incoming phone call or SMS message) or when
the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and invalidate graphics rendering
callbacks. Games should use this method to pause the game.
}
```

**While app terminate:**

```
- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See also
applicationDidEnterBackground:.
}
```

# Section 174.2: AppDelegate Roles:

- AppDelegate contains your app's `startup code`.
- It responds to `key changes in the state` of your app. Specifically, it responds to both temporary interruptions and to changes in the execution state of your app, such as when your app transitions from the foreground to the background.
- It `responds to notifications` originating from outside the app, such as remote notifications (also known as push notifications), low-memory warnings, download completion notifications, and more.
- It `determines` whether `state preservation` and `restoration` should occur and assists in the preservation and restoration process as needed.
- It `responds to events` that target the app itself and are not specific to your app's views or view controllers. You can use it to store your app's central data objects or any content that does not have an owning view controller.

# Section 174.3: Opening a URL-Specified Resource

Asks the delegate to open a resource specified by a URL, and provides a dictionary of launch options.

*Example of usage:*

```
func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey :
Any] = [:]) -> Bool {
        return SomeManager.shared.handle(
            url,
            sourceApplication: options[.sourceApplication] as? String,
            annotation: options[.annotation]
        )
    }
```

# Section 174.4: Handling Local and Remote Notifications

*Example of usage:*

```
/* Instance of your custom APNs/local notification manager */
private var pushManager: AppleNotificationManager!
```

Registration:

```
func application(application: UIApplication, didRegisterUserNotificationSettings
notificationSettings: UIUserNotificationSettings) {
    // Called to tell the delegate the types of notifications that can be used to get the user's
attention
    pushManager.didRegisterSettings(notificationSettings)
}

func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken
deviceToken: NSData) {
    // Tells the delegate that the app successfully registered with Apple Push Notification service
```

```
(APNs)
    pushManager.didRegisterDeviceToken(deviceToken)
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError
error: NSError) {
    // Sent to the delegate when Apple Push Notification service cannot successfully complete the
registration process.
    pushManager.didFailToRegisterDeviceToken(error)
}
```

Remote notifications handling:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject :
AnyObject]) {
    // Remote notification arrived, there is data to be fetched
    // Handling it
    pushManager.handleNotification(userInfo,
                                   background: application.applicationState == .Background
    )
}
```

Local notifications handling:

```
func application(application: UIApplication, didReceiveLocalNotification notification:
UILocalNotification) {
    pushManager.handleLocalNotification(notification, background: false)
}
```

Handling action (deprecated):

```
func application(application: UIApplication, handleActionWithIdentifier identifier: String?,
forRemoteNotification userInfo: [NSObject : AnyObject],
                 completionHandler: () -> Void) {
    pushManager.handleInteractiveRemoteNotification(userInfo, actionIdentifier: identifier,
completion: completionHandler)
}
```

# Chapter 175: App Submission Process

This tutorial covers all the necessary steps required to upload an iOS app to the App Store.

## Section 175.1: Setup provisioning profiles

In previous versions, setting up provisioning profiles was done manually. You generate distribution profile, download it and then distribute your app. This had to be done for every development machine which was extremely time consuming. However, in most situations nowadays, Xcode 8 will do most of this work for you. Make sure you sign in with the account which will be used for distributing the app and then simply select "Automatically manage code signing" in Targets -> General.



## Section 175.2: Archive the code

Once the provisioning profiles are all set, next step in the process of submitting the app is to archive your code. From the dropdown of devices and simulators select option "Generic iOS device". Then, under "Product" menu select option "Archive".
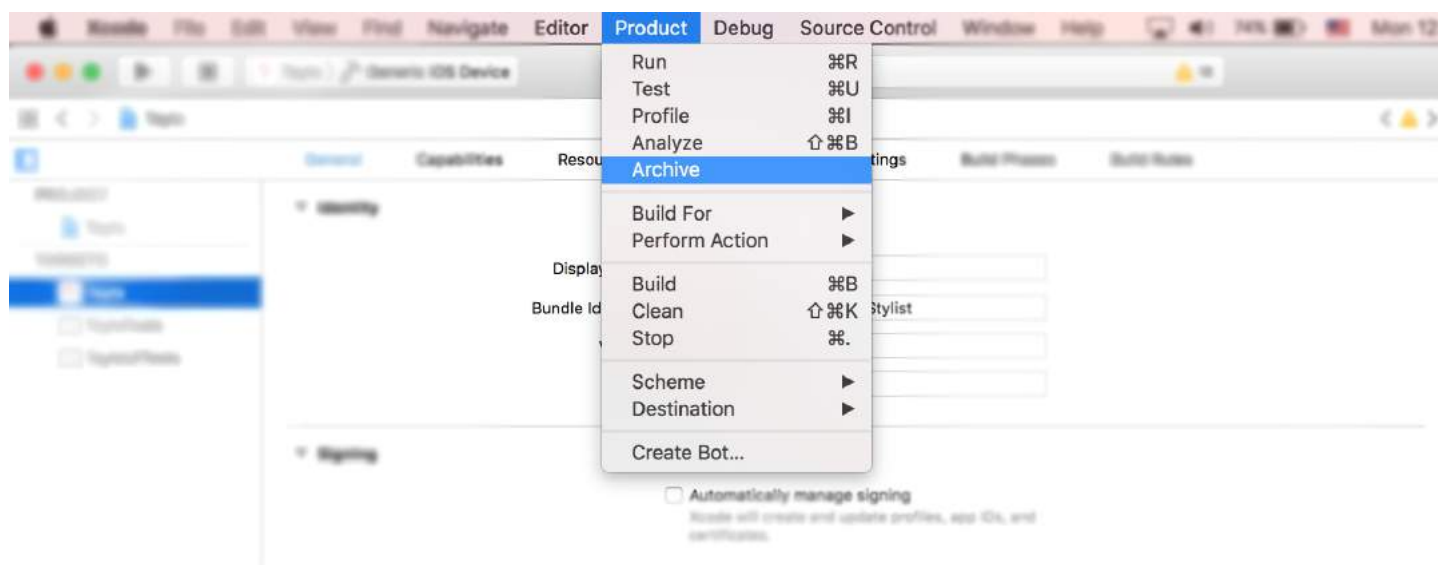


In case where the submission is an update to the existing app on the store, make sure that the build number is higher than the current and that the version number is different. For example, current application has build number 30 and version label 1.0. The next update should have at least build number 31 and version label 1.0.1. In most cases, you should add third decimal to your version in case of some urgent bug fixes or small patches, second decimal is mostly reserved for feature updates while first decimal is incremented in case of a major app update.

## Section 175.3: Export IPA file

Once it's done, you can find your archive in the Xcode organizer. This is where all your previous versions and archive builds are saved and organized in case you do not delete them. You will immediately notice a large blue button saying "Upload to App Store..." however in 9/10 cases this will not work due to various reasons (Xcode bugs mostly). Workaround is to export your archive and upload it using another Xcode tool called Application Loader. However, since Application loader uploads IPA files to the App Store, archive needs to be exported to the correct format. This is a trivial task which might take ~ half an hour. Click on the "Export" button in the right side panel.

If you are uploading app to the App Store, select the first option and click Next. Sign in and validate your code once again and go grab a cup of coffee. Once the exporting process is done, you will be asked where to save the generated IPA file. Usually, desktop is the most convenient choice.

## Section 175.4: Upload IPA file using Application Loader

Once the IPA file is generated, open Xcode, navigate to developer tools and open Application Loader.
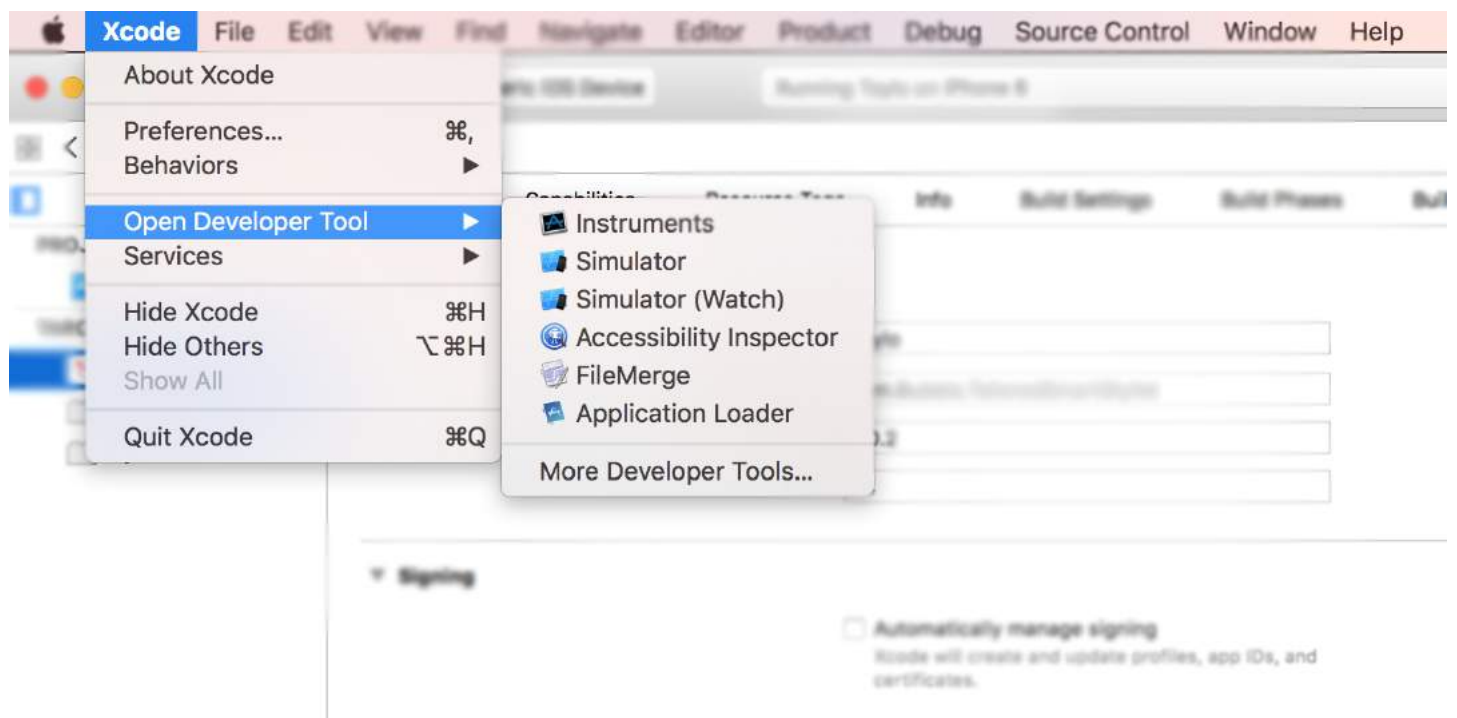


If you have multiple accounts in your Xcode, you will be asked to choose. Naturally pick the one you used for code signing in the first step. Pick "Deliver your app" and upload the code. Once the upload is done it may take up to one

hour to appear in your build list on iTunes Connect.
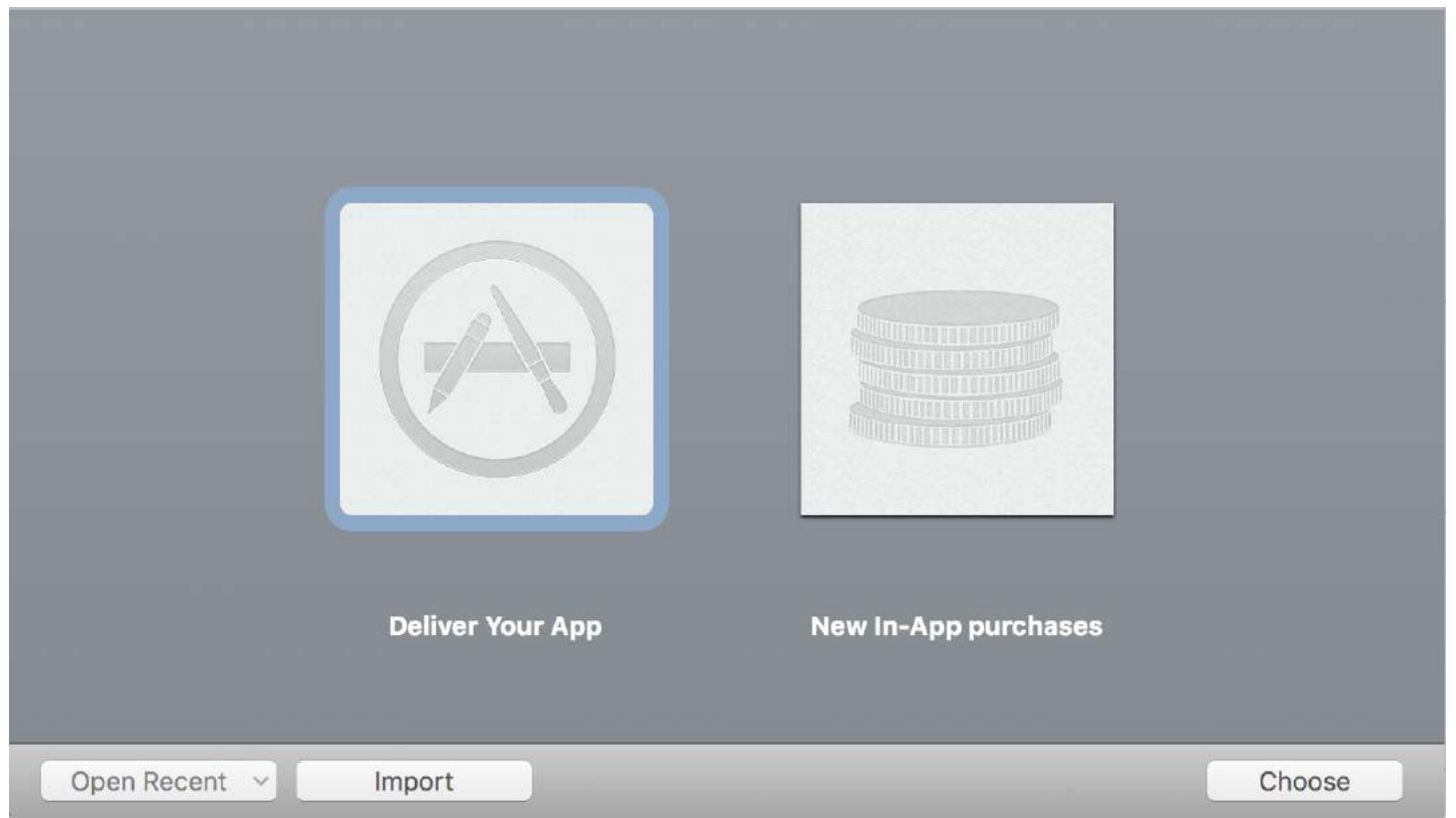
# Chapter 176: FileHandle

Read file in chunks from document directory

## Section 176.1: Read file from document directory in chunks

I get the file path from document directory and read that file in chunks of 1024 and save (append) to `NSMutableData` object or you can directly write to socket.

```
// MARK: - Get file data as chunks Methode.
func getFileDataInChunks() {

    let doumentDirectoryPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)[0] as NSString
    let filePath = doumentDirectoryPath.appendingPathComponent("video.mp4")


    //Check file exits at path or not.
    if FileManager.default.fileExists(atPath: filePath) {

        let chunkSize = 1024 // divide data into 1 kb

        //Create NSMutableData object to save read data.
        let ReadData = NSMutableData()

        do {

            //open file for reading.
            outputFileHandle = try FileHandle(forReadingFrom: URL(fileURLWithPath: filePath))

            // get the first chunk
            var datas = outputFileHandle?.readData(ofLength: chunkSize)

            //check next chunk is empty or not.
            while !(datas?.isEmpty)! {

                //here I write chunk data to ReadData or you can directly write to socket.
                ReadData.append(datas!)

                // get the next chunk
                datas = outputFileHandle?.readData(ofLength: chunkSize)

                print("Running: \(ReadData.length)")
            }

            //close outputFileHandle after reading data complete.
            outputFileHandle?.closeFile()

            print("File reading complete")

        }catch let error as NSError {
            print("Error : \(error.localizedDescription)")
        }
    }
}
```

After file reading complete you will get file Data in `ReadData` variable Here outputFileHandle is a object of `FileHandle`

```
var outputFileHandle:FileHandle?
```

# Chapter 177: Basic text file I/O

## Section 177.1: Read and write from Documents folder

**Swift 3**

```
import UIKit

// Save String to file
let fileName = "TextFile"
let documentDirectory = try FileManager.default.urlForDirectory(.documentDirectory, in:
.userDomainMask, appropriateFor: nil, create: true)

var fileURL = try documentDirectory.appendingPathComponent(fileName).appendingPathExtension("txt")

print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
  // Write to file
  try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("Failed writing to URL: \(fileURL), Error:\(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
  fromFileString = try String(contentsOfURL: fileURL)
} catch let error as NSError {
  print("Failed reading from URL: \(fileURL), Error: " + error.localizedDescription)
}
print("Text input from file: \(fromFileString)")
```

**Swift 2**

```
import UIKit

// Save String to file
let fileName = "TextFile"
let DocumentDirectoryURL = try! NSFileManager.defaultManager().URLForDirectory(.DocumentDirectory,
inDomain: .UserDomainMask, appropriateForURL: nil, create: true)

let fileURL =
DocumentDirectoryURL.URLByAppendingPathComponent(fileName).URLByAppendingPathExtension("txt")
print("FilePath: \(fileURL.path)")

var toFileString = "Text to write"
do {
  // Write to file
  try toFileString.writeToURL(fileURL, atomically: true, encoding: NSUTF8StringEncoding)
} catch let error as NSError {
  print("Failed writing to URL: \(fileURL), Error:\(error.localizedDescription)")
}

// Reading
var fromFileString = ""
do {
  fromFileString = try String(contentsOfURL: fileURL)
```

```
} catch let error as NSError {
  print("Failed reading from URL: \(fileURL), Error: " + error.localizedDescription)
}
print("Text input from file: \(fromFileString)")
```

# Chapter 178: iOS TTS

Find how to produce synthesized speech from text on an iOS device

## Section 178.1: Text To Speech

**Objective C**

```
AVSpeechSynthesizer *synthesizer = [[AVSpeechSynthesizer alloc]init];
AVSpeechUtterance *utterance = [AVSpeechUtterance speechUtteranceWithString:@"Some text"];
[utterance setRate:0.2f];
[synthesizer speakUtterance:utterance];
```

**Swift**

```
let synthesizer = AVSpeechSynthesizer()
let utterance = AVSpeechUtterance(string: "Some text")
utterance.rate = 0.2
```

You can also change the voice like this :

```
utterance.voice = AVSpeechSynthesisVoice(language: "fr-FR")
```

And then speek

- In Swift 2 : `synthesizer.speakUtterance(utterance)`
- In Swift 3 : `synthesizer.speak(utterance)`

**Don't forget to import AVFoundation**

**Helpful methods**

You can Stop or Pause all speech using these two methods :

```
- (BOOL)pauseSpeakingAtBoundary:(AVSpeechBoundary)boundary;
- (BOOL)stopSpeakingAtBoundary:(AVSpeechBoundary)boundary;
```

The AVSpeechBoundary indicates if the speech should pause or stop immediately (`AVSpeechBoundaryImmediate`) or it should pause or stop after the word currently being spoken (`AVSpeechBoundaryWord`).

# Chapter 179: MPVolumeView

The MPVolumeView class is volume view to presents the user with a slider control for setting the system audio output volume, and a button for choosing the audio output route.

## Section 179.1: Adding a MPVolumeView

```swift
// Add MPVolumeView in a holder view
let mpVolumeHolderView = UIView(frame: CGRect(x: 0, y: view.bounds.midY, width: view.bounds.width,
height: view.bounds.height))
// Set the holder view's background color to transparent
mpVolumeHolderView.backgroundColor = .clear
let mpVolume = MPVolumeView(frame: mpVolumeHolderView.bounds)
mpVolume.showsRouteButton = true
mpVolumeHolderView.addSubview(mpVolume)
view.addSubview(mpVolumeHolderView)
// the volume view is white, set the parent background to black to show it better in this example
view.backgroundColor = .black
```

!!! A very important note is that the MPVolumeView only works on an actual device and not on a simulator.
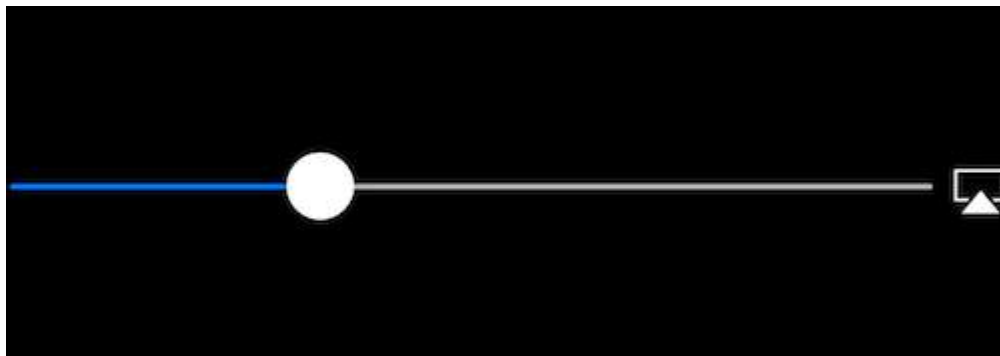
# Chapter 180: Objective-C Associated Objects

| Param | Details |
|-------|---------|
| object | The existing object you want to modify |
| key | This can basically be any pointer that has a constant memory address, but a nice practice is to use here a computed property (getter) |
| value | The object you want to add |
| policy | The memory policy for this new `value` i.e. should it be retained / assigned, copied etc.. just like any other property you'd declare |

First introduced in iOS 3.1 as part of the Objective-C runtime, associated objects provide a way to add instance variables to an existing class object (w\o subclassing.

This means you'll be able to attach any object to any other object without subclassing.

## Section 180.1: Basic Associated Object Example

Assume we need to add an NSString object to `SomeClass` (we cant subclass).

In this example we not only create an associated object but also wrap it in a computed property in a category for extra neatness

```objc
#import <objc/runtime.h>

@interface SomeClass (MyCategory)
// This is the property wrapping the associated object. below we implement the setter and getter
which actually utilize the object association
@property (nonatomic, retain) NSString *associated;
@end

@implementation SomeClass (MyCategory)

- (void)setAssociated:(NSString *)object {
    objc_setAssociatedObject(self, @selector(associated), object,
                             OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (NSString *)associated {
    return objc_getAssociatedObject(self, @selector(associated));
}
```

Now it would be as easy as this to use the property

```objc
SomeClass *instance = [SomeClass alloc] init];
instance.associated = @"this property is an associated object under the hood";
```

# Chapter 181: Passing Data between View Controllers (with MessageBox-Concept)

MessageBox is a simple concept for decoupling entities.

For example entity A can place a message that entity B can read whenever suitable.

A view controller would like to talk to another view controller, but you don't want to create a strong or weak relationship.

## Section 181.1: Simple Example Usage
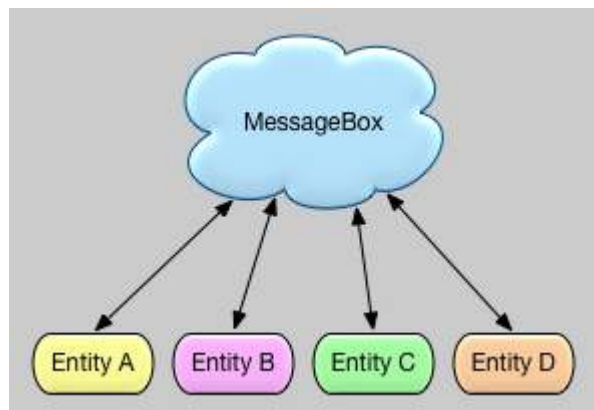


```
let messageBox:MessageBox = MessageBox()

// set
messageBox.setObject("TestObject1", forKey:"TestKey1")

// get
// but don't remove it, keep it stored, so that it can still be retrieved later
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:false)

// get
// and remove it
let someObject:String = messageBox.getObject(forKey:"TestKey1", removeIfFound:true)
```

# Chapter 182: MVVM

## Section 182.1: MVVM Without Reactive Programming

I'll start with a really short explanation what is and why use Model-View-ViewModel (MVVM) design pattern in your iOS apps. When iOS first appeared, Apple suggested to use MVC (Model-View-Controller) as a design pattern. They showed it in all of their examples and all first developers were happy using it because it nicely separated concerns between business logic and user interface. As applications became larger and more complex a new problem appeared appropriately called Massive View Controllers (MVC). Because all business logic was added in the ViewController, with time they usually became too large and complex. To avoid MVC issue, a new design pattern was introduced to the world of iOS - Model-View-ViewModel (MVVM) pattern.
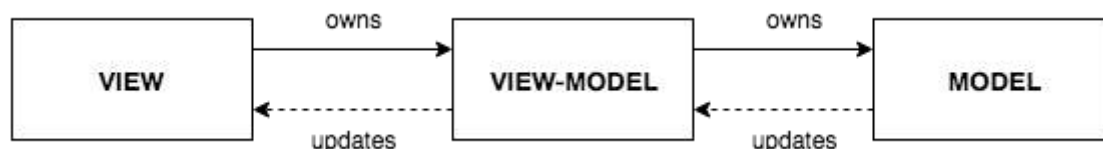


The diagram above shows how MVVM looks like. You have a standard ViewController + View (in storyboard, XIB or Code), which acts as MVVM's View (in later text - View will reference MVVM's View). A view has a reference to a ViewModel, where our business logic is. It's important to notice that ViewModel doesn't know anything about the View and never has a reference to the view. ViewModel has a reference to a Model.
This is enough with a theoretical part of the MVVM. More about it can be read here.

