MOBILE DEVELOPMENT

BY JEFF ALGERA

GENERAL ASSEMBLY

Table of Contents

Mobile Development

Welcome

Welcome to the Mobile Development Course at General Assembly. We're glad you've decided to learn with us for the next several weeks.

How to use this document

Use this GitBook as a reference before, during and after your class. Material will be referenced by chapter and section number by your instructors at General Assembly.

What this document covers

The virtual book you hold in your hands will cover iOS, Swift programming and publishing your next app on the iTunes App Store.

Git and GitHub

Overview

Our Mobile Development course will be using Git as our versioning control system. We will use GitHub as our interface to that system. This document will cover

1. Version control
2. The GitHub app
3. Common terms

Git is a tool software developers use to manage changes in code. Those changes may be your own, or may be made by members of your team.

November 30	fix navigating the page controller	Jeff Algera	162e7da224a799918970bec6b8775eeb80362ffb
November 30	User management	Jeff Algera	3279948c9c362b5905ba73e6aac88237095cdaba
November 21	Update from FB	Jeff Algera	a393c006ad955fd8b89198555de170f96dab0334
November 20	Linking FB friends	Jeff Algera	38244b656f5753d4a8018adc6f3f1f9e28ad180b
November 19	Loading data from FB	Jeff Algera	59472f4eabb9cb1dc6af4bea9450d205b7f753d7
November 19	Cleanup user manager	Jeff Algera	c103e4de342df61a6b1980de23f2715095fcafb2

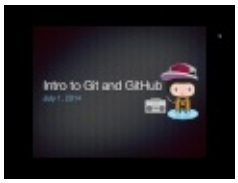
A history of changes made in a Git repository

GitHub is a company that provides a really excellent interface for managing Git repositories.

Secret lair of GitHub in San Francisco.

[Video] Introduction to Git and Github

What Git and GitHub do, why you should use them, and how to get started:



References

Git quick reference for beginners - http://www.dataschool.io/git-quick-reference-for-beginners/

GitHub - http://github.com

Git according to Wikipedia - http://en.wikipedia.org/wiki/Git_(software)

GitHub instructional videos published on YouTube - https://www.youtube.com/user/GitHubGuides

Version Control

Version control provides a number of benefits to the developer. Here are some of those benefits:

1. History
2. Version Control
3. Branching
4. A Distributed System

Benefit 1: History

Imagine working on a piece of code. Let's call this version 1:

```
var eureka = "The meaning of life is the following number"
var meaning_of_life = 42
print("Guys, I've figured it out.  \(eureka) is \(meaning_of_life)!")
```

Then after a few days, you change your mind and update the code. Let's call this version 2:

```
var eureka = "The meaning of life is the following number"
var meaning_of_life = 99999
print("Guys, I've figured it out.  \(eureka) is \(meaning_of_life)!")
```

Some time passes, you close Xcode, shut down your computer and then you realize:

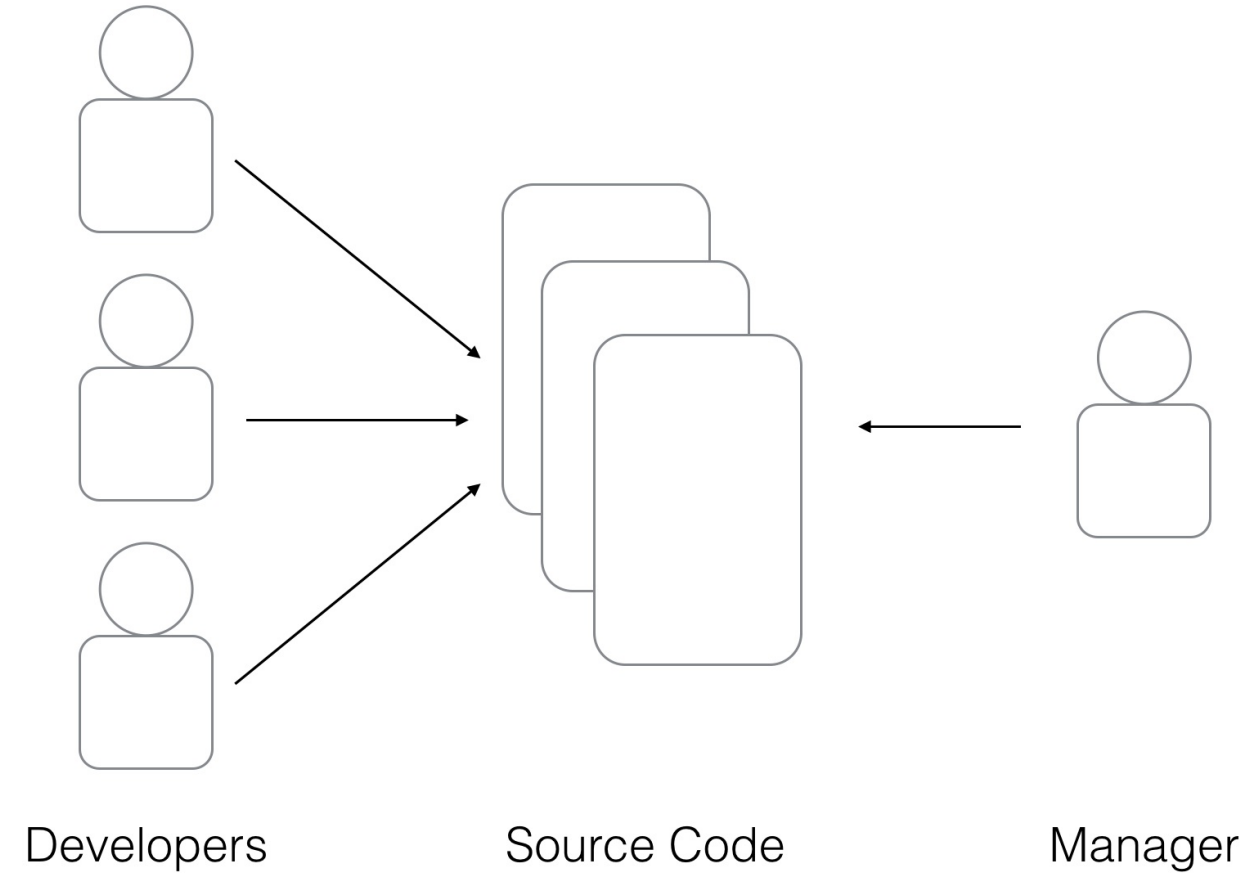
> The first version was right! What did I have?

With Git, you can have a historical record of all changes made in your code. You can then reference those changes at a later time.

```
var meaning_of_life = 42
```

Benefit 2: Working with a team

Now imagine you are working with a team of three programmers on an awesome new app. There is one Xcode project, three programmers and one manager.