One of the **main issues with MVVM** is how to update View via the ViewModel when ViewModel doesn't have any references and doesn't even know anything about the View.

The main part of this example is to show how to use MVVM (more precisely, how to bind ViewModel and View) without any reactive programming (ReactiveCocoa, ReactiveSwift or RxSwif). Just as a note: if you want to use Reactive programming, even better since MVVM bindings are done really easy using it. But this example is on how to use MVVM without Reactive programming.

Let's create a simple example to demonstrate how to use MVVM.

Our `MVVMExampleViewController` is a simple ViewController with a label and a button. When button is pressed, the label text should be set to 'Hello'. Since deciding what to do on user user interaction is part of the business logic, ViewModel will have to decide what to do when user presses the button. MVVM's View shouldn't do any business logic.

```
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

`MVVMExampleViewModel` is a simple ViewModel.

```
class MVVMExampleViewModel {

    func userTriggeredSayHelloButton() {
        // How to update View's label when there is no reference to the View??
    }
}
```

You might wonder, how to set the ViewModel's reference in the View. I usually do it when ViewController is being initialized or before it will be shown. For this simple example, I would do something like this in the `AppDelegate`:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        if let rootVC = window?.rootViewController as? MVVMExampleViewController {
            let viewModel = MVVMExampleViewModel()
            rootVC.viewModel = viewModel
        }

        return true
```

**The real question now is: how to update View from ViewModel without giving a reference to the View to the ViewModel?** (Remember, we won't use any of the Reactive Programming iOS libraries)

You could think about using KVO, but that would just complicate things too much. Some smart people have thought about the issue and came up with the [Bond library](). The library might seem complicated and a little harder to understand at first, so I'll just take one small part of it and make our MVVM fully functional.

Let's introduce the `Dynamic` class which is the core of our simple yet fully functional MVVM pattern.

```
class Dynamic<T> {
    typealias Listener = (T) -> Void
    var listener: Listener?

    func bind(_ listener: Listener?) {
        self.listener = listener
    }

    func bindAndFire(_ listener: Listener?) {
        self.listener = listener
        listener?(value)
    }

    var value: T {
        didSet {
            listener?(value)
        }
    }

    init(_ v: T) {
        value = v
    }
}
```

Dynamic class is using Generics and Closures to bind our ViewModel with our View. I won't go into details about this class, we can do it in the comments (to make this example shorter). Let's now update our `MVVMExampleViewController` and `MVVMExampleViewModel` to use those classes.

Our updated `MVVMExampleViewController`

```swift
class MVVMExampleViewController: UIViewController {

    @IBOutlet weak var helloLabel: UILabel!

    var viewModel: MVVMExampleViewModel?

    override func viewDidLoad() {
        super.viewDidLoad()
        bindViewModel()
    }

    func bindViewModel() {
        if let viewModel = viewModel {
            viewModel.helloText.bind({ (helloText) in
                DispatchQueue.main.async {
                    // When value of the helloText Dynamic variable
                    // is set or changed in the ViewModel, this code will
                    // be executed
                    self.helloLabel.text = helloText
                }
            })
        }
    }

    @IBAction func sayHelloButtonPressed(_ sender: UIButton) {
        viewModel?.userTriggeredSayHelloButton()
    }
}
```

Updated `MVVMExampleViewModel`:

```swift
class MVVMExampleViewModel {

    // we have to initialize the Dynamic var with the
    // data type we want
    var helloText = Dynamic("")

    func userTriggeredSayHelloButton() {
        // Setting the value of the Dynamic variable
        // will trigger the closure we defined in the View
        helloText.value = "Hello"
    }
}
```

That is it. Your `ViewModel` is now able to update `View` without it having a reference to the `View`.

This is a really simple example, but I think you have an idea how powerful this can be. I won't go into details about benefits of the MVVM, but once you switch from MVC to MVVM, you won't go back. Just try it and see for yourself.

# Chapter 183: Cache online images

## Section 183.1: AlamofireImage

Caching Online Images Using `AlamofireImage`. It works on top of `Alamofire` in Swift. Install `AlamofireImage` using cocoapods

```
pod 'AlamofireImage', '~> 3.1'
```

**SetUp:**

1. Import `AlamofireImage` and `Alamofire`
2. SetUp the Image cache: `let imageCache = AutoPurgingImageCache( memoryCapacity: 111_111_111, preferredMemoryUsageAfterPurge: 90_000_000)`
3. Making a request and adding the Image to Cache:

```
Alamofire.request(self.nameUrl[i]).responseImage { response in
                if response.result.value != nil {
                    let image = UIImage(data: response.data!, scale: 1.0)!
                    imageCache.add(image, withIdentifier: self.nameUrl[i])
                }
    }
```

4. Retrieve Images From Cache:

```
if let image = imageCache.image(withIdentifier: self.nameUrl[self.a])
        {
            self.localImageView.image = image
        }
```

For more Info follow [this link](#)

# Chapter 184: Chain Blocks in a Queue (with MKBlockQueue)

MKBlockQueue allows you to create a chain of blocks and execute them one after the other in a queue. Compared with NSOperation, with MKBlockQueue you decide yourself when a block is complete and when you want the queue to continue. You can also pass data from one block to the next.

https://github.com/MKGitHub/MKBlockQueue

## Section 184.1: Example Code



**Block Queue**

Block 1 → Block 2 → Block 3 → Block ...
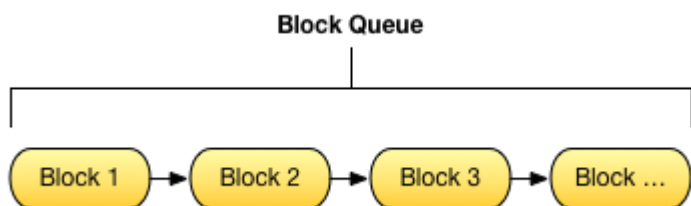
```swift
// create the dictionary that will be sent to the blocks
var myDictionary:Dictionary<String, Any> = Dictionary<String, Any>()
myDictionary["InitialKey"] = "InitialValue"

// create block queue
let myBlockQueue:MKBlockQueue = MKBlockQueue()

// block 1
let b1:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    print("Block 1 started with dictionary: \(dictionary)")
    dictionary["Block1Key"] = "Block1Value"

    // tell this block is now completed
    blockQueueObserver.blockCompleted(with:&dictionary)

}

// block 2
let b2:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // test calling on main thread, async, with delay
    DispatchQueue.main.asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 2 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block2Key"] = "Block2Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}
```

```swift
// block 3
let b3:MKBlockQueueBlockType =
{
    (blockQueueObserver:MKBlockQueueObserver, dictionary:inout Dictionary<String, Any>) in

    var copyOfDictionary:Dictionary<String, Any> = dictionary

    // test calling on global background queue, async, with delay
    DispatchQueue.global(qos:.background).asyncAfter(deadline:(.now() + .seconds(1)), execute:
    {
        print("Block 3 started with dictionary: \(copyOfDictionary)")

        copyOfDictionary["Block3Key"] = "Block3Value"

        // tell this block is now completed
        blockQueueObserver.blockCompleted(with:&copyOfDictionary)
    })
}

// add blocks to the queue
myBlockQueue.addBlock(b1)
myBlockQueue.addBlock(b2)
myBlockQueue.addBlock(b3)

// add queue completion block for the queue
myBlockQueue.queueCompletedBlock(
{
    (dictionary:Dictionary<String, Any>) in
    print("Queue completed with dictionary: \(dictionary)")
})

// run queue
print("Queue starting with dictionary: \(myDictionary)")
myBlockQueue.run(with:&myDictionary)
```

# Chapter 185: Simulator

iOS, watchOS and tvOS Simulators are great ways to test your apps without using an actual device. Here we will talk about working with Simulators.

## Section 185.1: Launching Simulator

You should go to Xcode -> Open Developer Tool:
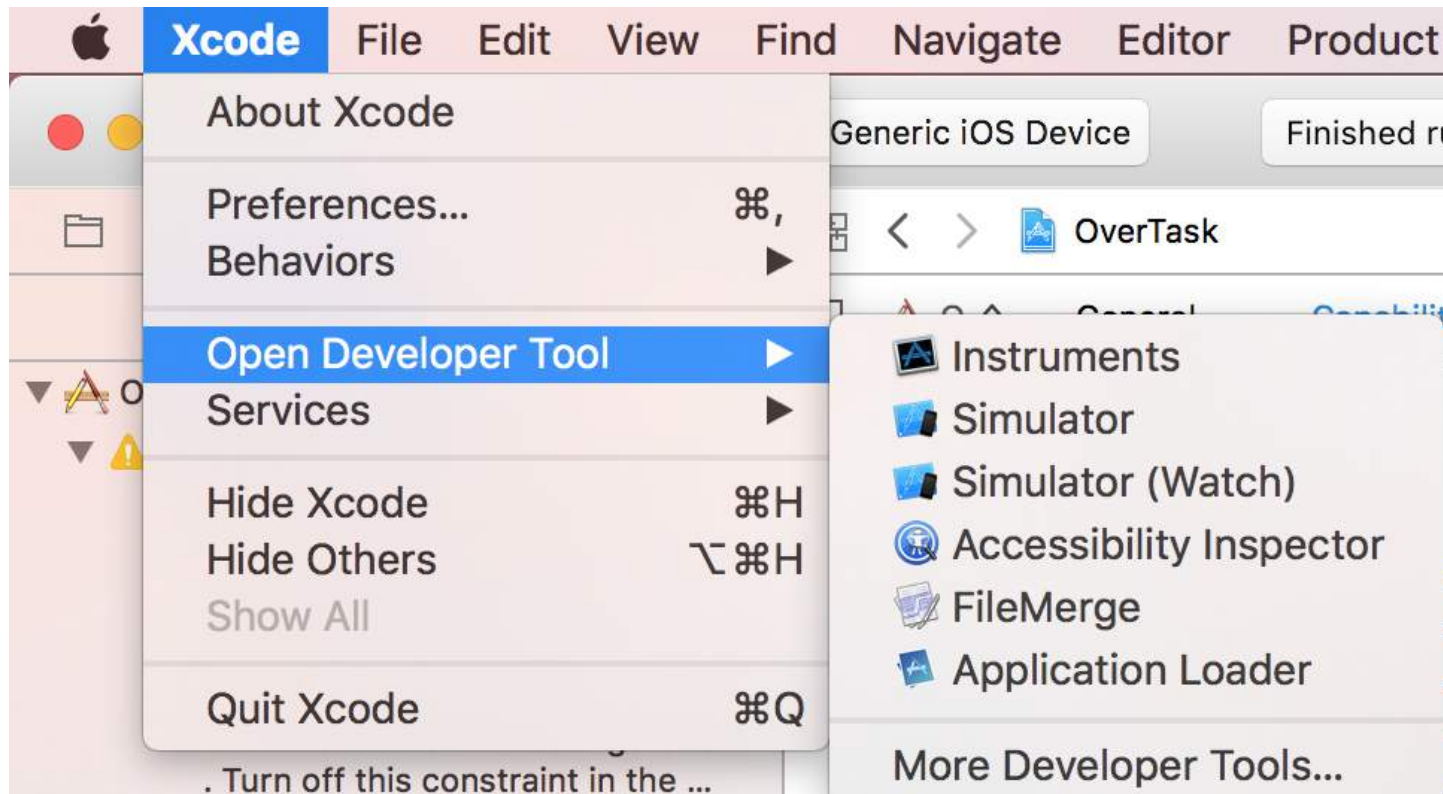


> Using "Simulator", you could run iOS and tvOS, but "Simulator (Watch)" will only run watchOS.

## Section 185.2: 3D / Force Touch simulation

Go to Hardware -> Touch Pressure:
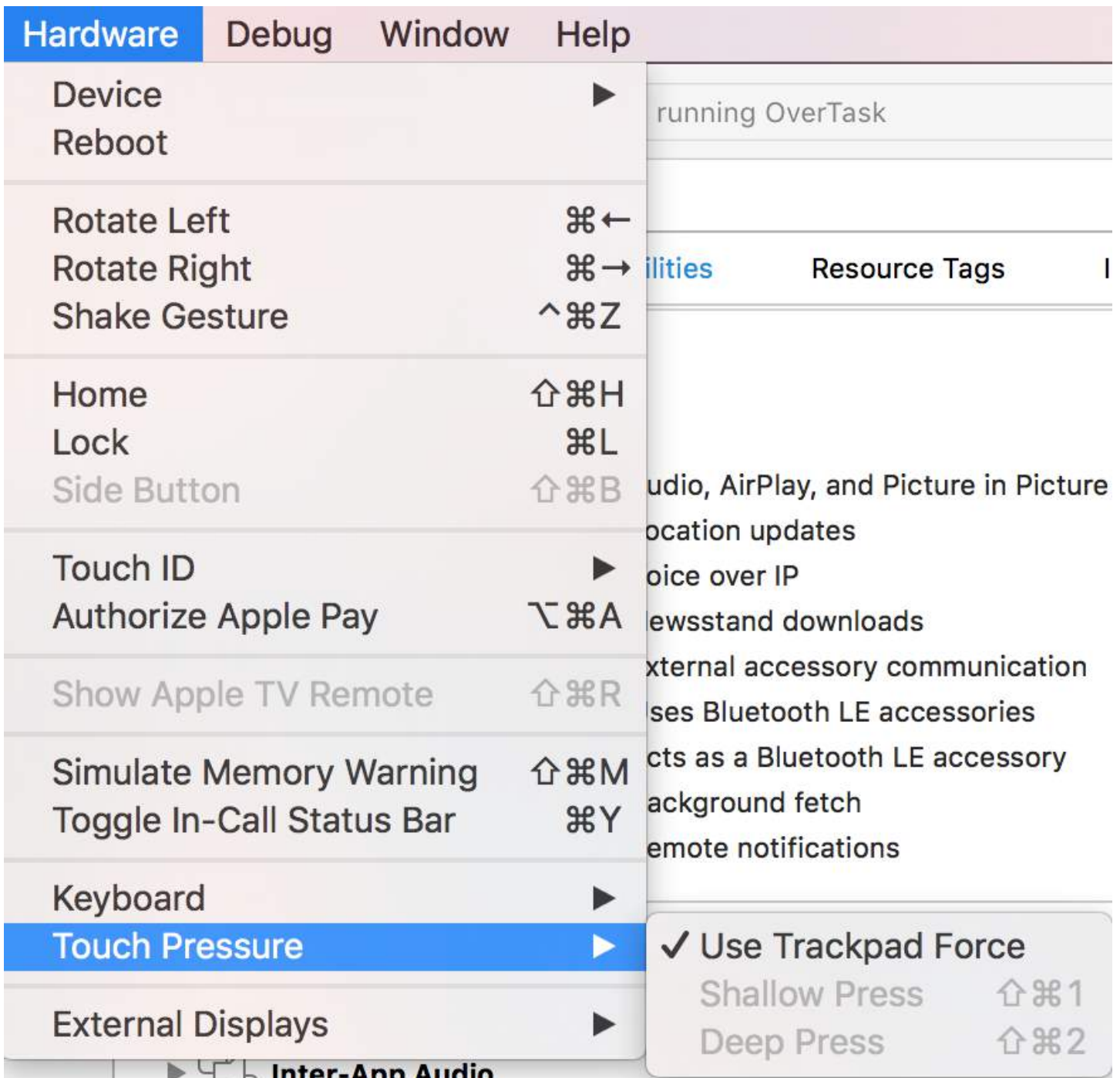
To simulate a normal press, use Shift-Command-1 and a deep one, Shift-Command-2.

> If your MacBook has a Force Touch trackpad, you could use your trackpad force to simulate 3D / Force Touch.

## Section 185.3: Change Device Model

Go to Hardware -> Device:
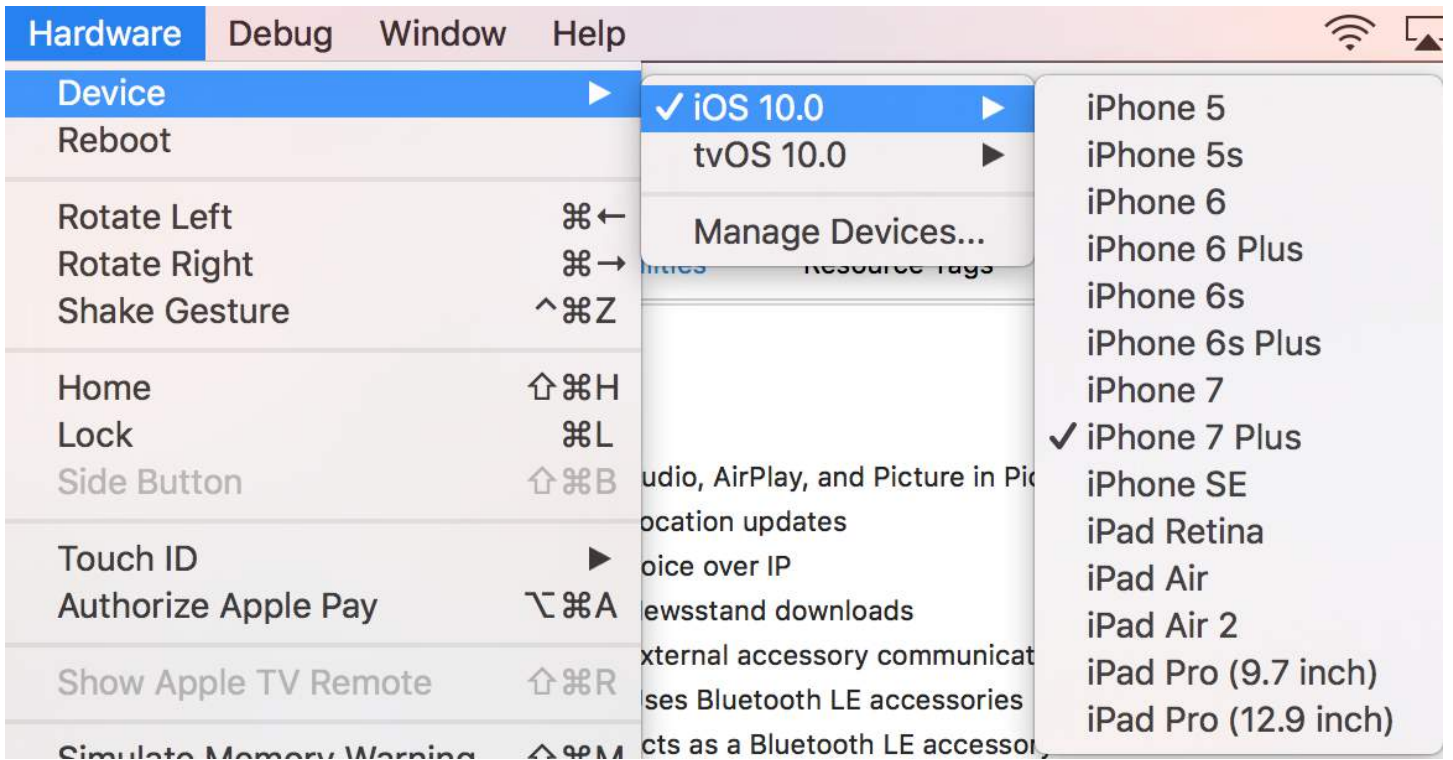
# Section 185.4: Navigating Simulator

**Home button**

You should use Shift-Command-H to have an animation when closing an app.

**Lock**

To lock the device, use Command-L.

**Rotation**

Use Command with arrow keys.

# Chapter 186: Background Modes

Being responsive is a need for every app. Users want to have apps which have their content ready when they open it, so developers should use Background Modes to make their apps more user friendly.

## Section 186.1: Turning on the Background Modes capability

1. Go to Xcode and open your project.

2. In your app target, navigate to Capabilities tab.

3. Turn on Background Modes.
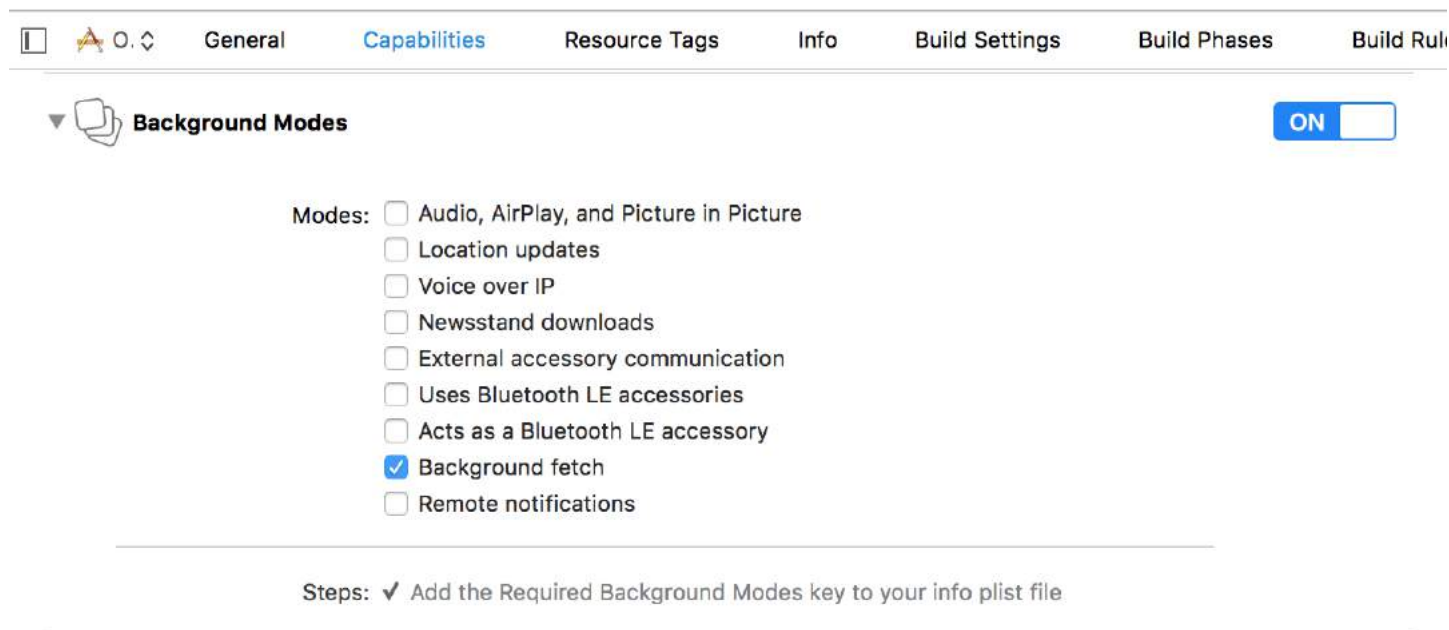


## Section 186.2: Background Fetch

Background fetch is a new mode that lets your app appear always up-to-date with the latest information while minimizing the impact on battery. You could download feeds within fixed time intervals with this capability.

To get started:

1- Check Background Fetch in capabilities screen in Xcode.

2- In `application(_:didFinishLaunchingWithOptions:)` method in `AppDelegate`, add:

**Swift**
```
UIApplication.shared.setMinimumBackgroundFetchInterval(UIApplicationBackgroundFetchIntervalMinimum)
```
**Objective-C**
```
[[UIApplication shared]
setMinimumBackgroundFetchInterval:UIApplicationBackgroundFetchIntervalMinimum]
```

> Instead of `UIApplicationBackgroundFetchIntervalMinimum`, you could use any `CGFloat` value to set fetch intervals.

3- You must implement `application(_:performFetchWithCompletionHandler:)`. Add that to your `AppDelegate`:

**Swift**

```swift
func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler:
@escaping (UIBackgroundFetchResult) -> Void) {
    // your code here
}
```

# Section 186.3: Testing background fetch

1- Run the app on a real device and attach it to Xcode debugger.

2- From Debug menu, select **Simulate Background Fetch**:
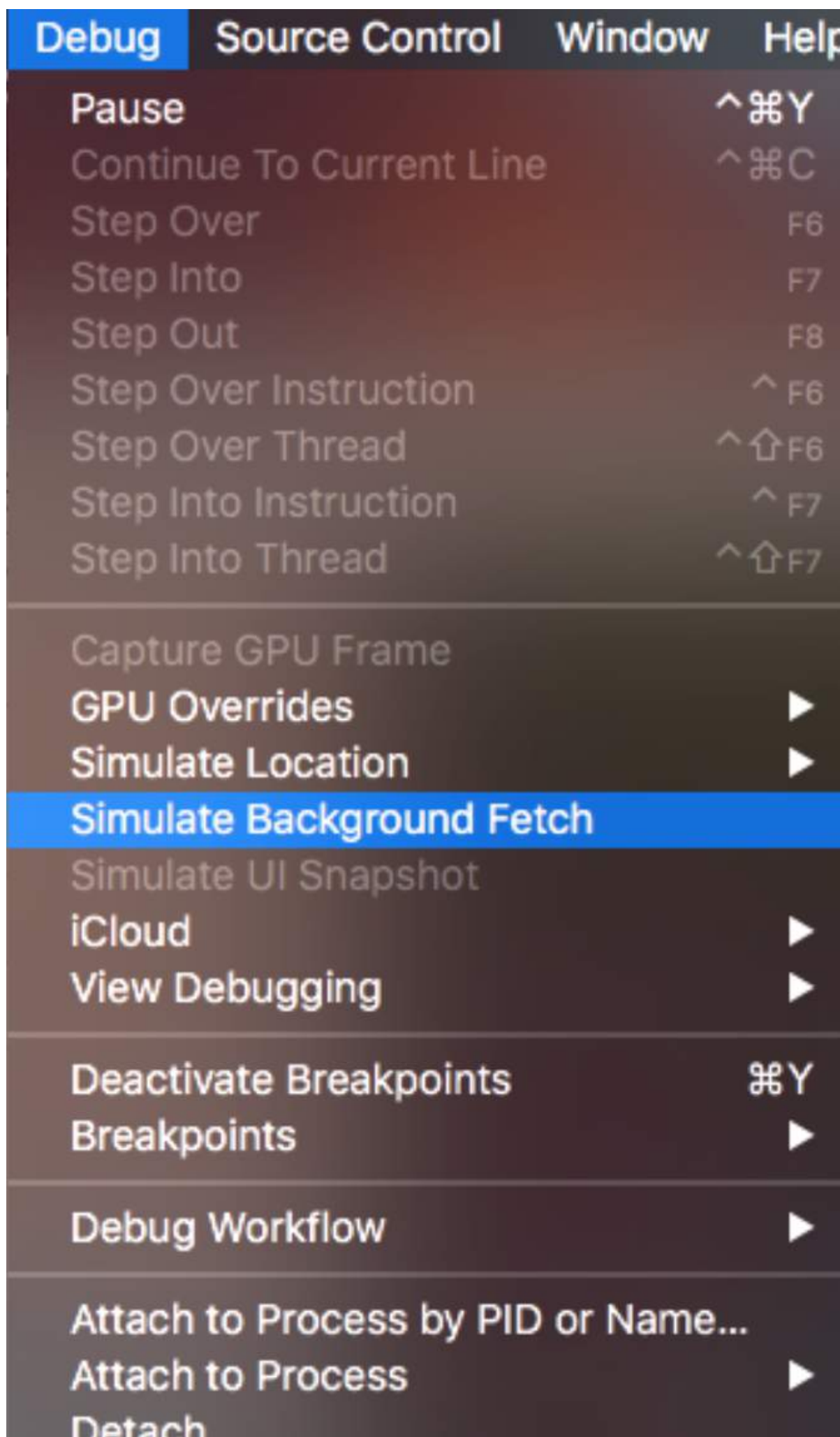
3- Now Xcode will pause the app with SIGSTOP signal. Just tap the continue button to let the app do the background fetch.
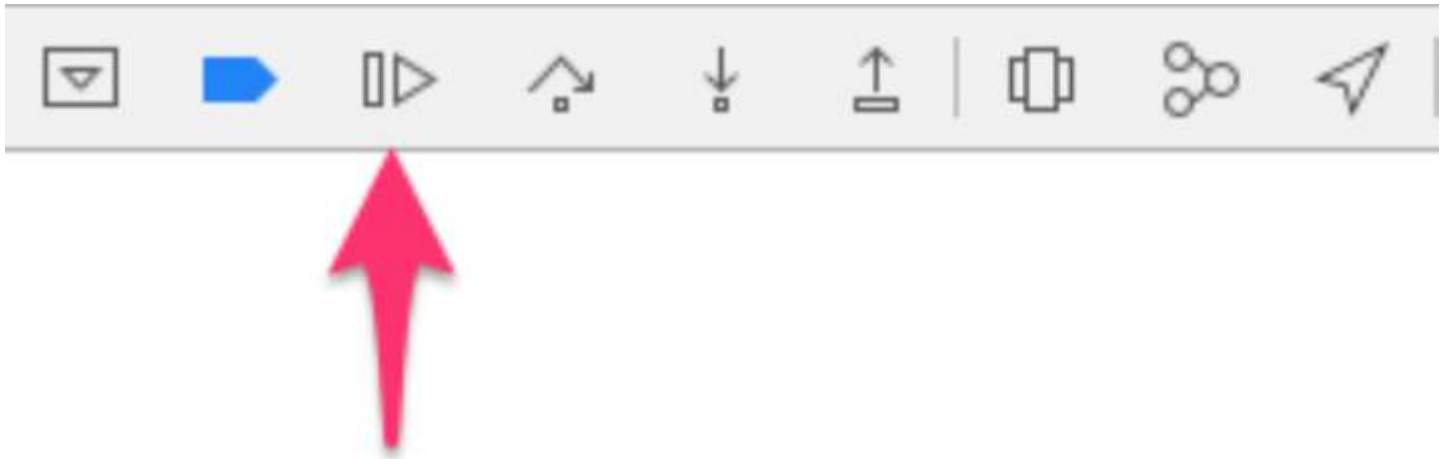


Now you will see that data is fetched and ready for you.

## Section 186.4: Background Audio

By default, when you are streaming an audio, by exiting the app it will stop, but you can prevent this by turning on the first check box in Background capability page in Xcode.

iOS will automatically handle this for you, and you don't need to write any code!

# Chapter 187: OpenGL

OpenGL ES is a graphics library that iOS uses to do 3D rendering.

## Section 187.1: Example Project

An example project (Git repo) that can be used as a starting point for doing some 3D rendering. The code for setting up OpenGL and the shaders is quite long and tedious so it wont fit well under this example format. Later pieces of it can be shown in separate examples detailing what exactly is going on with each piece of code, but for now here is the Xcode project.

# Chapter 188: MVP Architecture

MVP is an architectural pattern, a derivation of the Model–View–Controller. It's represented by three distinct components: Model, View and the Presenter. It was engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.

In examples you'll find a simple project built with MVP pattern in mind.

## Section 188.1: Dog.swift

```swift
import Foundation

enum Breed: String {
    case bulldog  = "Bulldog"
    case doberman = "Doberman"
    case labrador = "Labrador"
}

struct Dog {
    let name:  String
    let breed: String
    let age:   Int
}
```

## Section 188.2: DoggyService.swift

```swift
import Foundation

typealias Result = ([Dog]) -> Void

class DoggyService {

    func deliverDoggies(_ result: @escaping Result) {

        let firstDoggy  = Dog(name: "Alfred", breed: Breed.labrador.rawValue, age: 1)
        let secondDoggy = Dog(name: "Vinny",  breed: Breed.doberman.rawValue, age: 5)
        let thirdDoggy  = Dog(name: "Lucky",  breed: Breed.labrador.rawValue, age: 3)

        let delay = DispatchTime.now() + Double(Int64(Double(NSEC_PER_SEC)*2)) /
Double(NSEC_PER_SEC)

        DispatchQueue.main.asyncAfter(deadline: delay) {
            result([firstDoggy,
                    secondDoggy,
                    thirdDoggy])
        }
    }
}
```

## Section 188.3: DoggyPresenter.swift

```swift
import Foundation

class DoggyPresenter {

    // MARK: - Private
    fileprivate let dogService: DoggyService
```

```
    weak fileprivate var dogView: DoggyView?

    init(dogService: DoggyService){
        self.dogService = dogService
    }

    func attachView(_ attach: Bool, view: DoggyView?) {
        if attach {
            dogView = nil
        } else {
            if let view = view { dogView = view }
        }
    }

    func getDogs(){
        self.dogView?.startLoading()

        dogService.deliverDoggies { [weak self] doggies in
            self?.dogView?.finishLoading()

            if doggies.count == 0 {
                self?.dogView?.setEmpty()
            } else {
                self?.dogView?.setDoggies(doggies.map {
                    return DoggyViewData(name: "\($0.name) \($0.breed)",
                                         age: "\($0.age)")
                })
            }
        }
    }
}

struct DoggyViewData {
    let name: String
    let age:  String
}
```

# Section 188.4: DoggyView.swift

```
import Foundation

protocol DoggyView: NSObjectProtocol {
    func startLoading()
    func finishLoading()
    func setDoggies(_ doggies: [DoggyViewData])
    func setEmpty()
}
```

# Section 188.5: DoggyListViewController.swift

```
import UIKit

class DoggyListViewController: UIViewController, UITableViewDataSource {

    @IBOutlet weak var emptyView: UIView?
    @IBOutlet weak var tableView: UITableView?
    @IBOutlet weak var spinner:   UIActivityIndicatorView?

    fileprivate let dogPresenter = DoggyPresenter(dogService: DoggyService())
    fileprivate var dogsToDisplay = [DoggyViewData]()
```

```swift
    override func viewDidLoad() {
        super.viewDidLoad()

        tableView?.dataSource = self
        spinner?.hidesWhenStopped = true
        dogPresenter.attachView(true, view: self)
        dogPresenter.getDogs()
    }

    // MARK: DataSource
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return dogsToDisplay.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
{
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
        let userViewData = dogsToDisplay[indexPath.row]
        cell.textLabel?.text = userViewData.name
        cell.detailTextLabel?.text = userViewData.age
        return cell
    }
}

extension DoggyListViewController: DoggyView {

    func startLoading() {
        spinner?.startAnimating()
    }

    func finishLoading() {
        spinner?.stopAnimating()
    }

    func setDoggies(_ doggies: [DoggyViewData]) {
        dogsToDisplay = doggies
        tableView?.isHidden = false
        emptyView?.isHidden = true;
        tableView?.reloadData()
    }

    func setEmpty() {
        tableView?.isHidden = true
        emptyView?.isHidden = false;
    }
}
```

# Chapter 189: Configure Beacons with CoreBluetooth

Hot to read and write data to a bluetooth low energy device.

## Section 189.1: Showing names of all Bluetooth Low Energy (BLE)

- For this example I have a controlled room with a single BLE device enable.
- Your class should extend CBCentralManagerDelegate.
- Implement the method: centralManagerDidUpdateState(_ central: CBCentralManager).
- Use global queue to not freeze the screen while searching for a device.
- Instantiate CBCentralManager and wait for callback centralManagerDidUpdateState response.

```
class BLEController: CBCentralManagerDelegate{

var cb_manager: CBCentralManager!
var bles : [CBPeripheral] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        cb_manager = CBCentralManager(delegate: self, queue: DispatchQueue.global())
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        print("UPDATE STATE - \(central)")
    }
}
```

Callback to centralManagerDidUpdateState indicates that CoreBluetooth is ready, so you can search for BLE now. Update centralManagerDidUpdateState code to search for all BLE device when it is ready.

```
func centralManagerDidUpdateState(_ central: CBCentralManager) {
    print("UPDATE STATE - \(central)")
    SearchBLE()
}

func SearchBLE(){
    cb_manager.scanForPeripherals(withServices: nil, options: nil)
    StopSearchBLE()
}

func StopSearchBLE() {
    let when = DispatchTime.now() + 5 // change 5 to desired number of seconds
    DispatchQueue.main.asyncAfter(deadline: when) {
        self.cb_manager.stopScan()
    }
}
```

- SearchBLE() search for BLE devices and stop searching after 5s
- cb_manager.scanForPeripherals(withServices: nil, options: nil) looks for every BLE in range with you.
- StopSearchBLE() will stop the search after 5s.
- Each BLE found will callback func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber)

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral:
```

```
    CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    guard let name = peripheral.name else {
        return
    }
    print(name)
    bles.append(peripheral)
}
```

# Section 189.2: Connect and read major value

- I'm in a controlled room with a single minew beacon that use IBEACON protocol.
- BLEController needs to extend CBPeripheralDelegate
- I'll use the first BLE to connect after the search has stop.
- Modify the method StopSearchBLE()

```
class BLEController: CBCentralManagerDelegate, CBPeripheralDelegate{
//...
    func StopSearchMiniewBeacon() {
        let when = DispatchTime.now() + 5 // change 2 to desired number of seconds
        DispatchQueue.main.asyncAfter(deadline: when) {
            self.cb_manager.stopScan()
            self.cb_manager.connect(bles.first)
        }
    }
/...
}
```

- In the documention of your BLE device, you should look for the SERVICE UUID and MAJOR UUID
  CHARACTERISTIC

```
var service_uuid =  CBUUID(string: "0000fff0-0000-1000-8000-00805f9b34fb")
var major_uuid =  CBUUID(string: "0000fff2-0000-1000-8000-00805f9b34fb")
func centralManager(_ central: CBCentralManager, didConnect peripheral:
CBPeripheral) {
    peripheral.delegate = self
    peripheral.discoverServices([service_uuid])
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    print("Service: \(service)\n error: \(error)")
    peripheral.discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!)
}
```

- Create a variable 'service_uuid' and 'major_uuid' like code above. '-0000-1000-8000-00805f9b34fb' is part of
  the standard. 'fff0' is my SERVICE UUID, 'fff2' is my MAJOR UUID characteristic and '0000' are required to fill
  the 4 bytes uuid 1° block.
- discoverCharacteristics([major_uuid], for: (peripheral.services?[0])!) will get major characteristic from my
  device gatt server and it will have NIL as value for now.
- (peripheral.services?[0])! - 0 beacuse will return a single value once I did
  peripheral.discoverServices([service_uuid])

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
```

```
        }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
}
```

- Characteristic value will only be readable after call peripheral.readValue(for: characteristic)
- readValue will result in func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) with value in Data type.

## Section 189.3: Write major value

- You need discover the services and characteristic
- You don't need read value from the characteristic before writing over it.
- will continue for, for this example, after read value. Modify func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?)
- Add a variable new_major and reset_characteristic

```
var reset_characteristic : CBCharacteristic!
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService,
error: Error?) {
    for characteristic in service.characteristics! {
        print("Characteristic: \(characteristic)\n error: \(error)")
        if(characteristic.uuid.uuidString == "FFF2"){
            peripheral.readValue(for: characteristic)
        }
        if(characteristic.uuid.uuidString == "FFFF"){
            reset_characteristic = characteristic
        }
    }
}
let new_major : UInt16 = 100
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)
}
```

- iPhone by deafult will send and receive bytes in Little Endian format, but my device MINEW witch chipset NRF51822 have ARM archteture and need bytes in Big Endian format, so I have to swap it.
- BLE Device documentation will say what type of input and output each characteristic will have and if you can read it like above (CBCharacteristicWriteType.withResponse).

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic write: \(characteristic)\n error: \(error)")
    if(characteristic.uuid.uuidString == "FFF2"){
            print("Resetting")
            peripheral.writeValue("minew123".data(using: String.Encoding.utf8)!, for:
reset_characteristic, type: CBCharacteristicWriteType.withResponse)
        }
    if(characteristic.uuid.uuidString == "FFFF"){
```

```
        print("Reboot finish")
        cb_manager.cancelPeripheralConnection(peripheral)
    }
}
```

- To update a gatt server information you have to reboot it programmatically or save data to it and turn off and turn on manually.
- FFFF is characteristic that do it in this device.
- 'minew123' is the default password for reboot o save information in this case.
- run your app and watch you console for any error, I hope none, but you will not see the new value yet.

```
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic,
error: Error?) {
    print("Characteristic read: \(characteristic)\n error: \(error)")
    let major = UInt16.init(bigEndian: UInt16(data: characteristic.value!)!)
    print("major: \(major)")
    //peripheral.writeValue(new_major.data, for: characteristic, type:
CBCharacteristicWriteType.withResponse)

}
```

- Last step is to comment last line in method didUpdateValueFor and rerun the app, now you will the new value.

# Chapter 190: Core Data

Core Data is the model layer of your application in the broadest sense possible. It's the Model in the Model-View-Controller pattern that permeates the iOS SDK.

Core Data isn't the database of your application nor is it an API for persisting data to a database. Core Data is a framework that manages an object graph. It's as simple as that. Core Data can persist that object graph by writing it to disk, but that is not the primary goal of the framework.

## Section 190.1: Operations on core data

**To Get context:**

```
NSManagedObjectContext *context = ((AppDelegate*)[[UIApplication sharedApplication]
delegate]).persistentContainer.viewContext;
```

**To fetch data:**

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

**To fetch data with sorting:**

```
NSFetchRequest<EntityName *> *fetchRequest = [EntityName fetchRequest];
NSSortDescriptor *sortDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"someKey"
ascending:YES];
fetchRequest.sortDescriptors = @[sortDescriptor];
NSError *error ;
NSArray *resultArray= [context executeFetchRequest:fetchRequest error:&error];
```

**To add data:**

```
NSManagedObject *entityNameObj = [NSEntityDescription insertNewObjectForEntityForName:@"EntityName"
inManagedObjectContext:context];
[entityNameObj setValue:@"someValue" forKey:@"someKey"];
```

**To save context:**

```
[((AppDelegate*)[[UIApplication sharedApplication] delegate]) saveContext];
```

# Chapter 191: Profile with Instruments

Xcode includes a performance tuning application named Instruments that you can use to profile your application using all sorts of different metrics. They have tools to inspect CPU usage, memory usage, leaks, file/network activity, and energy usage, just to name a few. It's really easy to start profiling your app from Xcode, but it's sometimes not as easy to understand what you see when it's profiling, which deters some developers from being able to use this tool to its fullest potential.

## Section 191.1: Time Profiler

The first instrument you'll look at is the `Time Profiler`. At measured intervals, Instruments will halt the execution of the program and take a stack trace on each running thread. Think of it as pressing the pause button in Xcode's debugger.Here's a sneak preview of the Time Profiler:



This screen displays the `Call Tree`. The `Call Tree` shows the amount of time spent executing in various methods within an app. Each row is a different method that the program's execution path has followed. The time spent in each method can be determined from the number of times the profiler is stopped in each method. For instance, if **100** samples are done at **1 millisecond intervals**, and a particular method is found to be at the top of the stack in 10 samples, then you can deduce that approximately **10%** of the total execution time — **10 milliseconds** — was spent in that method. It's a fairly crude approximation, but it works!

From Xcode's menu bar, select `Product\Profile`, or press `?I`. This will build the app and launch Instruments. You will be greeted with a selection window that looks like this:
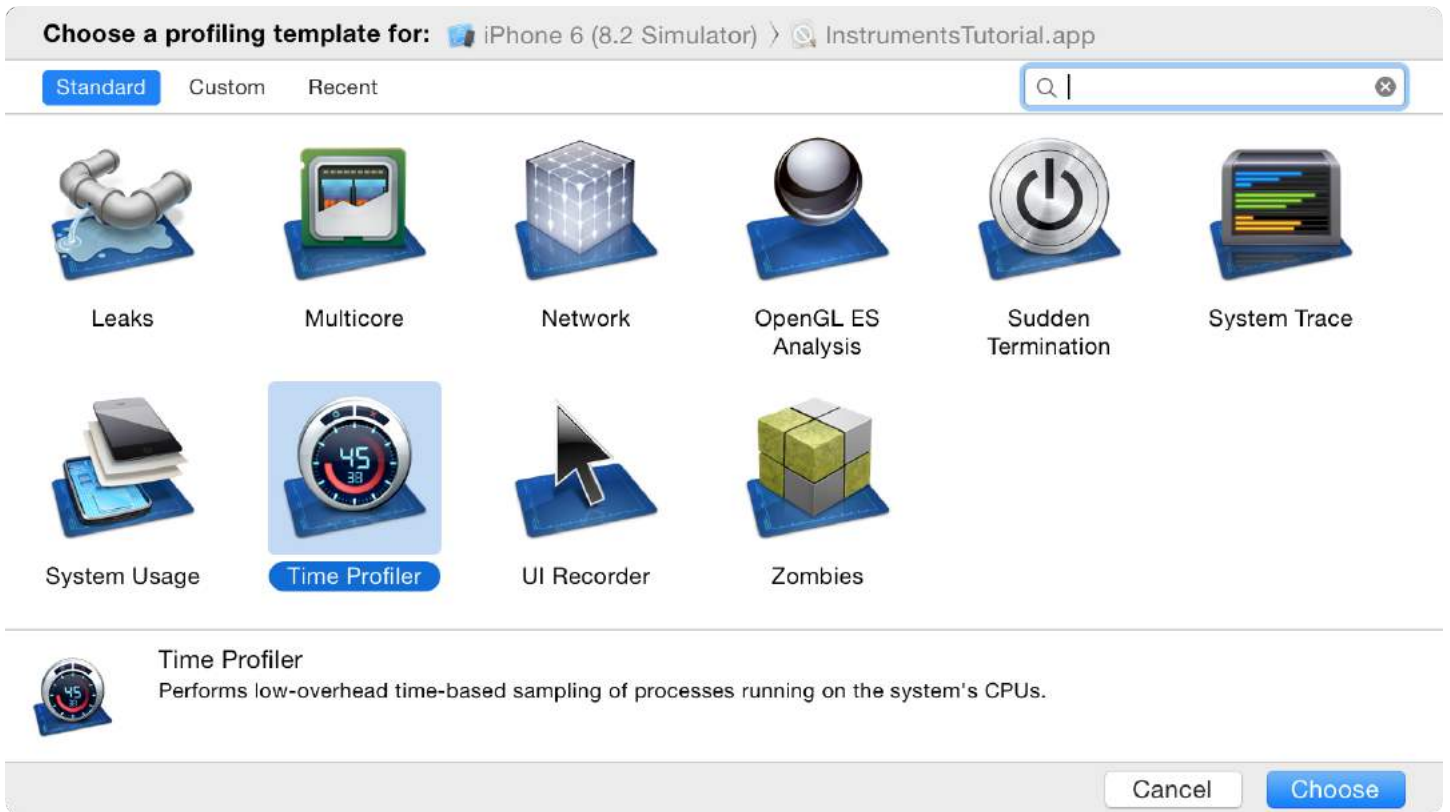
These are all different templates that come with Instruments.

Select the `Time Profiler` instrument and click Choose. This will open up a new Instruments document. Click the red **record button** in the top left to start recording and launch the app. You may be asked for your password to authorize **Instruments** to analyze other processes — fear not, it's safe to provide here! In the **Instruments window**, you can see the time counting up, and a little arrow moving from left to right above the **graph** in the center of the screen. This indicates that the app is running.

Now, start using the app. Search for some images, and drill down into one or more of the search results. You have probably noticed that going into a search result is tediously slow, and scrolling through a list of search results is also incredibly annoying – it's a terribly clunky app!

Well, you're in luck, for you're about to embark on fixing it! However, you're first going to get a quick run down on what you're looking at in **Instruments**. First, make sure the view selector on the right hand side of the toolbar has both options selected, like so:
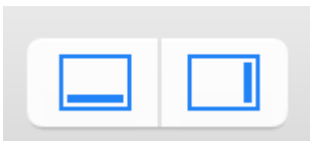


That will ensure that all panels are open. Now study the screenshot below and the explanation of each section beneath it:
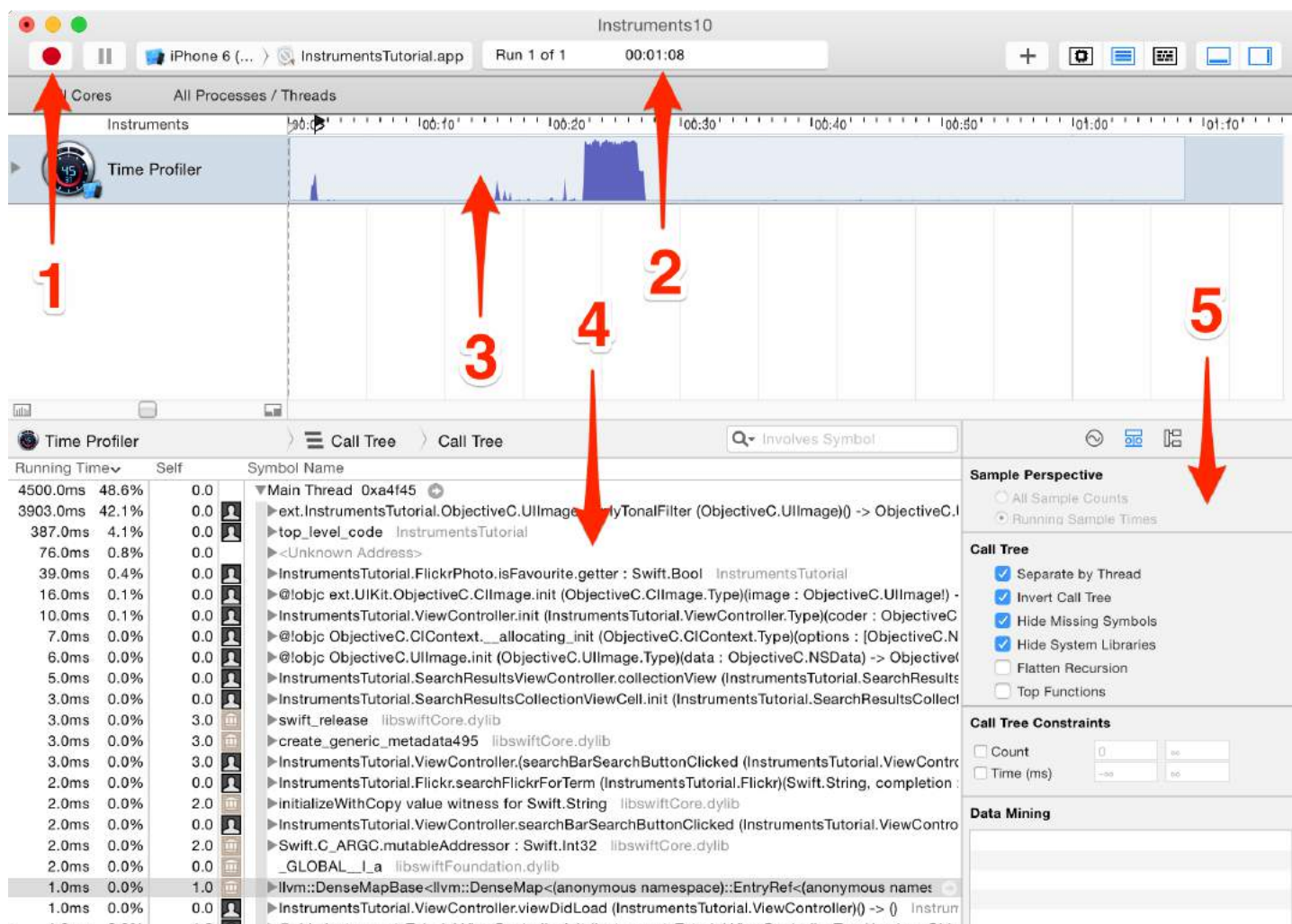
**1.** These are the **recording controls**. The red 'record' button will stop & start the app currently being profiled when it is clicked (it toggles between a record and stop icon). The pause button does exactly what you'd expect and pauses the current execution of the app.

**2.** This is the run timer. The timer counts how long the app being profiled has been running, and how many times it has been run. If you stop and then restart the app using the recording controls, that would start a new run and the display would then show Run 2 of 2.

**3.** This is called a track. In the case of the Time Profiler template you selected, there's just one instrument so there's just one track. You'll learn more about the specifics of the graph shown here later in the tutorial.

**4.** This is the detail panel. It shows the main information about the particular instrument you're using. In this case, it's showing the methods which are "hottest" — that is, the ones that have used up the most CPU time. If you click on the bar at the top which says Call Tree (the left hand one) and select Sample List, then you are presented with a different view of the data. This view is showing every single sample. Click on a few samples, and you'll see the captured stack trace appear in the Extended Detail inspector.

**5.** This is the inspectors panel. There are three inspectors: Record Settings, Display Settings, and Extended Detail. You'll be learning more about some of these options shortly.

> Drilling Deep

Perform an image search, and drill into the results. I personally like searching for "dog", but choose whatever you wish – you might be one of those cat people!

Now, scroll up and down the list a few times so that you've got a good amount of data in the `Time Profiler`. You should notice the numbers in the middle of the screen changing and the **graph** filling in; this tells you that **CPU cycles** are being used.

You really wouldn't expect any UI to be as clunky as this no `table view` is ready to ship until it scrolls like butter! To help pinpoint the problem, you need to set some options.

On the right hand side, select the **Display Settings inspector** (or press ?+2). In the **inspector**, under the `Call Tree` section, select Separate by **Thread**, Invert `Call Tree`, Hide Missing Symbols and Hide System Libraries. It will look like this:
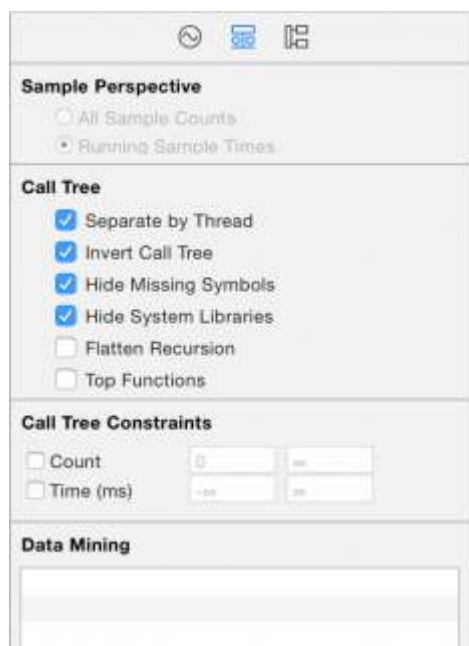


Here's what each option is doing to the data displayed in the table to the left:

**Separate by Thread:** Each thread should be considered separately. This enables you to understand which threads are responsible for the greatest amount of **CPU** use.

**Invert Call Tree:** With this option, the `stack trace` is considered from top to bottom. This is usually what you want, as you want to see the deepest methods where the **CPU** is spending its time.

**Hide Missing Symbols:** If the dSYM file cannot be found for your app or a `system framework`, then instead of seeing method names (symbols) in the table, you'll just see hex values corresponding to addresses inside the binary. If this option is selected, then only fully resolved symbols are displayed and the unresolved **hex** values are hidden. This helps to declutter the data presented.

**Hide System Libraries:** When this option is selected, only symbols from your own app are displayed. It's often useful to select this option, since usually you only care about where the **CPU** is spending time in your own code – you can't do much about how much **CPU** the `system libraries` are using!

**Flatten Recursion:** This option treats **recursive** functions (ones which call themselves) as one entry in each `stack trace`, rather than multiple.

**Top Functions:** Enabling this makes `Instruments` consider the total time spent in a function as the sum of the time directly within that function, as well as the time spent in functions called by that function.

So if function A calls B, then A's time is reported as the time spent in A PLUS the time spent in B. This can be really useful, as it lets you pick the largest time figure each time you descend into the call stack, zeroing in on your most

time-consuming methods.

If you're running an `Objective-C` app, there's also an option of Show **Obj-C Only**: If this is selected, then only `Objective-C` methods are displayed, rather than any `C` or `C++` functions. There are none in your program, but if you were looking at an `OpenGL` app, it might have some `C++`, for example.

Although some values may be slightly different, the order of the entries should be similar to the table below once you have enabled the options above:



Well, that certainly doesn't look too good. The vast majority of time is spent in the method that applies the 'tonal' filter to the thumbnail photos. That shouldn't come as too much of a shock to you, as the table loading and scrolling were the clunkiest parts of the UI, and that's when the table cells are constantly being updated.

To find out more about what's going on within that method, double click its row in the table. Doing so will bring up the following view:



Well that's interesting, isn't it! `applyTonalFilter()` is a method added to `UIImage` in an extension, and almost **100**% of the time spent in it is spent creating the CGImage output after applying the image filter.

There's not really much that can be done to speed this up: creating the image is quite an intensive process, and takes as long as it takes. Let's try stepping back and seeing where `applyTonalFilter()` is called from. **Click** `Call Tree` in the breadcrumb trail at the top of the code view to get back to the previous screen:



Now click the small arrow to the left of the applyTonalFilter row at the top of the table. This will unfold the Call Tree

to show the caller of applyTonalFilter. You may need to unfold the next row too; when profiling Swift, there will sometimes be duplicate rows in the Call Tree, prefixed with @objc. You're interested in the first row that's prefixed with your app's target name (InstrumentsTutorial):
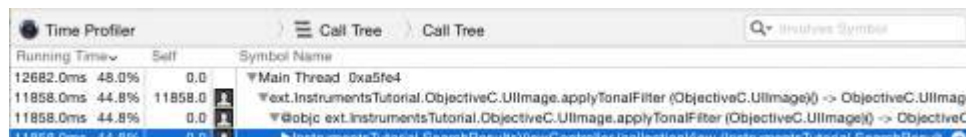
 In this case, this row refers to the results collection view's `cellForItemAtIndexPath`. Double click the row to see the associated code from the project.

Now you can see what the problem is. The method to apply the tonal filter takes a long time to execute, and it's called directly from cellForItemAtIndexPath, which will block the `main thread` (and therefore the entire UI) each time it's ask for a filtered image.

**Allocations**

There are detailed information about all the **objects** that are being created and the memory that backs them; it also shows you `retain counts` of each object. To start afresh with a new `instruments profile`, quit the Instruments app. This time, build and run the app, and open the Debug Navigator in the Navigators area. Then click on **Memory** to display graphs of memory usage in the main window:
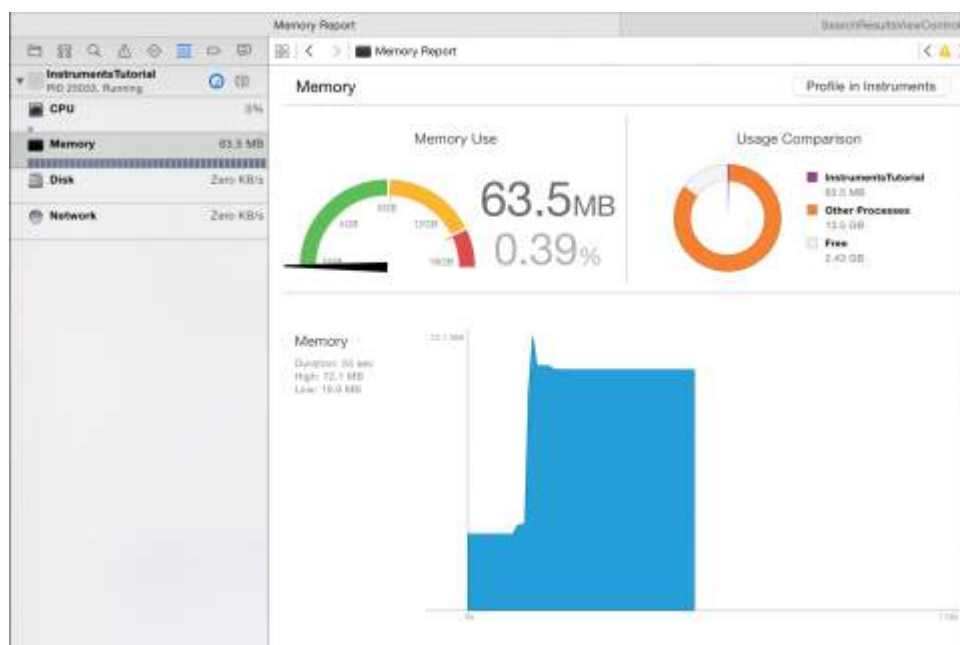


These graphs are useful for to get a quick idea about how your app is performing. But you're going to need a bit more power. Click the `Profile in Instruments` button and then Transfer to bring this session into **Instruments**. The **Allocations instrument** will start up automatically.
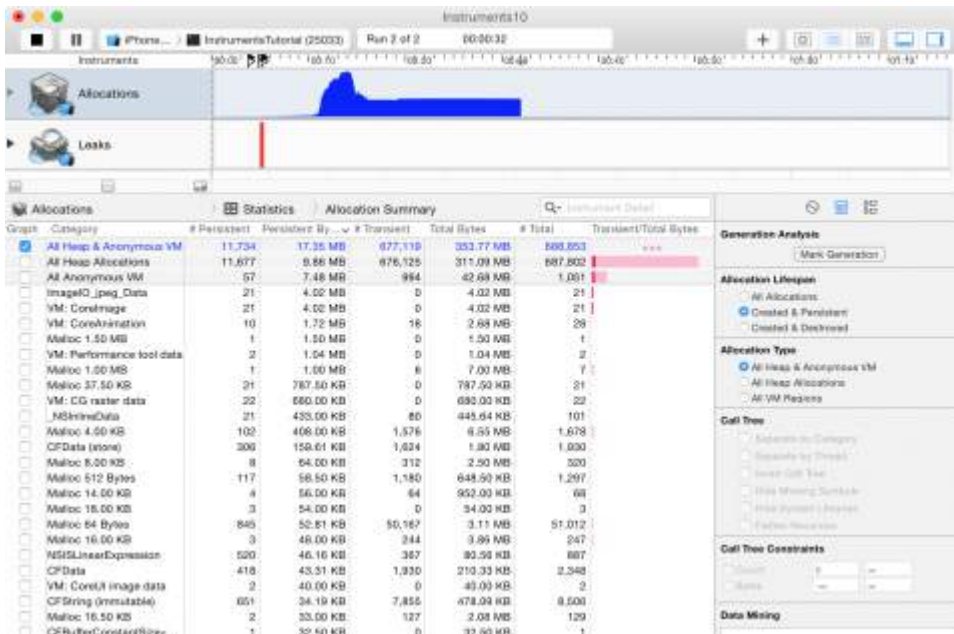
This time you'll notice two tracks. One is called Allocations, and one is called Leaks. The Allocations track will be discussed in detail later on; the Leaks track is generally more useful in Objective-C, and won't be covered in this tutorial. So what bug are you going to track down next? There's something hidden in the project that you probably don't know is there. You've likely heard about memory leaks. But what you may not know is that there are actually two kinds of leaks:

**True memory leaks** are where an object is no longer referenced by anything but still allocated – that means the memory can never be re-used. Even with Swift and `ARC` helping manage memory, the most common kind of memory leak is a `retain cycle or strong reference cycle`. This is when two objects hold strong references to one another, so that each object keeps the other one from being deallocated. This means that their memory is never released!

**Unbounded memory growth** is where memory continues to be allocated and is never given a chance to be **deallocated**. If this continues forever, then at some point the `system's memory` will be filled and you'll have a big memory problem on your hands. In iOS this means that the app will be killed by the system.

With the Allocations **instrument** running on the app, make five different searches in the app but do not drill down into the results yet. Make sure the searches have some results! Now let the app settle a bit by waiting a few seconds.

You should have noticed that the **graph** in the Allocations track has been rising. This is telling you that memory is being allocated. It's this feature that will guide you to finding `unbounded memory growth`.

What you're going to perform is a `generation analysis`. To do this, press the button called Mark Generation. You'll find the button at the top of the Display Settings inspector:
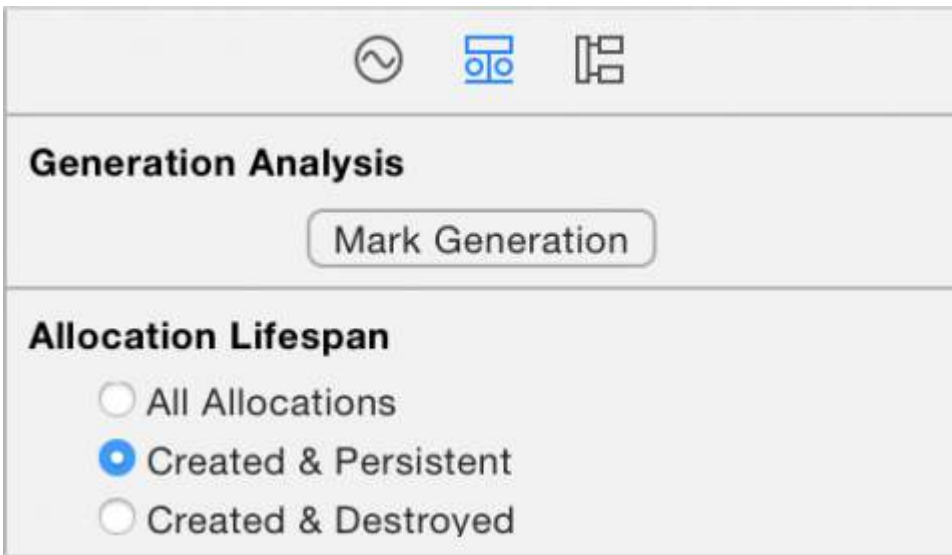
**Press it and you will see a red flag appear in the track, like so:**
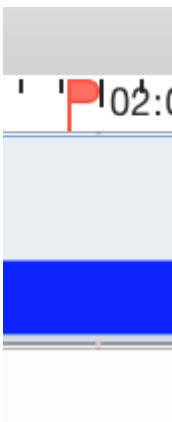


The purpose of `generation analysis` is to perform an action multiple times, and see if memory is growing in an unbounded fashion. **Drill** into a search, wait a few seconds for the images to load, and then go back to the main page. Then mark the generation again. Do this repeatedly for different searches. After a **drilling** into a few searches, **Instruments** will look like this:
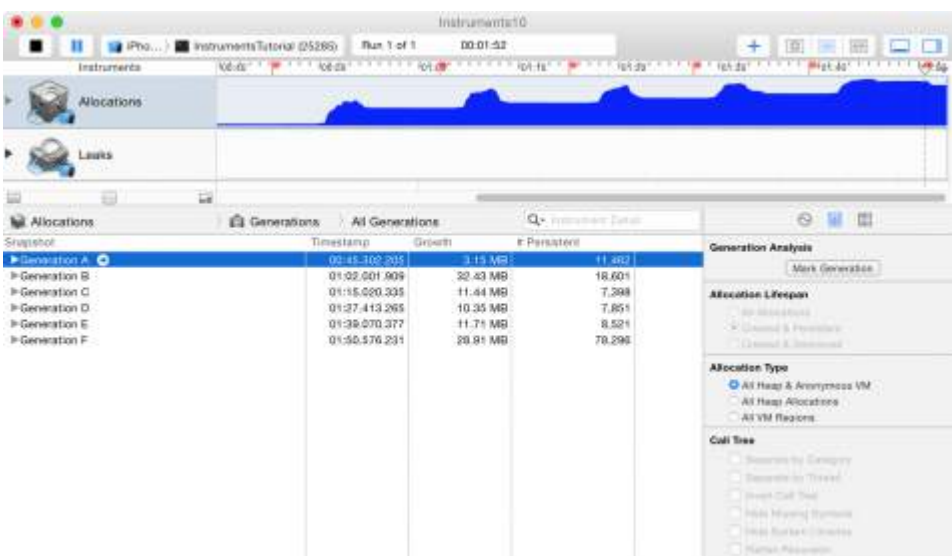


At this point, you should be getting suspicious. Notice how the blue graph is going up with each search that you **drill** into. Well, that certainly isn't good. But wait, what about `memory warnings`? You know about those, right? `Memory warnings` are iOS's way of telling an app that things are getting tight in the memory department, and you need to clear out some memory.

It's possible that this growth is not just due to your app; it could be something in the depths of `UIKit` that's holding onto memory. Give the system frameworks and your app a chance to clear their **memory** first before pointing a finger at either one.

Simulate a `memory warning` by selecting `Instrument\Simulate Memory Warning` in Instruments' menu bar, or `Hardware\Simulate Memory Warning` from the `simulator's` menu bar. You'll notice that memory usage dips a little, or perhaps not at all. Certainly not back to where it should be. So there's still **unbounded memory growth** happening somewhere.

The reason for marking a generation after each iteration of drilling into a search is that you can see what **memory** has been allocated between each generation. Take a look in the detail panel and you'll see a bunch of generations.

# Chapter 192: Application rating/review request

Now from iOS 10.3 , there is no need to navigate application to apple store for rating/review. Apple has introduced SKStoreReviewController class in storekit framework. In which developer just need to call requestReview() class method of SKStoreReviewController class and the system handles the entire process for you.

In addition, you can continue to include a persistent link in the settings or configuration screens of your app that deep-links to your App Store product page. To automatically ope

## Section 192.1: Rate/Review iOS Application

Just type below one line code from where you want user to rate/review your application.

SKStoreReviewController.requestReview()

# Chapter 193: MyLayout

MyLayout is a simple and easy objective-c framework for iOS view layout. MyLayout provides some simple functions to build a variety of complex interface. It integrates the functions including: Autolayout and SizeClass of iOS, five layout classes of Android, float and flex-box and bootstrap of HTML/CSS. you can visit from:

Objective-C: https://github.com/youngsoft/MyLinearLayout Swift: https://github.com/youngsoft/TangramKit

## Section 193.1: A Simple Demo to use MyLayout

1. There is a container view S which width is 100 and height is wrap to all subviews height. there are four subviews A,B,C,D arranged from top to bottom.
2. Subview A's left margin is 20% width of S, right margin is 30% width of S, height is equal to width of A.
3. Subview B's left margin is 40, width is filled in to residual width of S,height is 40. Subview C's width is filled in to S, height is
4. Subview D's right margin is 20, width is 50% width of S, height is 40

like below figure:
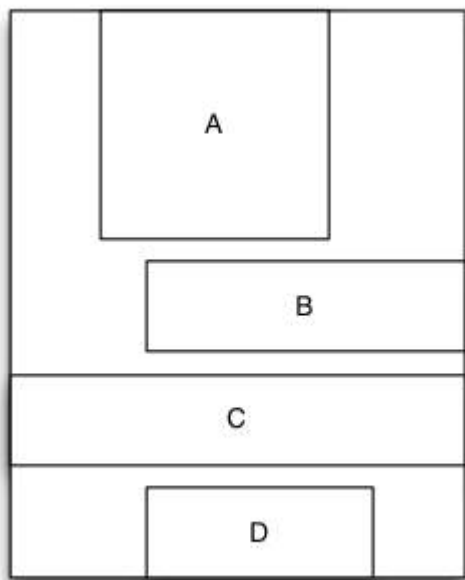


```objc
MyLinearLayout *S = [MyLinearLayout linearLayoutWithOrientation:MyLayoutViewOrientation_Vert];
S.subviewSpace = 10;
S.widthSize.equalTo(@100);

UIView *A = UIView.new;
A.leftPos.equalTo(@0.2);
A.rightPos.equalTo(@0.3);
A.heightSize.equalTo(A.widthSize);
[S addSubview:A];

UIView *B = UIView.new;
B.leftPos.equalTo(@40);
B.widthSize.equalTo(@60);
B.heightSize.equalTo(@40);
[S addSubview:B];

UIView *C = UIView.new;
```

```
C.leftPos.equalTo(@0);
C.rightPos.equalTo(@0);
C.heightSize.equalTo(@40);
[S addSubview:C];

UIView *D = UIView.new;
D.rightPos.equalTo(@20);
D.widthSize.equalTo(S.widthSize).multiply(0.5);
D.heightSize.equalTo(@40);
[S addSubview:D];
```

# Chapter 194: Simulator Builds

Where to find simulator build ?

Go to ~/Library/Developer/CoreSimulator/Devices/

You will find directories with alphanumeric names

then click on the one of the directories and make following selection

Data/Containers/Bundle/Application/

Again you will find directories with alphanumeric names if you click on that you will find

Simulator build over there

Note:

Installing iOS device build on simulator will not work out.

iPhone simulator uses i386 architecture iPad simulator builds uses x8

## Section 194.1: Installing the build manually on simulator

```
xcrun simctl install booted *.app
```

# Chapter 195: Simulating Location Using GPX files iOS

## Section 195.1: Your .gpx file: MPS_HQ.gpx

```xml
<gpx xmlns="http://www.topografix.com/GPX/1/1"
xmlns:gpxx = "http://www.garmin.com/xmlschemas/GpxExtensions/v3"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd
http://www.garmin.com/xmlschemas/GpxExtensions/v3
http://www8.garmin.com/xmlschemas/GpxExtensions/v3/GpxExtensionsv3.xsd"
version="1.1"
creator="gpx-poi.com">
 <wpt lat="38.9072" lon="77.0369">38.9072/-77.0369
 <time>2015-04-16T22:20:29Z</time>
  <name>Washington, DC</name>
  <extensions>
     <gpxx:WaypointExtension>
        <gpxx:Proximity>10</gpxx:Proximity>
        <gpxx:Address>
           <gpxx:StreetAddress>Washington DC</gpxx:StreetAddress>
           <gpxx:City>Washington</gpxx:City>
           <gpxx:State>DC</gpxx:State>
           <gpxx:Country>United States</gpxx:Country>
           <gpxx:PostalCode> 20005 </gpxx:PostalCode>
        </gpxx:Address>
     </gpxx:WaypointExtension>
  </extensions>
```

## Section 195.2: To set this location:

1. Go to Edit Scheme.
2. Select Run -> Options.
3. Check "Allow Location Simulation".
4. Select the *.GPX File Name from the "Default Location" drop down list.
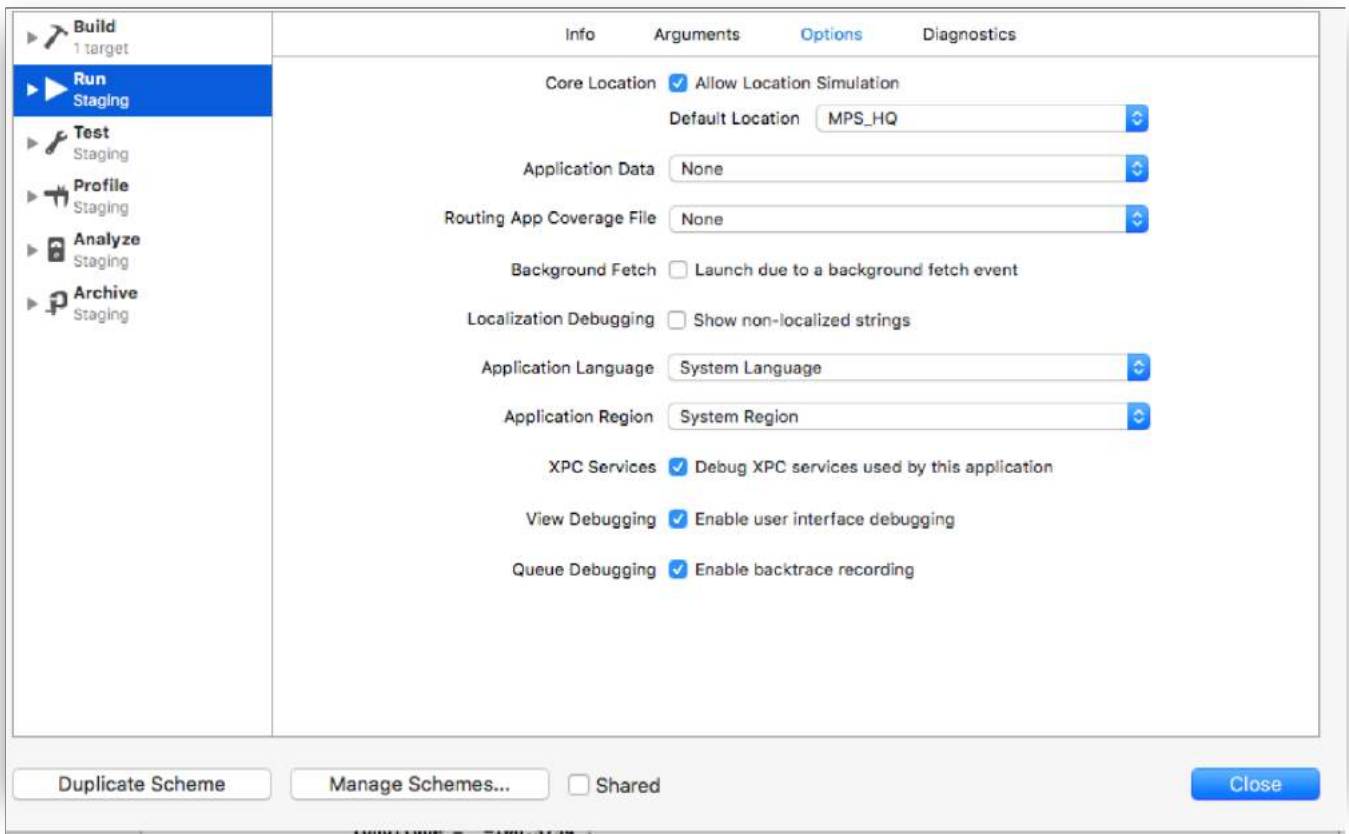
# Chapter 196: SqlCipher integration

SQLite is already a popular API for persistent data storage in iOS apps so the upside for development is obvious. As a programmer you work with a stable, well-documented API that happens to have many good wrappers available in Objective-C, such as FMDB and Encrypted Core Data. All security concerns are cleanly decoupled from application code and managed by the underlying framework.

## Section 196.1: Integration of code:

Integration to open database using password.

```
-(void)checkAndOpenDB{
    sqlite3 *db;
    NSString *strPassword = @"password";

        if (sqlite3_open_v2([[databaseURL path] UTF8String], &db, SQLITE_OPEN_READWRITE |
SQLITE_OPEN_CREATE, NULL) == SQLITE_OK) {
            const char* key = [strPassword UTF8String];
            sqlite3_key(db, key, (int)strlen(key));
            if (sqlite3_exec(db1, (const char*) "SELECT count(*) FROM sqlite_master;", NULL, NULL,
NULL) == SQLITE_OK) {
                NSLog(@"Password is correct, or a new database has been initialized");
            } else {
                NSLog(@"Incorrect password!");
            }
            sqlite3_close(db);
        }
}


- (NSURL *)databaseURL
{
    NSArray *URLs = [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask];
    NSURL *directoryURL = [URLs firstObject];
    NSURL *databaseURL = [directoryURL URLByAppendingPathComponent:@"database.sqlite"];
    return  databaseURL;
}
```

# Chapter 197: Security

Security in iOS is related to data security, transport security, code security, etc

## Section 197.1: Securing Data in iTunes Backups

If we want our app data to be protected against iTunes backups, we have to skip our app data from being backed up in iTunes.
Whenever iOS device backed up using iTunes on macOS, all the data stored by all the apps is copied in that backup and stored on backing computer.

But we can exclude our app data from this backup using `URLResourceKey.isExcludedFromBackupKey` key.
Here is the directory structure of our app:
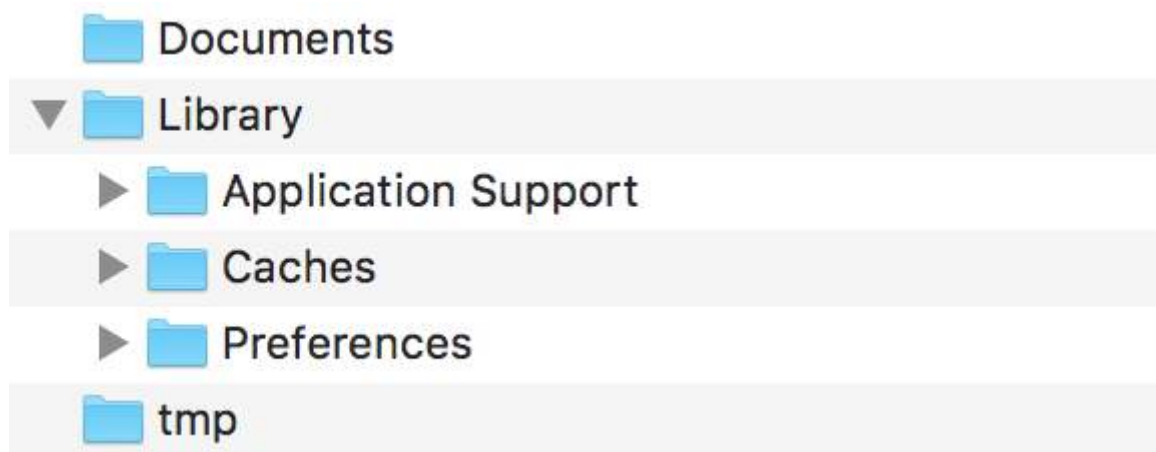


**Note:** Generally sensitive data is stored in 'Application Support' directory.

e.g. If we want to exclude all our data stored in **Application Support** directory then we can use above mentioned key as follow:

```swift
let urls = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask)
let baseURL = urls[urls.count-1];

let bundleIdentifier = Bundle.main.object(forInfoDictionaryKey: "CFBundleIdentifier") as!
String
let pathURL = baseURL.appendingPathComponent(bundleIdentifier)
let persistentStoreDirectoryPath = pathURL.path
if !FileManager.default.fileExists(atPath: persistentStoreDirectoryPath) {
    do {
        try FileManager.default.createDirectory(atPath: path, withIntermediateDirectories:
true, attributes: nil)
    }catch {
        //handle error
    }
}
let dirURL = URL.init(fileURLWithPath: persistentStoreDirectoryPath, isDirectory: true)
do {
    try (dirURL as NSURL).setResourceValue((true), forKey: .isExcludedFromBackupKey)
} catch {
    //handle error
}
```

There are lots of tools available to see iTunes backups for all the backed up data to confirm whether above approach works or not.

# Section 197.2: Transport Security using SSL

iOS apps needs to be written in a way to provide security to data which is being transported over network. SSL is the common way to do it.

Whenever app tries to call web services to pull or push data to servers, it should **use SSL over HTTP, i.e. HTTPS**. To do this, **app must call** `https://server.com/part` such web services and not `http://server.com/part`. In this case, app needs to trust the server `server.com` using SSL certificate.

Here is the example of validating server trust-

Implement `URLSessionDelegate` as:

```swift
func urlSession(_ session: URLSession, didReceive challenge: URLAuthenticationChallenge,
completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    if challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust {
        let serverTrust:SecTrust = challenge.protectionSpace.serverTrust!

        func acceptServerTrust() {
            let credential:URLCredential = URLCredential(trust: serverTrust)
            challenge.sender?.use(credential, for: challenge)
            completionHandler(.useCredential, URLCredential(trust:
challenge.protectionSpace.serverTrust!))
        }

        let success = SSLTrustManager.shouldTrustServerTrust(serverTrust, forCert:
"Server_Public_SSL_Cert")
        if success {
            acceptServerTrust()
            return
        }
    }
    else if challenge.protectionSpace.authenticationMethod ==
NSURLAuthenticationMethodClientCertificate {
        completionHandler(.rejectProtectionSpace, nil);
        return
    }
    completionHandler(.cancelAuthenticationChallenge, nil)
}
```

Here is trust manager: (couldn't found Swift code)

```objc
@implementation SSLTrustManager
+ (BOOL)shouldTrustServerTrust:(SecTrustRef)serverTrust forCert:(NSString*)certName {
// Load up the bundled certificate.
NSString *certPath = [[NSBundle mainBundle] pathForResource:certName ofType:@"der"];
NSData *certData = [[NSData alloc] initWithContentsOfFile:certPath];
CFDataRef certDataRef = (__bridge_retained CFDataRef)certData;
SecCertificateRef cert = SecCertificateCreateWithData(NULL, certDataRef);

// Establish a chain of trust anchored on our bundled certificate.
CFArrayRef certArrayRef = CFArrayCreate(NULL, (void *)&cert, 1, NULL);
SecTrustSetAnchorCertificates(serverTrust, certArrayRef);

// Verify that trust.
SecTrustResultType trustResult;
```

```
SecTrustEvaluate(serverTrust, &trustResult);

// Clean up.
CFRelease(certArrayRef);
CFRelease(cert);
CFRelease(certDataRef);

// Did our custom trust chain evaluate successfully?
return trustResult == kSecTrustResultUnspecified;
}
@end
```

**Server_Public_SSL_Cert.der** is servers' public SSL key.

Using this approach our app can make sure that it is communicating to the intended server and no one is intercepting the app-server communication.

# Chapter 198: App Transport Security (ATS)

| Parameter | Details |
| --- | --- |
| NSAppTransportSecurity | Configure ATS |
| NSAllowsArbitraryLoads | Set to YES to disable ATS everywhere. In iOS 10 and later, and macOS 10.12 and later, the value of this key is ignored if any of the following keys are present in your app's Info.plist file: NSAllowsArbitraryLoadsInMedia, NSAllowsArbitraryLoadsInWebContent, NSAllowsLocalNetworking |
| NSAllowsArbitraryLoadsInMedia | Set to YES to disable ATS for media loaded using APIs from the AV Foundation framework. *(iOS 10+, macOS 10.12+)* |
| NSAllowsArbitraryLoadsInWebContent | Set to YES to disable ATS in your app's web views (WKWebView, UIWebView, WebView) without affecting your NSURLSession connections. *(iOS 10+, macOS 10.12+)* |
| NSAllowsLocalNetworking | Set to YES to disable for connections to unqualified domains and to .local domains. *(iOS 10+, macOS 10.12+)* |
| NSExceptionDomains | Configure exceptions for specific domains |
| NSIncludesSubdomains | Set to YES to apply the exceptions to all subdomains of the selected domain. |
| NSRequiresCertificateTransparency | Set to YES to require that valid, signed Certificate Transparency (CT) timestamps, from known CT logs, be presented for server (X.509) certificates on a domain. *(iOS 10+, macOS 10.12+)* |
| NSExceptionAllowsInsecureHTTPLoads | Set to YES to allow HTTP on the selected domain. |
| NSExceptionRequiresForwardSecrecy | Defaults to YES; Set to NO to disable Forward Secrecy and accept more ciphers. |
| NSExceptionMinimumTLSVersion | Defaults to TLSv1.2; Possible values are: TLSv1.0, TLSv1.1, TLSv1.2 |
| NSThirdPartyExceptionAllowsInsecureHTTPLoads | Similar to NSExceptionAllowsInsecureHTTPLoads, but for domains that you have no control over |
| NSThirdPartyExceptionRequiresForwardSecrecy | Similar to NSExceptionRequiresForwardSecrecy, but for domains that you have no control over |
| NSThirdPartyExceptionMinimumTLSVersion | Similar to NSExceptionMinimumTLSVersion, but for domains that you have no control over |

## Section 198.1: Load all HTTP content

Apple introduced ATS with iOS 9 as a new security feature to improve privacy and security between apps and web services. ATS by default fails all non HTTPS requests. While this can be really nice for production environments, it can be a nuisance during testing.

ATS is configured in the target's Info.plist file with the NSAppTransportSecurity dictionary (App Transport Security Settings in the Xcode Info.plist editor). To allow all HTTP content, add the Allow Arbitrary Loads boolean (NSAllowsArbitraryLoads) and set it to YES. This is not recommended for production apps, and if HTTP content is required, it is recommended that it be selectively enabled instead.

## Section 198.2: Selectively load HTTP content

Similar to enabling all HTTP content, all configuration happens under the App Transport Security Settings. Add the Exception Domains dictionary (NSExceptionDomains) to the top level ATS settings.

For every domain, add a dictionary item to the Exception Domains, where the key is the domain in question. Set NSExceptionAllowsInsecureHTTPLoads to YES to disable the HTTPS requirement for that domain.

# Section 198.3: Endpoints require SSL

Introduced in iOS 9, all endpoints must adhere to the HTTPS specification.
Any endpoints not using SSL will fail with a warning in the console log. To your application it will appear that the internet connection failed.

To configure exceptions: Place the following in your Info.plist file:

1. Allow particular domain (testdomain.com) **only**:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSExceptionDomains</key>
<dict>
    <key>testdomain.com</key>
    <dict>
        <key>NSIncludesSubdomains</key>
        <true/>
        <key>NSExceptionAllowsInsecureHTTPLoads</key>
        <true/>
    </dict>
</dict>
</dict>
```

The key that permits such behavior is `NSExceptionAllowsInsecureHTTPLoads`. In this case, app will allow HTTP connection to mentioned domain (testdomain.com) only and block all other HTTP connections.

The key `NSIncludesSubdomains` specifies that any and all **subdomains** of the mentioned domain (testdomain.com) should also be allowed.

2. Allow any domain:

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

In this case, app will allow HTTP connection to **any** domain. As of January 1st 2017, using this flag will cause thorough App Store review and the app developers will have to explain why they need to use this exception in the first place. Possible explanations include:

- An application that loads encrypted media content that contains no personalized information.
- Connections to devices that cannot be upgraded to use secure connections.
- Connection to a server that is managed by another entity and does not support secure connections.

# Chapter 199: Guideline to choose best iOS Architecture Patterns

Demystifying MVC, MVP, MVVM and VIPER or any other design patterns to choose the best approach to building an app

## Section 199.1: MVP Patterns

```swift
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

protocol GreetingViewPresenter {
    init(view: GreetingView, person: Person)
    func showGreeting()
}

class GreetingPresenter : GreetingViewPresenter {
    unowned let view: GreetingView
    let person: Person
    required init(view: GreetingView, person: Person) {
        self.view = view
        self.person = person
    }
    func showGreeting() {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var presenter: GreetingViewPresenter!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.presenter.showGreeting()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of MVP
```

```swift
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
let presenter = GreetingPresenter(view: view, person: model)
view.presenter = presenter
```

## Section 199.2: MVVM Pattern

```swift
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

protocol GreetingViewModelProtocol: class {
    var greeting: String? { get }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())? { get set } // function to call
when greeting did change
    init(person: Person)
    func showGreeting()
}

class GreetingViewModel : GreetingViewModelProtocol {
    let person: Person
    var greeting: String? {
        didSet {
            self.greetingDidChange?(self)
        }
    }
    var greetingDidChange: ((GreetingViewModelProtocol) -> ())?
    required init(person: Person) {
        self.person = person
    }
    func showGreeting() {
        self.greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
    }
}

class GreetingViewController : UIViewController {
    var viewModel: GreetingViewModelProtocol! {
        didSet {
            self.viewModel.greetingDidChange = { [unowned self] viewModel in
                self.greetingLabel.text = viewModel.greeting
            }
        }
    }
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self.viewModel, action: "showGreeting", forControlEvents:
.TouchUpInside)
    }
    // layout code goes here
}
// Assembling of MVVM
let model = Person(firstName: "David", lastName: "Blaine")
let viewModel = GreetingViewModel(person: model)
let view = GreetingViewController()
```

```
view.viewModel = viewModel
```

# Section 199.3: VIPER Pattern

```swift
import UIKit

struct Person { // Entity (usually more complex e.g. NSManagedObject)
    let firstName: String
    let lastName: String
}

struct GreetingData { // Transport data structure (not Entity)
    let greeting: String
    let subject: String
}

protocol GreetingProvider {
    func provideGreetingData()
}

protocol GreetingOutput: class {
    func receiveGreetingData(greetingData: GreetingData)
}

class GreetingInteractor : GreetingProvider {
    weak var output: GreetingOutput!

    func provideGreetingData() {
        let person = Person(firstName: "David", lastName: "Blaine") // usually comes from data
access layer
        let subject = person.firstName + " " + person.lastName
        let greeting = GreetingData(greeting: "Hello", subject: subject)
        self.output.receiveGreetingData(greeting)
    }
}

protocol GreetingViewEventHandler {
    func didTapShowGreetingButton()
}

protocol GreetingView: class {
    func setGreeting(greeting: String)
}

class GreetingPresenter : GreetingOutput, GreetingViewEventHandler {
    weak var view: GreetingView!
    var greetingProvider: GreetingProvider!

    func didTapShowGreetingButton() {
        self.greetingProvider.provideGreetingData()
    }

    func receiveGreetingData(greetingData: GreetingData) {
        let greeting = greetingData.greeting + " " + greetingData.subject
        self.view.setGreeting(greeting)
    }
}

class GreetingViewController : UIViewController, GreetingView {
    var eventHandler: GreetingViewEventHandler!
    let showGreetingButton = UIButton()
```

```
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        self.eventHandler.didTapShowGreetingButton()
    }

    func setGreeting(greeting: String) {
        self.greetingLabel.text = greeting
    }

    // layout code goes here
}
// Assembling of VIPER module, without Router
let view = GreetingViewController()
let presenter = GreetingPresenter()
let interactor = GreetingInteractor()
view.eventHandler = presenter
presenter.view = view
presenter.greetingProvider = interactor
interactor.output = presenter
```

## Section 199.4: MVC pattern

```
import UIKit

struct Person { // Model
    let firstName: String
    let lastName: String
}

class GreetingViewController : UIViewController { // View + Controller
    var person: Person!
    let showGreetingButton = UIButton()
    let greetingLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
.TouchUpInside)
    }

    func didTapButton(button: UIButton) {
        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
        self.greetingLabel.text = greeting

    }
    // layout code goes here
}
// Assembling of MVC
let model = Person(firstName: "David", lastName: "Blaine")
let view = GreetingViewController()
view.person = model
```

# Chapter 200: Multicast Delegates

Pattern for adding multicasting capabilities to existing iOS controls. Adding multicasting allows for improved clarity and code re-use.

## Section 200.1: Multicast delegates for any controls

Forward messages one object to another by delegates, multicasting these messages to multiple observers..

**Step 1:** Create `NSObject` **class** of RRMulticastDelegate

**Step 2:** Following code implement in RRMulticastDelegate.`h` file

```objc
#import <Foundation/Foundation.h>

@interface RRMulticastDelegate : NSObject

{
    //Handle multiple observers of delegate
    NSMutableArray* _delegates;
}

// Delegate method implementation to the list of observers
- (void)addDelegate:(id)delegate;
- (void)removeDelegate:(id)delegate;

// Get multiple delegates
-(NSArray *)delegatesObjects;

@end
```

**Step 3:** Following code implement in RRMulticastDelegate.`m` file

```objc
#import "RRMulticastDelegate.h"

@implementation RRMulticastDelegate

- (id)init
{
    if (self = [super init])
    {
        _delegates = [NSMutableArray array];
    }
    return self;
}

-(NSArray *)delegatesObjects
{
    return _delegates;
}

- (void)removeDelegate:(id)delegate
{
    if ([_delegates containsObject:delegate])
        [_delegates removeObject:delegate];
}

- (void)addDelegate:(id)delegate
```

```objc
{
    if (![_delegates containsObject:delegate])
        [_delegates addObject:delegate];
}

- (BOOL)respondsToSelector:(SEL)aSelector
{
    if ([super respondsToSelector:aSelector])
        return YES;

    // if any of the delegates respond to this selector, return YES
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:aSelector])
        {
            return YES;
        }
    }
    return NO;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
{
    // can this class create the sinature?
    NSMethodSignature* signature = [super methodSignatureForSelector:aSelector];

    // if not, try our delegates
    if (!signature)
    {
        for(id delegate in _delegates)
        {
            if (!delegate)
                continue;

            if ([delegate respondsToSelector:aSelector])
            {
                return [delegate methodSignatureForSelector:aSelector];
            }
        }
    }
    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    // forward the invocation to every delegate
    for(id delegate in _delegates)
    {
        if (!delegate)
            continue;

        if ([delegate respondsToSelector:[anInvocation selector]])
        {
            [anInvocation invokeWithTarget:delegate];
        }
    }
}

@end
```

**Step 4:** Create `NSObject` **category class** of `RRProperty`

**Step 5:** Following code implement in `NSObject`+RRProperty.`h` file

```objc
#import <Foundation/Foundation.h>

#import "RRMulticastDelegate.h"

@interface NSObject (RRProperty)<UITextFieldDelegate,UITableViewDataSource>

-(void)setObject:(id)block forKey:(NSString *)key;
-(id)objectForKey:(NSString *)key;

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate;
- (RRMulticastDelegate *)multicastDatasource;

-(void)addDelegate:(id)delegate;
-(void)addDataSource:(id)datasource;

@end
```

**Step 6:** Following code implement in `NSObject`+RRProperty.`m` file

```objc
#import "NSObject+RRProperty.h"

#import <objc/message.h>
#import <objc/runtime.h>

#pragma GCC diagnostic ignored "-Wprotocol"

static NSString *const MULTICASTDELEGATE = @"MULTICASTDELEGATE";
static NSString *const MULTICASTDATASOURCE = @"MULTICASTDATASOURCE";

@implementation NSObject (RRProperty)

-(void)setObject:(id)block forKey:(NSString *)key
{
    objc_setAssociatedObject(self, (__bridge const void *)(key), block, OBJC_ASSOCIATION_RETAIN);
}

-(id)objectForKey:(NSString *)key
{
    return objc_getAssociatedObject(self, (__bridge const void *)(key));
}

#pragma mark - Multicast Delegate

- (RRMulticastDelegate *)multicastDelegate
{
    id multicastDelegate = [self objectForKey:MULTICASTDELEGATE];
    if (multicastDelegate == nil) {
        multicastDelegate = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDelegate forKey:MULTICASTDELEGATE];
    }

    return multicastDelegate;
}
```

```objc
- (RRMulticastDelegate *)multicastDatasource
{
    id multicastDatasource = [self objectForKey:MULTICASTDATASOURCE];
    if (multicastDatasource == nil) {
        multicastDatasource = [[RRMulticastDelegate alloc] init];

        [self setObject:multicastDatasource forKey:MULTICASTDATASOURCE];
    }

    return multicastDatasource;
}

-(void)addDelegate:(id)delegate
{
    [self.multicastDelegate addDelegate:delegate];

    UITextField *text = (UITextField *) self;
    text.delegate = self.multicastDelegate;
}

-(void)addDataSource:(id)datasource
{
    [self.multicastDatasource addDelegate:datasource];
    UITableView *text = (UITableView *) self;
    text.dataSource = self.multicastDatasource;
}

@end
```

Finally you ca use multicast dalegate for any controls...

For ex...

Import your viewcontroller class in `NSObject+RRProperty.h` file to access its methods to **set multicast delegate/datasource**.

```objc
UITextView *txtView = [[UITextView alloc]initWithFrame:txtframe];
[txtView addDelegate:self];

UITableView *tblView = [[UITableView alloc]initWithFrame:tblframe];
[tblView addDelegate:self];
[tblView addDataSource:self];
```

# Chapter 201: Using Image Aseets

Image assets are used to manage and organize different types of image assets in our iOS app using Xcode.

These assets can be **App Icons, Launch Images, images used throughout the app, full size images, random sized images** etc.

## Section 201.1: LaunchImage using Image Assets

Launch screen is a screen which appears while launching app and lasts till first screen of app appears.
Learn more about Launch Screen and guidelines here.

Similar to AppIcons we have to mention in project settings about using image assets for launch screen image.
By default project settings are like:



We have to change to like this:



Once we change these settings, Xcode will asks us to migrate to assets and create LaunchImage file in assets automatically as:
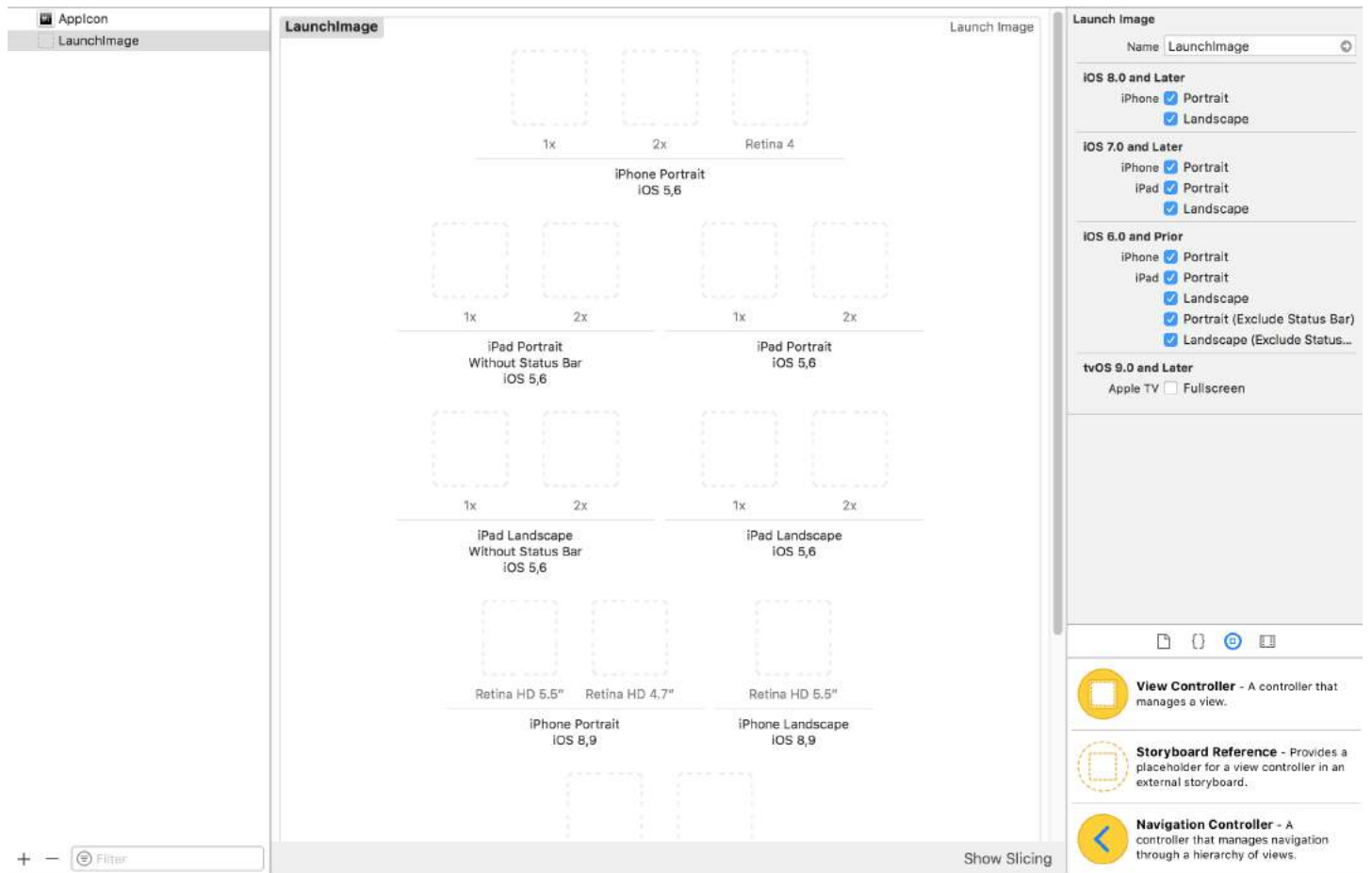
If not created, we can manually create one by clicking **+** button at the bottom as:

AppIcon

New Image Set
New Data Set
New Texture Set
New Cube Texture Set
App Icons & Launch Images ▶
New Watch Complication

New Apple TV Image Stack
New Apple TV Image Stack Layer

New Folder
New Folder from Selection
New Sprite Atlas

Game Center ▶

New Sticker Pack
New Sticker
New Sticker Sequence
New Sticker Sequence Frame

Import...
Import From Project...

New iOS App Icon
New iOS Launch Image

New tvOS App Icon and Top Shelf Image
New tvOS Launch Image

New watchOS App Icon

New macOS App Icon
New macOS Generic Icon

New Messages Extension Icon

No Select

＋ － ⊜ Filter

After this, according to our requirement we can change the empty boxes to devices which we support using attributes inspector by checking/unchecking boxes.

I filled these images for iPhones of 4" screen to 5.5" and for all iPads as:
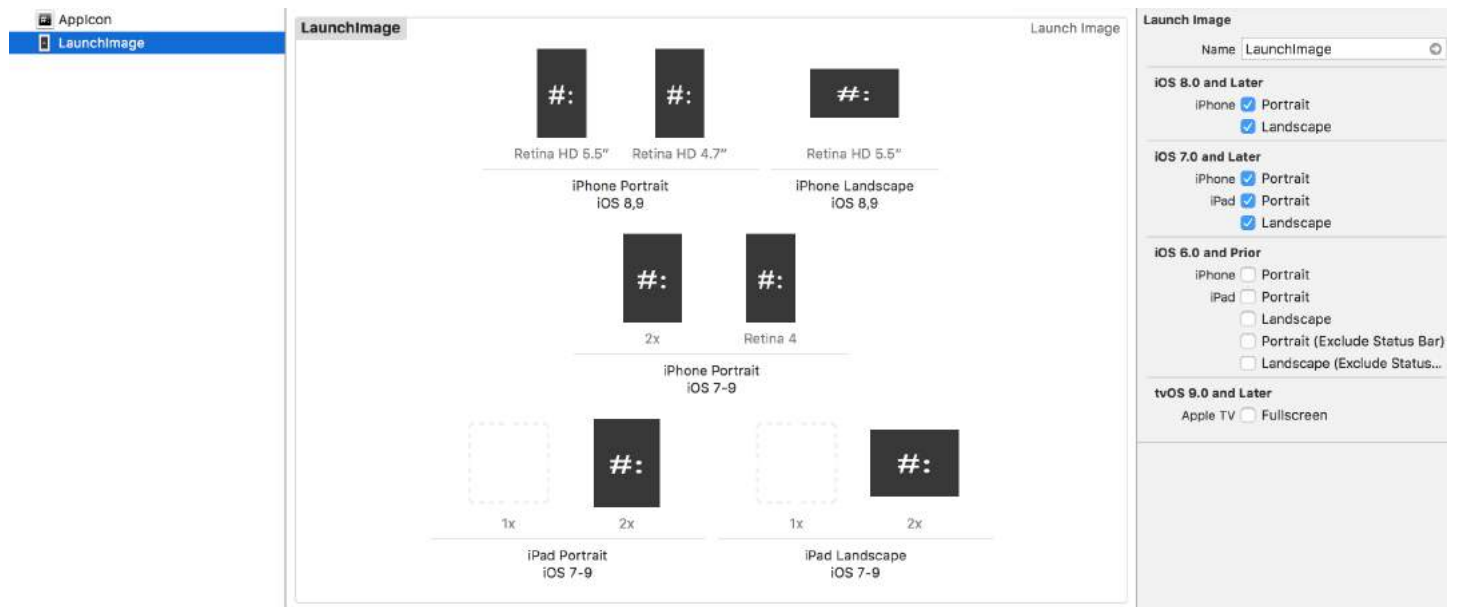
Here are sizes of all launch images:

```
Retina HD 5.5" iPhone Portrait - iPhone (6, 6S, 7)Plus - 1242x2208px
Retina HD 4.7" iPhone Portrait - iPhone 6, 6S, 7 - 750x1334px
Retina HD 5.5" iPhone Landscape - iPhone (6, 6S, 7)Plus - 2208x1242px
2x iPhone Portrait - (3.5") iPhone 4S - 640x960px
Retina 4 iPhone Portrait - (4") iPhone 5, 5S, 5C, iPod Touch, SE - 640x1136px
2x iPad Portrait - All Retina iPads - 1536x2048px
2x iPad Landscape - All Retina iPads - 2048x1536px
```

**Notes:**

1 **non-retina iPads:** I left blank `1x iPad Portrait` and `Landscape` because non-retina iPads will use 2x launch images by scaling

2 **12.9" iPad Pro**: there is no square for this iPad because this iPad will also use `2x iPad` images by scaling them

3 **Retina HD 5.5":** iPads should have `1920x1080px` for portrait and `1080x1920px` for landscape but Xcode will give waring and launch image will not be shown on those devices

4 **SplitView:** as we are using `LaunchImage Asset` instead of `LaunchScreen XIB`, our app will not support `SplitView` on iPads and landscape 5.5" iPhones

5 **Reinstall:** if our app is already installed on device and we try to run with these newly added launch image assets, then sometimes device will not show launch images while launching app. In this case just delete app from device, clean+build project and run it, it'll show new launch images

# Section 201.2: App Icon using Image Assets

Whenever we create a new project in Xcode for our new app, it gives us various in-built classes, targets, tests, plist file, etc. Similarly it also gives us as `Assets.xcassets` file, which manages all the image assets in our project.

This is how this file looks like in file navigator:
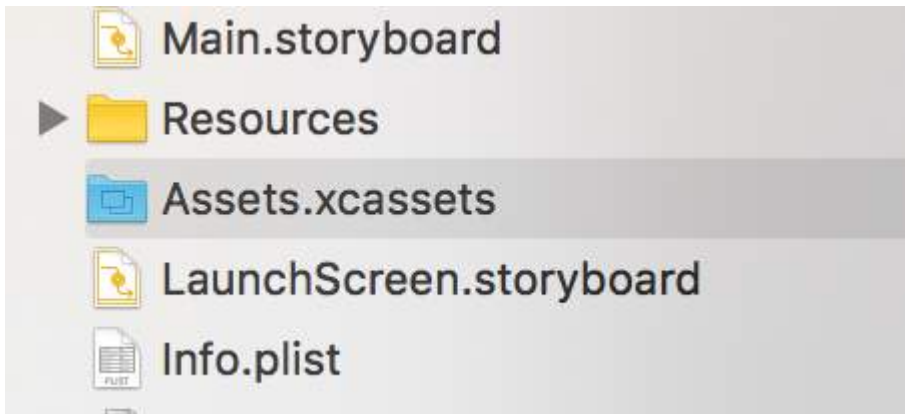
If we click it, it'll look like this:
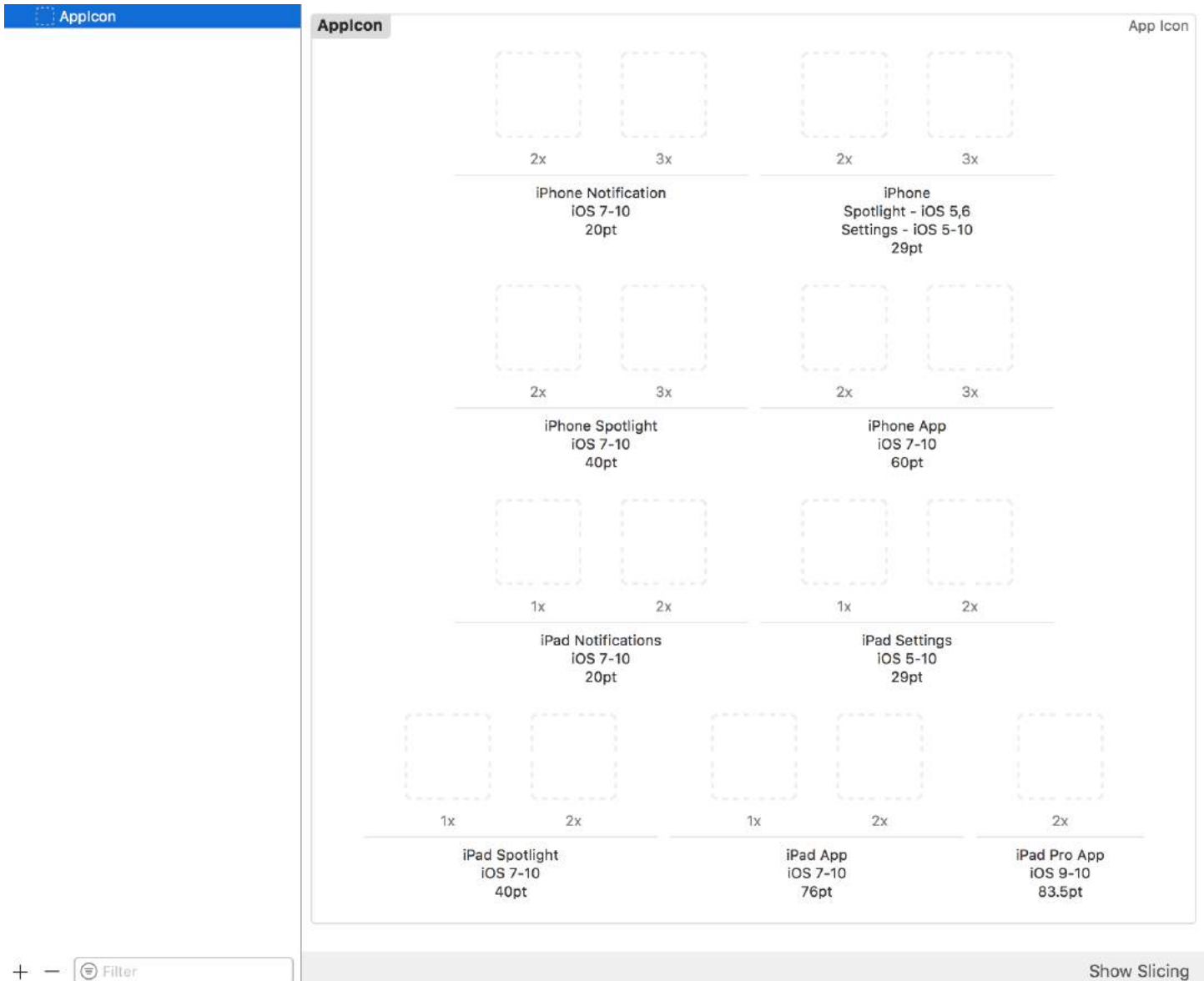


As I said, `AppIcon` asset is already created for us.

We just have to **drag and drop** respective image on each empty square block. Each black will tell us what size that image should be, it's written just below it.

After dragging and dropping all the images in all the squares, it'll look like this:
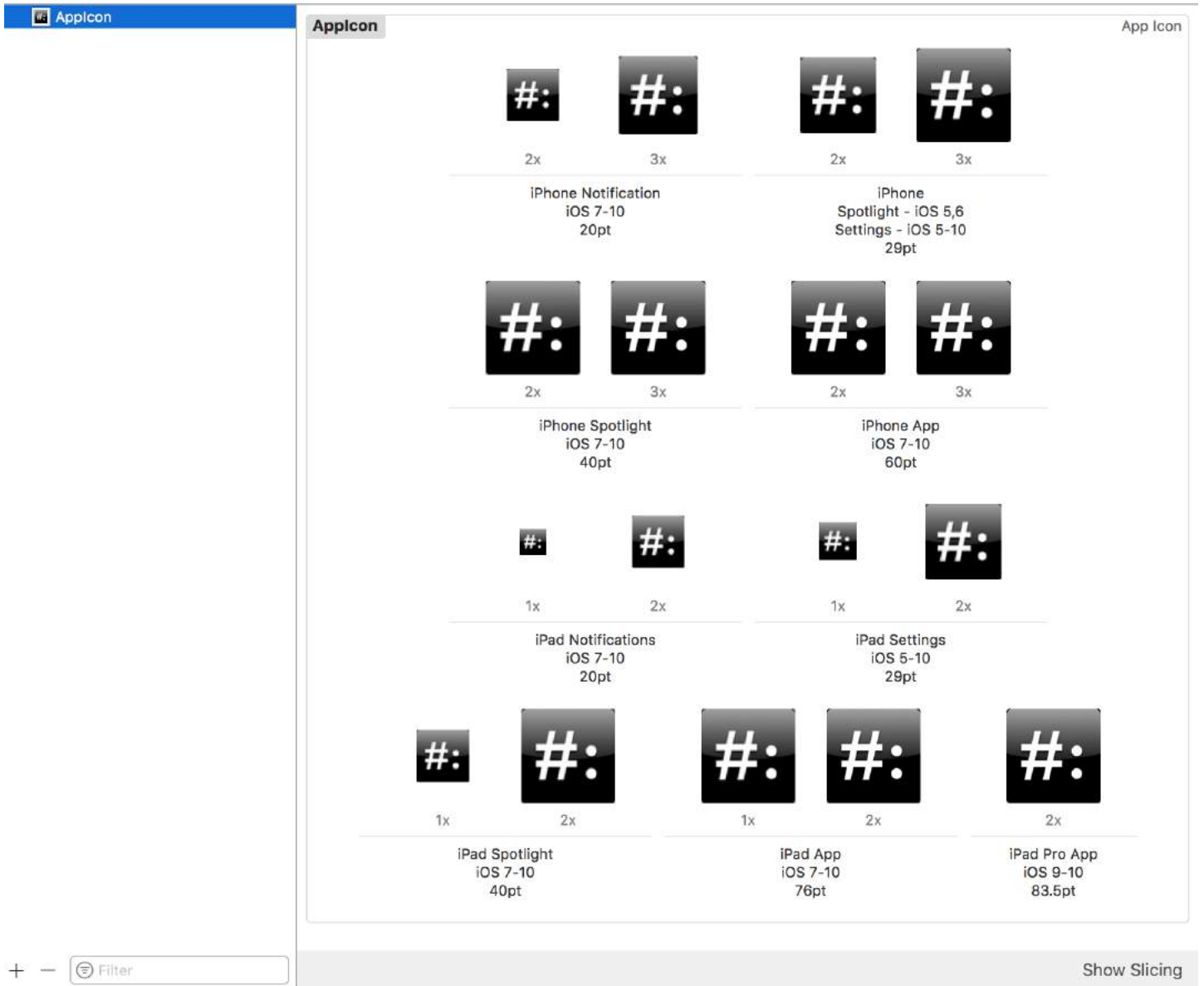
We can change the devices setting also for icon assets in Utilities -> Attributes Inspector as:

## App Icon

| | |
|---|---|
| Name | AppIcon |

| | |
|---|---|
| iPhone | iOS 7.0 and Later |
| iPad | iOS 7.0 and Later |
| CarPlay | ☐ All |
| Apple Watch | None |
| Mac | ☐ All |
| | ☐ iOS icon is pre-rendered |
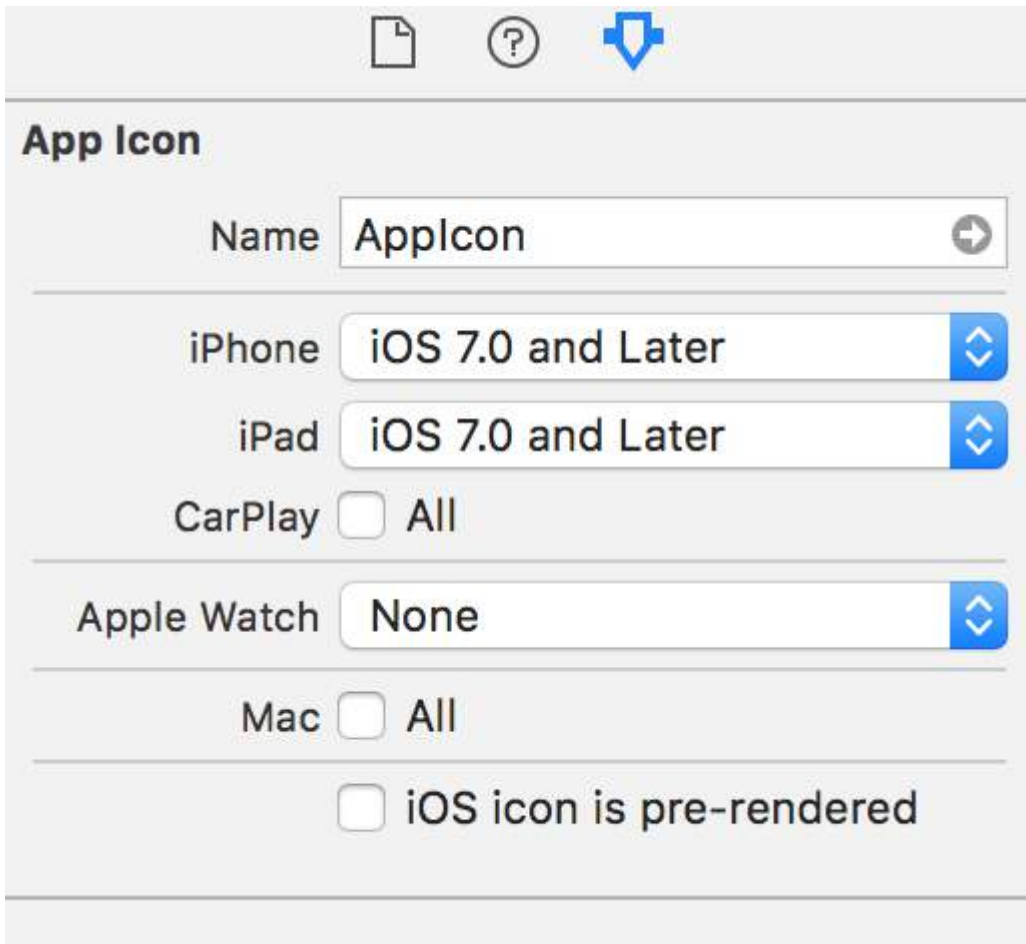
Once we finished this, just run an app and we'll be having nice icon to app as this:
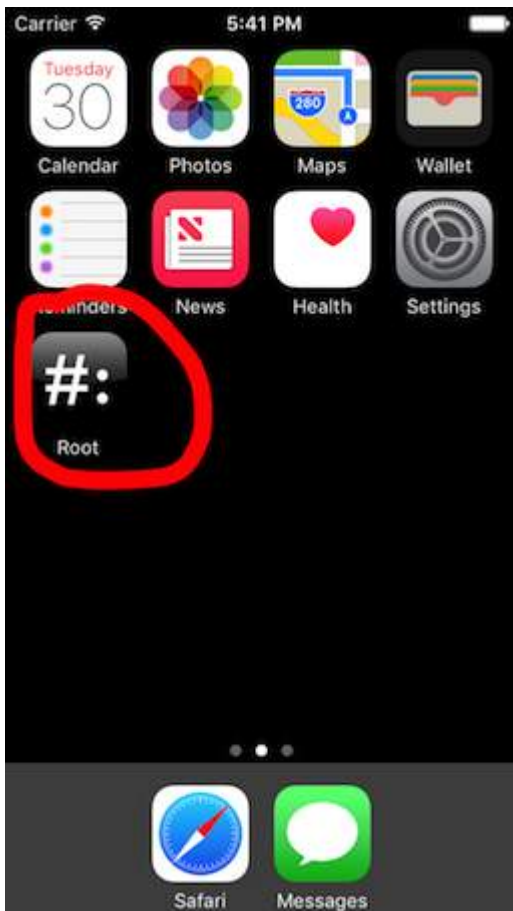


It is there by default, but if it's not then make sure this settings is as in Target->General settings:

**▼ App Icons and Launch Images**

App Icons Source    AppIcon

# Chapter 202: Runtime in Objective-C

## Section 202.1: Using Associated Objects

Associated objects are useful when you want to add functionality to existing classes which requires holding state.

For example, adding a activity indicator to every UIView:

*Objective-C Implementation*

```objc
#import <objc/runtime.h>

static char ActivityIndicatorKey;

@implementation UIView (ActivityIndicator)

- (UIActivityIndicatorView *)activityIndicator {
    return (UIActivityIndicatorView *)objc_getAssociatedObject(self, &ActivityIndicatorKey);
}

- (void)setActivityIndicator: (UIActivityIndicatorView *)activityIndicator {
    objc_setAssociatedObject(self, &ActivityIndicatorKey, activityIndicator,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (void)showActivityIndicator {
    UIActivityIndicatorView *activityIndicator = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle: UIActivityIndicatorViewStyleGray];

    [self setActivityIndicator:activityIndicator];

    activityIndicator.center = self.center;
    activityIndicator.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
UIViewAutoresizingFlexibleLeftMargin | UIViewAutoresizingFlexibleRightMargin |
UIViewAutoresizingFlexibleBottomMargin;

    [activityIndicator startAnimating];

    [self addSubview: activityIndicator];
}

- (void)hideActivityIndicator {
    UIActivityIndicatorView * activityIndicator = [self activityIndicator];

    if (activityIndicator != nil) {
        [[self activityIndicator] removeFromSuperview];
    }
}

@end
```

You can also access the Objective-C runtime through Swift:

*Swift Code*

```swift
extension UIView {
    private struct AssociatedKeys {
        static var activityIndicator = "UIView.ActivityIndicatorView"
    }
```

```
    private var activityIndicatorView: UIActivityIndicatorView? {
        get {
            return objc_getAssociatedObject(self, &AssociatedKeys.activityIndicator) as?
UIActivityIndicatorView
        }
        set (activityIndicatorView) {
            objc_setAssociatedObject(self, &AssociatedKeys.activityIndicator,
activityIndicatorView, .OBJC_ASSOCIATION_RETAIN_NONATOMIC)
        }
    }

     func showActivityIndicator() {
        activityIndicatorView = UIActivityIndicatorView(activityIndicatorStyle: .gray)
        activityIndicatorView.center = center
        activityIndicatorView.autoresizingMask = [.flexibleLeftMargin, .flexibleRightMargin,
.flexibleTopMargin, .flexibleBottomMargin]

        activityIndicatorView.startAnimating()

        addSubview(activityIndicatorView)
    }

    func hideActivityIndicator() {
        activityIndicatorView.removeFromSuperview()
    }
}
```

# Chapter 203: ModelPresentationStyles

Modal Presentation styles are used when you are transitioning from one view controller to another. There are 2 ways of achieving this customization. One is through code and another through Interface Builder(using segues). This effect is achieved by setting `modalPresentationStyle` variable to an instance of `UIModalPresentationStyle` enum. `modalPresentationStyle` property is a class variable of `UIViewController` and is used to specify how a `ViewController` is presented on screen.

## Section 203.1: Exploring ModalPresentationStyle using Interface Builder

This will be a very basic app which will illustrate different `ModalpresentationStyle` in iOS. According to documentation found <u>here</u>, There are 9 different values for `UIModalPresentationStyle` which are as follows,

1. `fullScreen`
2. `pageSheet`
3. `formSheet`
4. `currentContext`
5. `custom`
6. `overFullScreen`
7. `overCurrentContext`
8. `popover`
9. `none`

To setup a project, just create a normal iOS project and add 2 `ViewControllers`. Put a `UIButton` in you initial `ViewController` and connect it to 2nd `ViewController` via a `Target -> Action` mechanism. To distinguish both `ViewControllers`, set background property of `UIView` in `ViewController` some other color. If all goes well, your Interface Builder should look something this,
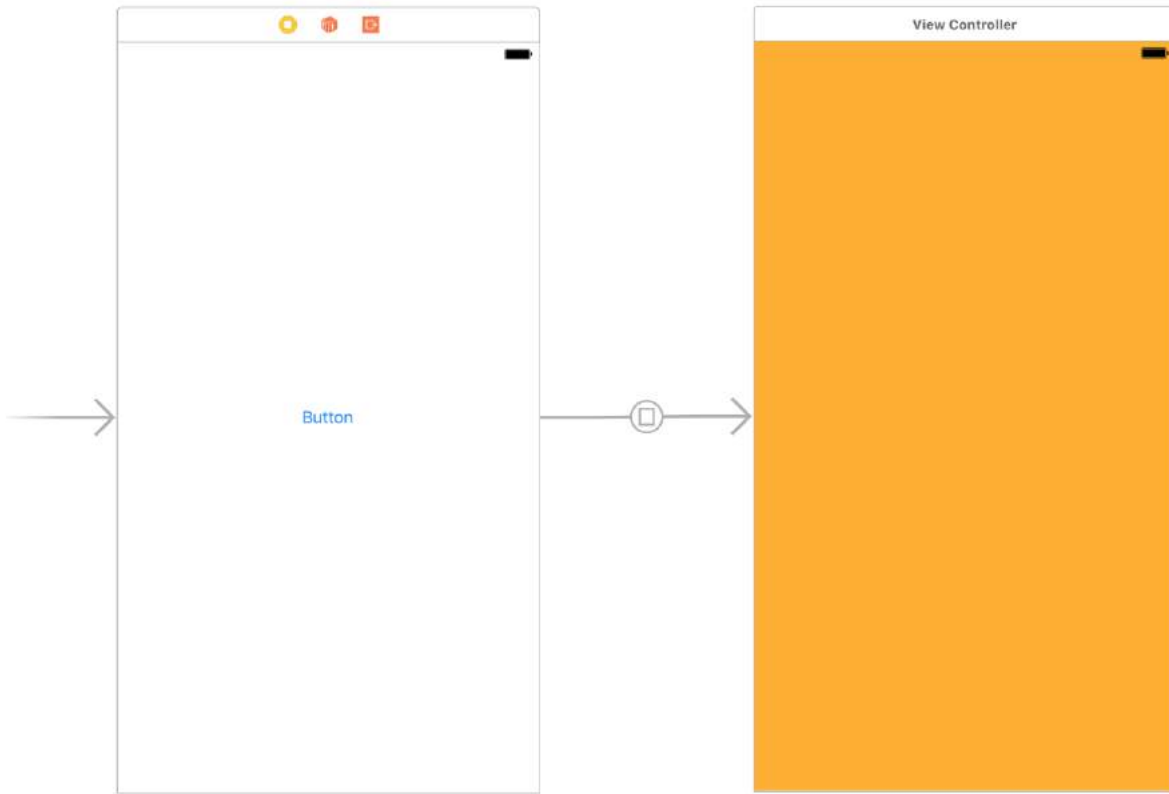
Make sure you build this project and run it on **iPad** (For details on why iPad, refer to Remarks section). Once you are done setting up your project, select the segue and go to the `attributes inspector`. You should be able to see
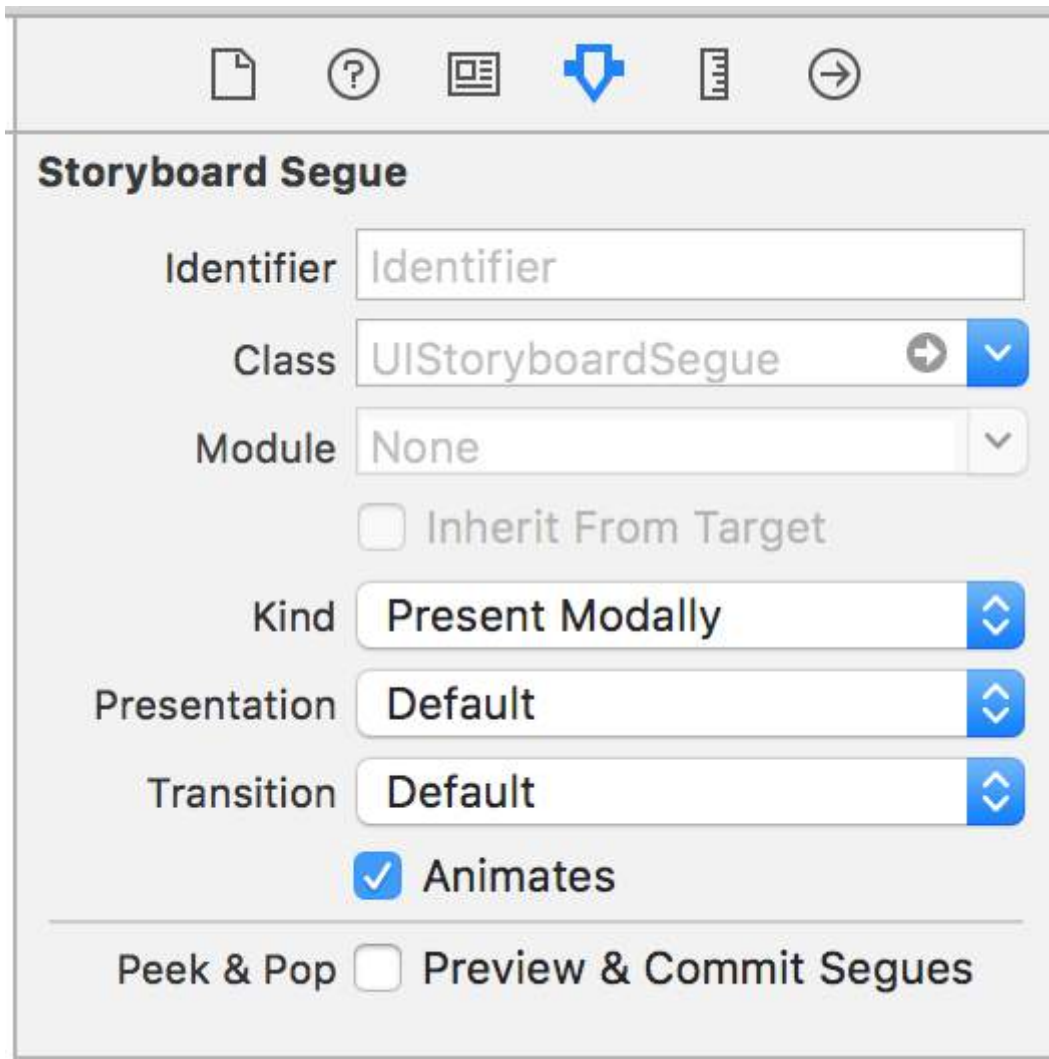
something like this,

Set the kind property to `Present Modally`.

Now, we won't see all of the effects in this example as some of them requires little bit of code.

Let's start with `fullscreen`. This effect is selected by default when you select `Present Modally` in `Kind` tab. When you build and run, the 2nd `ViewController` would occupy the full screen of you iPad.

Next is pageSheet. You can select this option from `Presentation` tab. In this option, when device is in portrait

mode, the 2nd `ViewController` is similar to full screen but in landscape mode, 2nd `ViewController` is much narrow the device width. Also, any content not covered by 2nd `ViewController` will be dimmed.
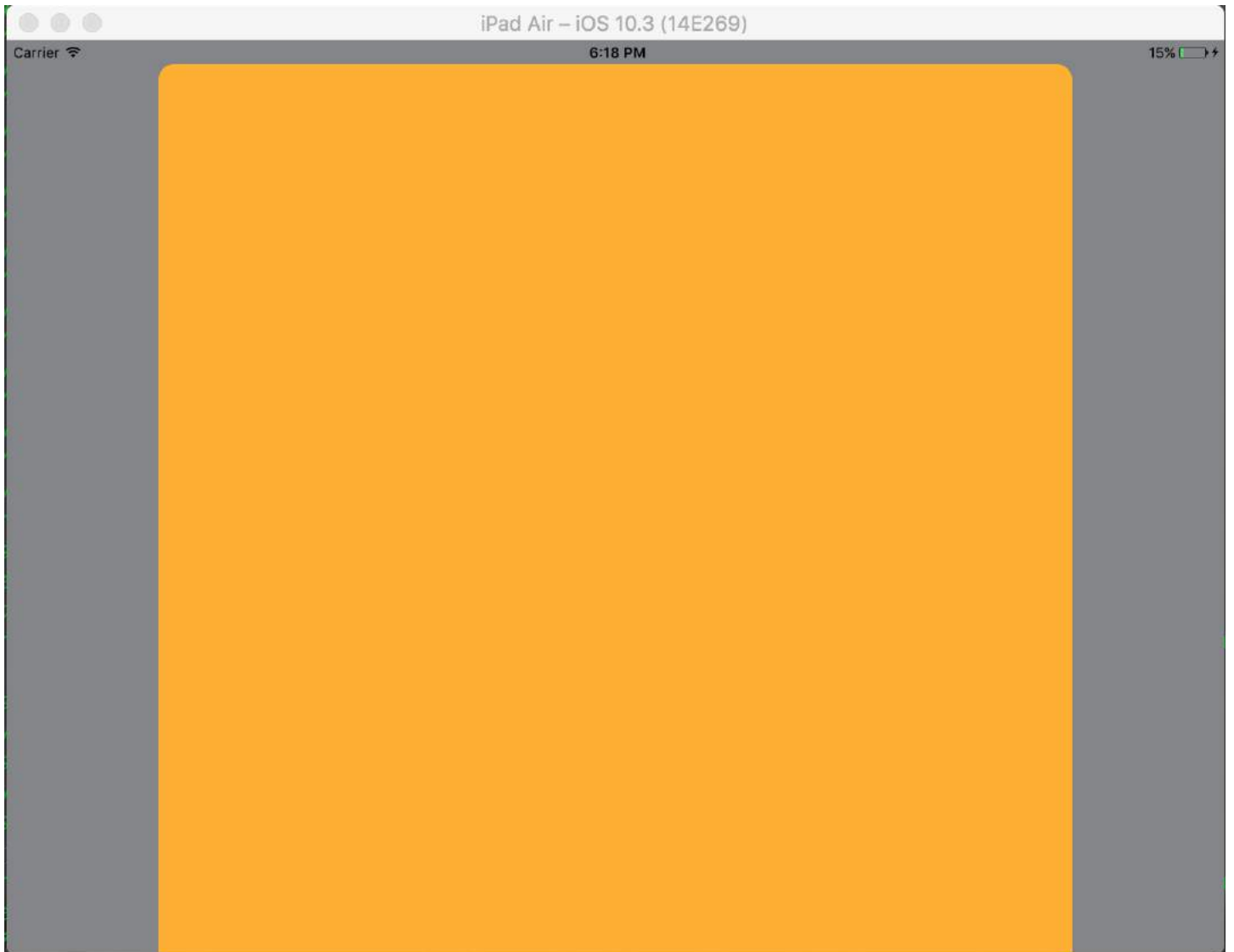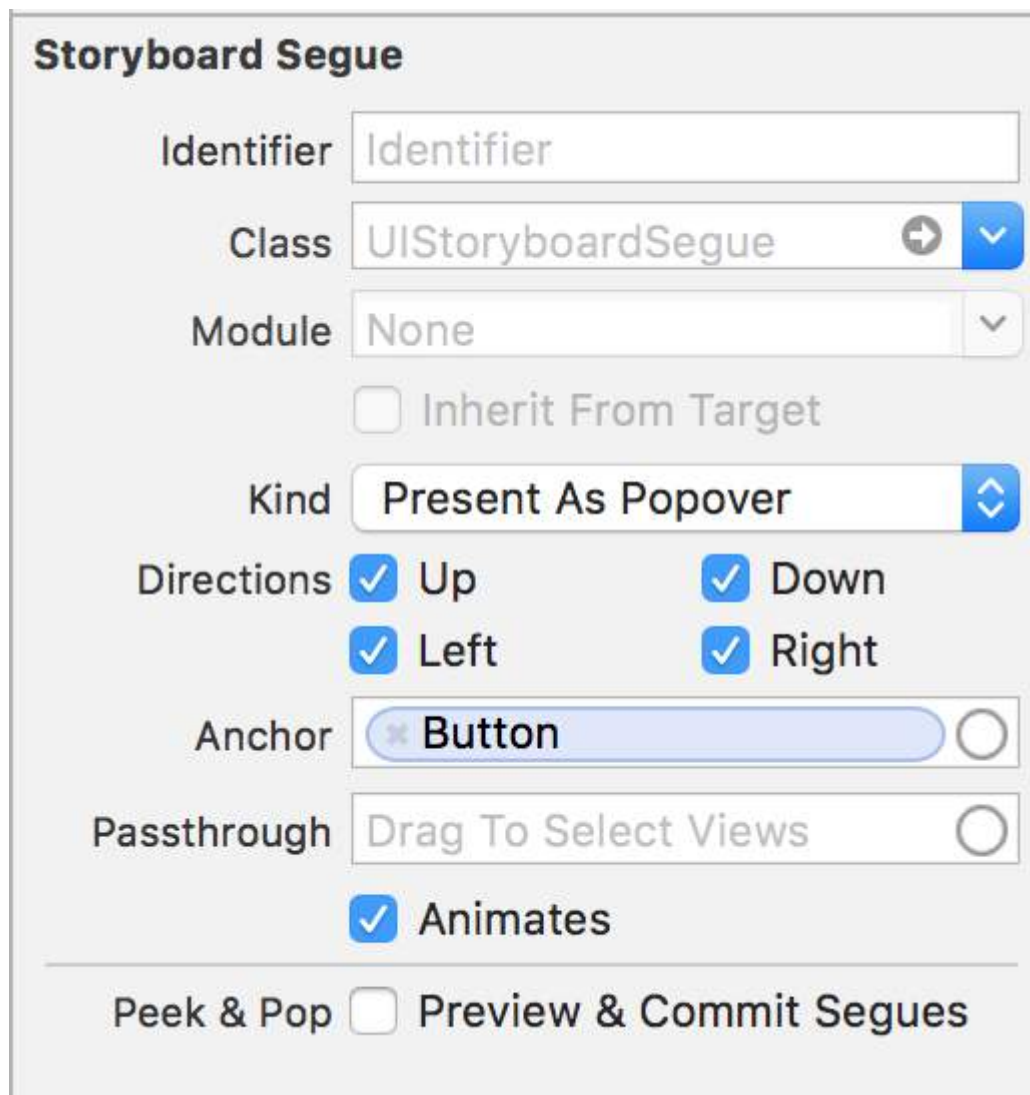


For `formSheet` style, the 2nd `ViewController` is placed in center of device and the size is smaller to that of device. Also when device is in landscape mode and keyboard is visible position of view is adjusted upwards to show the `ViewController`.
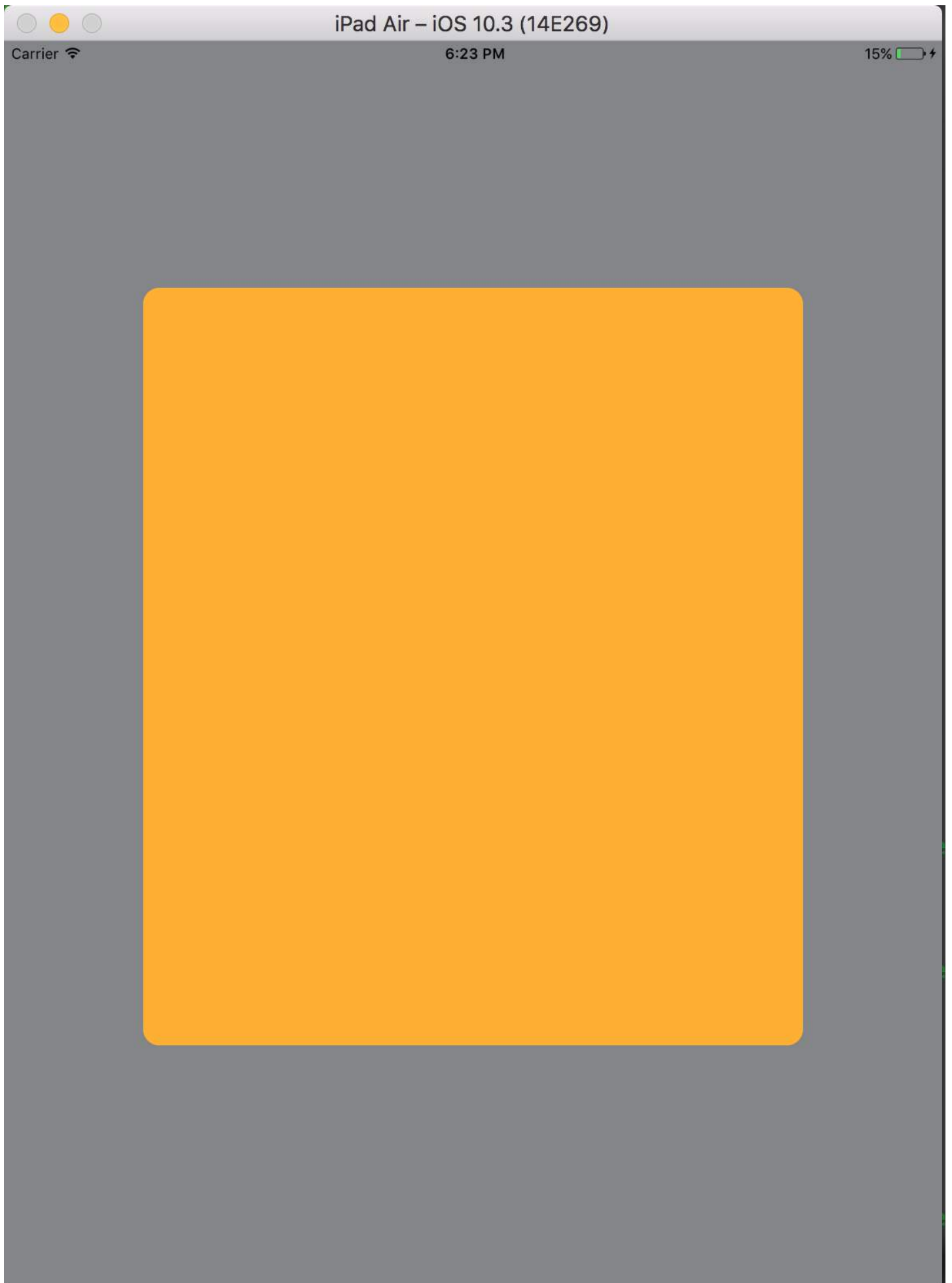
Last style which we are going to try is popover. To select this style, select `Present` `as` `Popover` in `Kind` tab. The 2nd

ViewController is presented as a small popover(size can be set). The background content is dimmed. Any tap outside the popover would dismiss the popover. Your `Attributes Inspector` should look something like this,



Anchor is the UI element to which you want your popover arrow to point. `Directions` are the directions you allow your popover `Anchor` to point in.

Button
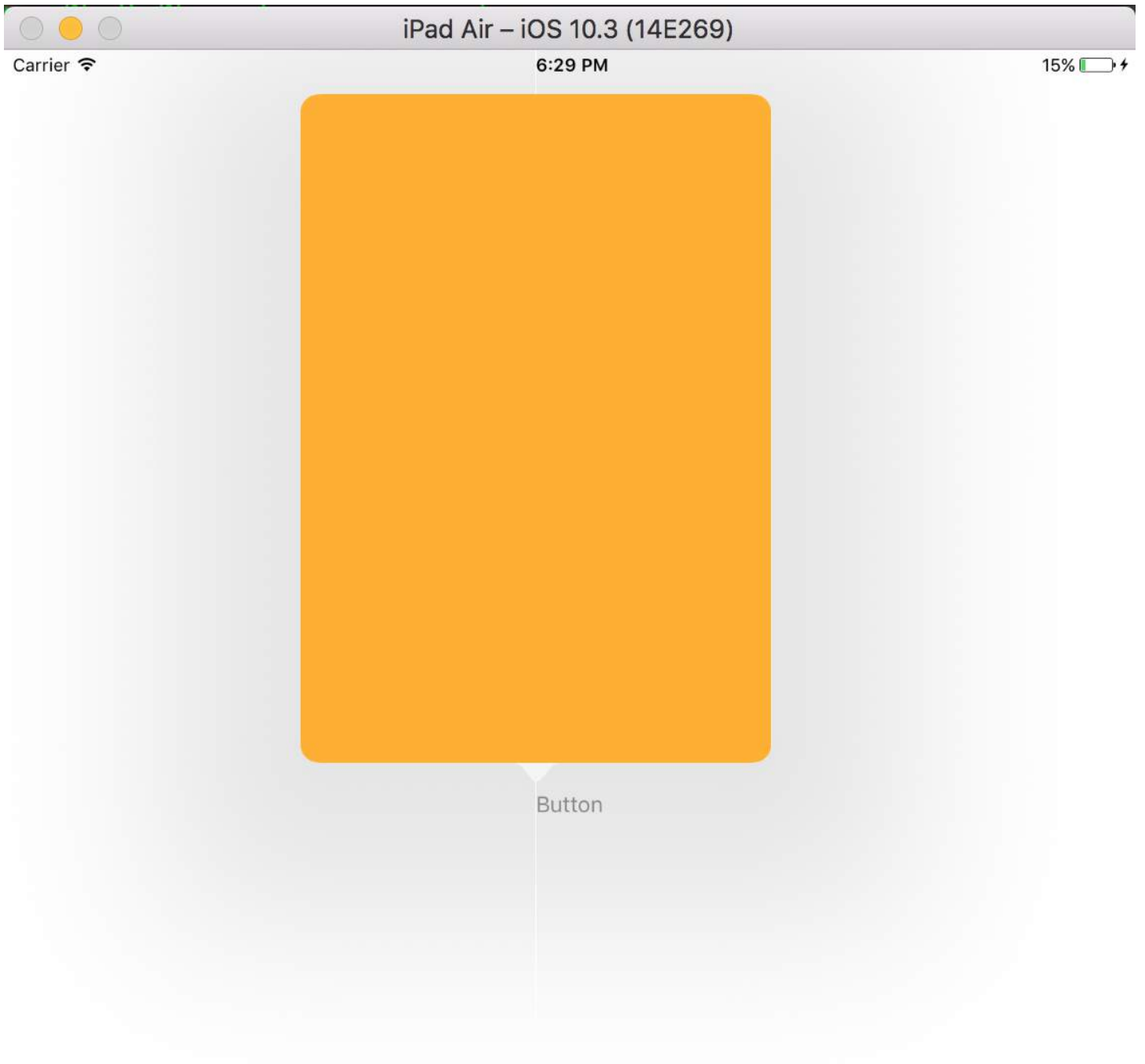
There more then these basic Modal Presentation Styles but they are little complicated to achieve and require some

code. More details can be found in the Apple Documentation.

# Chapter 204: CydiaSubstrate tweak

Learn how to create cydia substrate tweaks for jailbroken iPhones.

Those tweaks will enable you to modify the operating system's behavior to act the way you would like it to.

## Section 204.1: Create new tweak using Theos

**Use nic to create a new project**

Enter this command in your terminal

```
$THEOS/bin/nic.pl

NIC 2.0 - New Instance Creator
------------------------------
  [1.] iphone/activator_event
  [2.] iphone/application_modern
  [3.] iphone/cydget
  [4.] iphone/flipswitch_switch
  [5.] iphone/framework
  [6.] iphone/ios7_notification_center_widget
  [7.] iphone/library
  [8.] iphone/notification_center_widget
  [9.] iphone/preference_bundle_modern
  [10.] iphone/tool
  [11.] iphone/tweak
  [12.] iphone/xpc_service
Choose a Template (required):
```

Choose template `[11.] iphone/tweak`

Fill in the details and you will get the following files created:

```
-rw-r--r--@  1 gkpln3  staff    214B Jun 12 15:09 Makefile
-rw-r--r--@  1 gkpln3  staff     89B Jun 11 22:58 TorchonFocus.plist
-rw-r--r--   1 gkpln3  staff    2.7K Jun 12 16:10 Tweak.xm
-rw-r--r--   1 gkpln3  staff    224B Jun 11 16:17 control
drwxr-xr-x   3 gkpln3  staff    102B Jun 11 16:18 obj
drwxr-xr-x  16 gkpln3  staff    544B Jun 12 16:12 packages
```

**Override iOS screenshots saving method**

open the `Tweak.xm` file using your favorite code editor.

hook to a certain method from the operating system.

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    %orig;
    NSLog(@"saveScreenshot: is called");
}
%end
```

Note you can choose wether or not the original function should be called, for example:

```
%hook SBScreenShotter
- (void)saveScreenshot:(BOOL)screenshot
{
    NSLog(@"saveScreenshot: is called");
}
%end
```

will override the function without calling the original one, thus casing screenshots not being saved.

# Chapter 205: Create a video from images

Create a video from images using AVFoundation

## Section 205.1: Create Video from UIImages

First of all you need to create `AVAssetWriter`

```
NSError *error = nil;
NSURL *outputURL = <#NSURL object representing the URL where you want to save the video#>;
AVAssetWriter *assetWriter = [AVAssetWriter assetWriterWithURL:outputURL
fileType:AVFileTypeQuickTimeMovie error:&error];
if (!assetWriter) {
    // handle error
}
```

`AVAssetWriter` needs at least one asset writer input.

```
 NSDictionary *writerInputParams = [NSDictionary dictionaryWithObjectsAndKeys:
                                       AVVideoCodecH264, AVVideoCodecKey,
                                       [NSNumber numberWithInt:renderSize.width],
AVVideoWidthKey,
                                       [NSNumber numberWithInt:renderSize.height],
AVVideoHeightKey,
                                       AVVideoScalingModeResizeAspectFill,
AVVideoScalingModeKey,
                                       nil];

 AVAssetWriterInput *assetWriterInput = [AVAssetWriterInput
assetWriterInputWithMediaType:AVMediaTypeVideo outputSettings:writerInputParams];
 if ([assetWriter canAddInput:assetWriterInput]) {
     [assetWriter addInput:assetWriterInput];
 } else {
     // show error message
 }
```

To append `CVPixelBufferRef`'s to `AVAssetWriterInput` we need to create
`AVAssetWriterInputPixelBufferAdaptor`

```
NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                              [NSNumber numberWithUnsignedInt:kCVPixelFormatType_32ARGB],
(NSString*)kCVPixelBufferPixelFormatTypeKey,
                              [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey,
                              [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                              nil];
AVAssetWriterInputPixelBufferAdaptor *writerAdaptor = [AVAssetWriterInputPixelBufferAdaptor
assetWriterInputPixelBufferAdaptorWithAssetWriterInput:assetWriterInput
sourcePixelBufferAttributes:attributes];
```

Now we can start writing

```
[assetWriter startWriting];
[assetWriter startSessionAtSourceTime:kCMTimeZero];
[assetWriterInput requestMediaDataWhenReadyOnQueue:exportingQueue usingBlock:^{
    for (int i = 0; i < images.count; ++i) {
        while (![assetWriterInput isReadyForMoreMediaData]) {
```

```
            [NSThread sleepForTimeInterval:0.01];
            // can check for attempts not to create an infinite loop
        }

        UIImage *uIImage = images[i];

        CVPixelBufferRef buffer = NULL;
        CVReturn err = PixelBufferCreateFromImage(uIImage.CGImage, &buffer);
        if (err) {
            // handle error
        }

        // frame duration is duration of single image in seconds
        CMTime presentationTime = CMTimeMakeWithSeconds(i * frameDuration, 1000000);

        [writerAdaptor appendPixelBuffer:buffer withPresentationTime:presentationTime];

        CVPixelBufferRelease(buffer);
    }

    [assetWriterInput markAsFinished];
    [assetWriter finishWritingWithCompletionHandler:^{
        if (assetWriter.error) {
            // show error message
        } else {
            // outputURL
        }
    }];
}];
```

Here is a function to get `CVPixelBufferRef` from `CGImageRef`

```
CVReturn PixelBufferCreateFromImage(CGImageRef imageRef, CVPixelBufferRef *outBuffer) {
    CIContext *context = [CIContext context];
    CIImage *ciImage = [CIImage imageWithCGImage:imageRef];

    NSDictionary *attributes = [NSDictionary dictionaryWithObjectsAndKeys:
                                    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGBitmapContextCompatibilityKey,
                                    [NSNumber numberWithBool:YES], (NSString
*)kCVPixelBufferCGImageCompatibilityKey
                                    ,nil];

    CVReturn err = CVPixelBufferCreate(kCFAllocatorDefault, CGImageGetWidth(imageRef),
CGImageGetHeight(imageRef), kCVPixelFormatType_32ARGB, (__bridge CFDictionaryRef
_Nullable)(attributes), outBuffer);
    if (err) {
        return err;
    }

    if (outBuffer) {
        [context render:ciImage toCVPixelBuffer:*outBuffer];
    }

    return kCVReturnSuccess;
}
```

# Chapter 206: Codable

[Codable](#) is added with Xcode 9, iOS 11 and Swift 4. Codable is used to make your data types encodable and decodable for compatibility with external representations such as JSON.

Codable use to support both encoding and decoding, declare conformance to Codable, which combines the Encodable and Decodable protocols. This process is known as making your types codable.

## Section 206.1: Use of Codable with JSONEncoder and JSONDecoder in Swift 4

Let's Take an Example with Structure of Movie, here we have defined the structure as Codable. So, We can encode and decode it easily.

```
struct Movie: Codable {
    enum MovieGenere: String, Codable {
        case horror, skifi, comedy, adventure, animation
    }

    var name : String
    var moviesGenere : [MovieGenere]
    var rating : Int
}
```

*We can create a object from movie like as:*

```
let upMovie = Movie(name: "Up", moviesGenere: [.comedy , .adventure, .animation], rating : 4)
```

The upMovie contains the name "Up" and it's movieGenere is comedy, adventure and animation witch contains 4 rating out of 5.

**Encode**

JSONEncoder is an object that encodes instances of a data type as JSON objects. JSONEncoder supports the Codable object.

```
// Encode data
let jsonEncoder = JSONEncoder()
do {
    let jsonData = try jsonEncoder.encode(upMovie)
    let jsonString = String(data: jsonData, encoding: .utf8)
    print("JSON String : " + jsonString!)
}
catch {
}
```

JSONEncoder will give us the JSON data which is used to retrieve JSON string.

*Output string will be like :*

```
{
  "name": "Up",
  "moviesGenere": [
    "comedy",
    "adventure",
    "animation"
```

```
    ],
    "rating": 4
}
```

**Decode**

JSONDecoder is an object that decodes instances of a data type from JSON objects. We can get the object back from the JSON string.

```
do {
    // Decode data to object

    let jsonDecoder = JSONDecoder()
    let upMovie = try jsonDecoder.decode(Movie.self, from: jsonData)
    print("Rating : \(upMovie.name)")
    print("Rating : \(upMovie.rating)")
}
catch {
}
```

By decoding the JSONData we will receive the Movie object back. So we can get all the values which is saved in that object.

*Output will be like:*

```
Name : Up
Rating : 4
```

# Chapter 207: Load images async

## Section 207.1: Easiest way

The most simple way to create this is to use [Alamofire](#) and its [UIImageViewExtension](#). What we need is a tableview with a cell that has an imageView in it and lets call it `imageView`.

In the cellForRowAt: function of the tableView we would download the image and set in the following way:

```
let url = URL(string: "https://httpbin.org/image/png")!
let placeholderImage = UIImage(named: "placeholder")!

imageView.af_setImage(withURL: url, placeholderImage: placeholderImage)
```

The url should point to the image that you want to download and the placeHolder image should be a stored image. We then call the `af_setImage` method on the `imageView` which downloads the image at the given url and during the download the placeholder image will be shown. As soon as the image is downloaded the requested image is displayed

## Section 207.2: Check that the cell is still visible after download

Sometimes the download takes longer than the cell is being displayed. In this case it can happen, that the downloaded image is shown in the wrong cell. To fix this we can not use the [UIImageView Extension](#).

We still will be using [Alamofire](#) however we will use the completion handler to display the image.

In this scenario we still need a tableView with a cell which has a imageView in it. In the cellForRowAt: method we would download the image with the following code:

```
let placeholderImage = UIImage(named: "placeholder")!
imageView.image = placeholderImage

let url = URL(string: "https://httpbin.org/image/png")!

Alamofire.request(url!, method: .get).responseImage { response in
    guard let image = response.result.value else { return }

    if let updateCell = tableView.cellForRow(at: indexPath) {
        updateCell.imageView.image = image
    }
}
```

In this example we first set the image to the placeholder image. Afterwards we download the image with the `request` method of [Alamofire](#). We pass the url as the first argument and since we just want to get the image we will use the `.get` HTTP method. Since we are downloading an image we want the response to be an image therefore we use the `.responseImage` method.

After the image has been downloaded the closure gets called and first of all we make sure that the downloaded image actually exists. Then we make sure that the cell is still visible by checking that the cellForRow(at: indexPath) doesn't return nil. If it does nothing happens, if it doesn't we assign the recently downloaded image.

This last if statement ensures that the cell is still visible if the user already scrolled over the cell the updateCell will be nil and the if statement returns nil. This helps us prevent displaying the wrong image in a cell.

# Chapter 208: Adding a Swift Bridging Header

## Section 208.1: How to create a Swift Bridging Header Manually

- Add a new file to Xcode (File > New > File), then select "Source" and click "Header File".

- Name your file "YourProjectName-Bridging-Header.h". Example: In my app Station, the file is named "Station-Bridging-Header".

- Create the file.

- Navigate to your project build settings and find the "Swift Compiler – Code Generation" section. You may find it faster to type in "Swift Compiler" into the search box to narrow down the results. Note: If you don't have a "Swift Compiler – Code Generation" section, this means you probably don't have any Swift classes added to your project yet. Add a Swift file, then try again.

- Next to "Objective-C Bridging Header" you will need to add the name/path of your header file. If your file resides in your project's root folder simply put the name of the header file there. Examples: "ProjectName/ProjectName-Bridging-Header.h" or simply "ProjectName-Bridging-Header.h".

- Open up your newly created bridging header and import your Objective-C classes using #import statements. Any class listed in this file will be able to be accessed from your swift classes.

## Section 208.2: Xcode create automatically

Add a new Swift file to your Xcode project. Name it as you please and you should get an alert box asking if you would like to create a bridging header. Note: If you don't receive a prompt to add a bridging header, you probably declined this message once before and you will have to add the header manually (see below)

# Chapter 209: Creating an App ID

## Section 209.1: Creating In-App Purchase Products

- When offering IAP within an an app, you must first add an entry for each individual purchase within iTunes Connect. If you've ever listed an app for sale in the store, it's a similar process and includes things like choosing a pricing tier for the purchase. When the user makes a purchase, the App Store handles the complex process of charging the user's iTunes account. There are a whole bunch of different types of IAP you can add:
  - **Consumable**: These can be bought more than once and can be used up. These are things such as extra lives, in-game currency, temporary power-ups, and the like.
  - **Non-Consumable**: Something that you buy once, and expect to have permanently such as extra levels and unlockable content.
  - **Non-Renewing Subscription**: Content that's available for a fixed period of time.
  - **Auto-Renewing Subscription**: A repeating subscription such as a monthly raywenderlich.com subscription.

You can only offer In-App Purchases for digital items, and not for physical goods or services. For more information about all of this, check out Apple's full documentation on Creating In-App Purchase Products. Now, while viewing your app's entry in iTunes Connect, click on the Features tab and then select In-App Purchases. To add a new IAP product, click the + to the right of In-App Purchases.
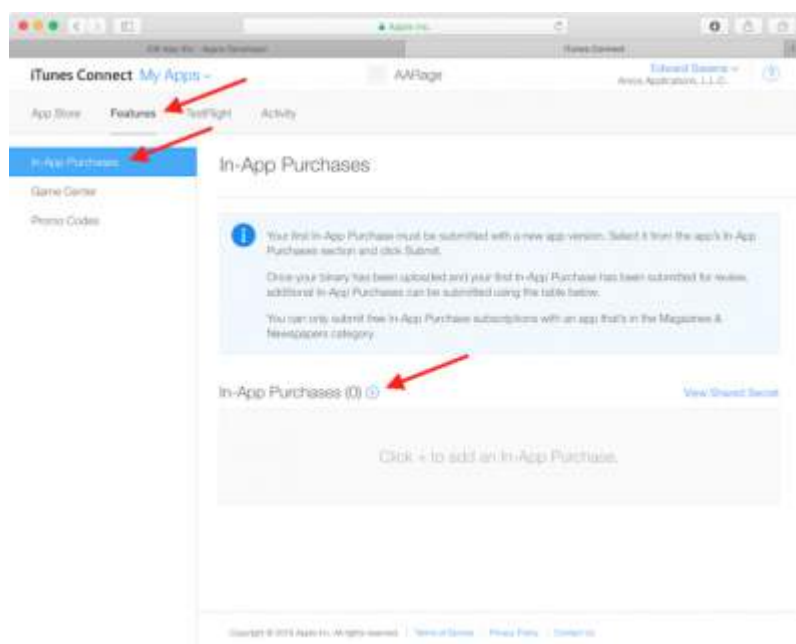


You will see the following dialog appear:

**Select the In-App Purchase you want to create.**

○ **Consumable**
A product that is used once, after which it becomes depleted and must be purchased again.

**Example:** Fish food for a fishing app.

○ **Non-Consumable**
A product that is purchased once and does not expire or decrease with use.

**Example:** Race track for a game app.

○ **Auto-Renewable Subscription**
A product that allows users to purchase dynamic content for a set period. This type of subscription renews automatically unless cancelled by the user.

**Example:** Monthly subscription for an app offering a streaming service.

○ **Non-Renewing Subscription**
A product that allows users to purchase a service with a limited duration. The content of this in-app purchase can be static. This type of subscription does not renew automatically.

**Example:** Annual subscription to a catalog of archived articles.

Learn more about In-App Purchases.    [ Cancel ]  [ Create ]

When a user purchases a rage comic in your app, you'll want them to always have access to it, so select Non-Consumable, and click Create. Next, fill out the details for the IAP as follows:

- **Reference Name**: A nickname identifying the IAP within iTunes Connect. This name does not appear anywhere in the app. The title of the comic you'll be unlocking with this purchase is **"Girlfriend of Drummer"**, so enter that here.
- **Product ID**: This is a unique string identifying the IAP. Usually it's best to start with the Bundle ID and then append a unique name specific to this purchasable item. For this tutorial, make sure you append "GirlfriendOfDrummerRage", as this will be used later within the app to look up the comic to unlock. So, for example: com.theNameYouPickedEarlier.Rage.GirlFriendOfDrummerRage.
- **Cleared for Sale**: Enables or disables the sale of the IAP. You want to enable it!
- **Price Tier**: The cost of the IAP. Choose Tier 1.

Now scroll down to the Localizations section and note that there is a default entry for English (U.S.). Enter "Girlfriend of Drummer" for both the Display Name and the Description. Click Save. Great! You've created your first IAP product.

There's one more step required before you can delve into some code. When testing in-app purchases in a development build of an app, Apple provides a test environment which allows you to 'purchase' your IAP products without creating financial transactions.

## Section 209.2: Creating a Sandbox User

In iTunes Connect, click iTunes Connect in the top left corner of the window to get back to the main menu. Select Users and Roles, then click the Sandbox Testers tab. Click + next to the "Tester" title.



Fill out the information and click Save when you're done. You can make up a first and last name for your test user, but the email address chosen must be a real email address as a verification will be sent to the address by Apple. Once you receive that email, be sure to click the link in it to verify your address. The email address you enter should also NOT already be associated with an Apple ID account. Hint: if you have a gmail account, you can simply use an address alias instead of having to create a brand new account

# Chapter 210: Swift: Changing the rootViewController in AppDelegate to present main or login/onboarding flow

It is often useful to present a first-run experience to new users of your App. This could be for any number of reasons, such as prompting them to sign in (if required for your situation), explaining how to use the App, or simply informing them of new features in an update (as Notes, Photos and Music do in iOS11).

## Section 210.1: Option 1: Swap the Root View Controller (Good)

There are benefits to switching the root view controller, although the transition options are limited to those supported by `UIViewAnimationOptions`, so depending on how you wish to transition between flows might mean you have to implement a custom transition - which can be cumbersome.

You can show the Onboarding flow by simply setting the `UIApplication.shared.keyWindow.rootViewController`

Dismissal is handled by utilizing `UIView.transition(with:)` and passing the transition style as a `UIViewAnimationOptions`, in this case Cross Dissolve. (Flips and Curls are also supported).

You also have to set the frame of the Main view before you transition back to it, as you're instantiating it for the first time.

```swift
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = UIApplication.shared.keyWindow, let onboardingViewController =
UIStoryboard(name: "Onboarding", bundle: nil).instantiateInitialViewController() as?
OnboardingViewController {
            onboardingViewController.delegate = self
            window.rootViewController = onboardingViewController
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow, let mainViewController = UIStoryboard(name:
"Main", bundle: nil).instantiateInitialViewController() {
            mainViewController.view.frame = window.bounds
            UIView.transition(with: window, duration: 0.5, options: .transitionCrossDissolve,
animations: {
                window.rootViewController = mainViewController
            }, completion: nil)
        }
    }
}
```

## Section 210.2: Option 2: Present Alternative Flow Modally (Better)

In the most straightforward implementation, the Onboarding flow can simply be presented in a modal context, since semantically the User is on a single journey.

> [Apple Human Interface Guidelines – Modality][1]:

> Consider creating a modal context only when it's critical to get someone's attention, when a task must be completed or abandoned to continue using the app, or to save important data.

Presenting modally allows the simple option of dismissal at the end of the journey, with little of the cruft of swapping controllers.

Custom transitions are also supported in the standard way, since this uses the `ViewController.present()` API:

```swift
// MARK: - Onboarding

extension AppDelegate {

    func showOnboarding() {
        if let window = window, let onboardingViewController = UIStoryboard(name: "Onboarding",
bundle: nil).instantiateInitialViewController() as? OnboardingViewController {
            onboardingViewController.delegate = self
            window.makeKeyAndVisible()
            window.rootViewController?.present(onboardingViewController, animated: false,
completion: nil)
        }
    }

    func hideOnboarding() {
        if let window = UIApplication.shared.keyWindow {
            window.rootViewController?.dismiss(animated: true, completion: nil)
        }
    }
}
```

# Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content, more changes can be sent to web@petercv.com for new content to be published or updated

| | |
|---|---|
| 4oby | Chapter 2 |
| Abhijit | Chapter 192 |
| abjurato | Chapter 149 |
| Adam Eberbach | Chapters 11 and 38 |
| Adam Preble | Chapter 27 |
| Adnan Aftab | Chapter 79 |
| Adriana Carelli | Chapters 151 and 46 |
| Ahmed Khalaf | Chapter 28 |
| AJ9 | Chapter 14 |
| ajmccall | Chapter 65 |
| Aju | Chapter 15 |
| Akilan Arasu | Chapter 2 |
| alaphao | Chapters 67 and 20 |
| Alex | Chapters 92 and 40 |
| Alex Kallam | Chapter 74 |
| Alex Koshy | Chapters 2, 14, 74, 34, 95 and 38 |
| Alex Rouse | Chapter 61 |
| Alexander Tkachenko | Chapter 64 |
| Ali Abbas | Chapter 178 |
| Ali Beadle | Chapter 1 |
| Ali Elsokary | Chapter 34 |
| Alistra | Chapter 89 |
| Alvin Abia | Chapter 122 |
| Amanpreet | Chapter 107 |
| amar | Chapters 67, 96 and 97 |
| Anand Nimje | Chapters 1, 74, 38, 40, 76 and 117 |
| Anatoliy | Chapter 1 |
| Andreas | Chapters 151, 25 and 45 |
| Andres Canella | Chapter 2 |
| Andres Kievsky | Chapter 14 |
| Andrii Chernenko | Chapter 2 |
| Anh Pham | Chapters 11 and 20 |
| Ankit chauhan | Chapter 190 |
| ApolloSoftware | Chapter 79 |
| Arefly | Chapters 5 and 96 |
| Ashish Kakkad | Chapters 107, 113 and 206 |
| Ashutosh Dave | Chapter 1 |
| askielboe | Chapter 142 |
| AstroCB | Chapter 2 |
| azimov | Chapters 47, 56 and 119 |
| backslash | Chapter 52 |
| Badal Shah | Chapters 67, 30 and 144 |
| balagurubaran | Chapter 77 |
| Bean | Chapters 67 and 53 |
| Bence Pattogato | Chapter 74 |
| BennX | Chapter 131 |
| bentford | Chapter 2 |
| Beto Caldas | Chapter 189 |
| beyowulf | Chapters 59, 55 and 57 |

# You may also like

**Android**
Notes for Professionals

1000+ pages
of professional hints and tricks

**C#**
Notes for Professionals

700+ pages
of professional hints and tricks

**CSS**
Notes for Professionals

200+ pages
of professional hints and tricks

**HTML5**
Notes for Professionals

100+ pages
of professional hints and tricks

**HTML5 Canvas**
Notes for Professionals

100+ pages
of professional hints and tricks

**JavaScript**
Notes for Professionals

400+ pages
of professional hints and tricks

**jQuery**
Notes for Professionals

50+ pages
of professional hints and tricks

**Objective-C**
Notes for Professionals

100+ pages
of professional hints and tricks

**Swift**
Notes for Professionals

200+ pages
of professional hints and tricks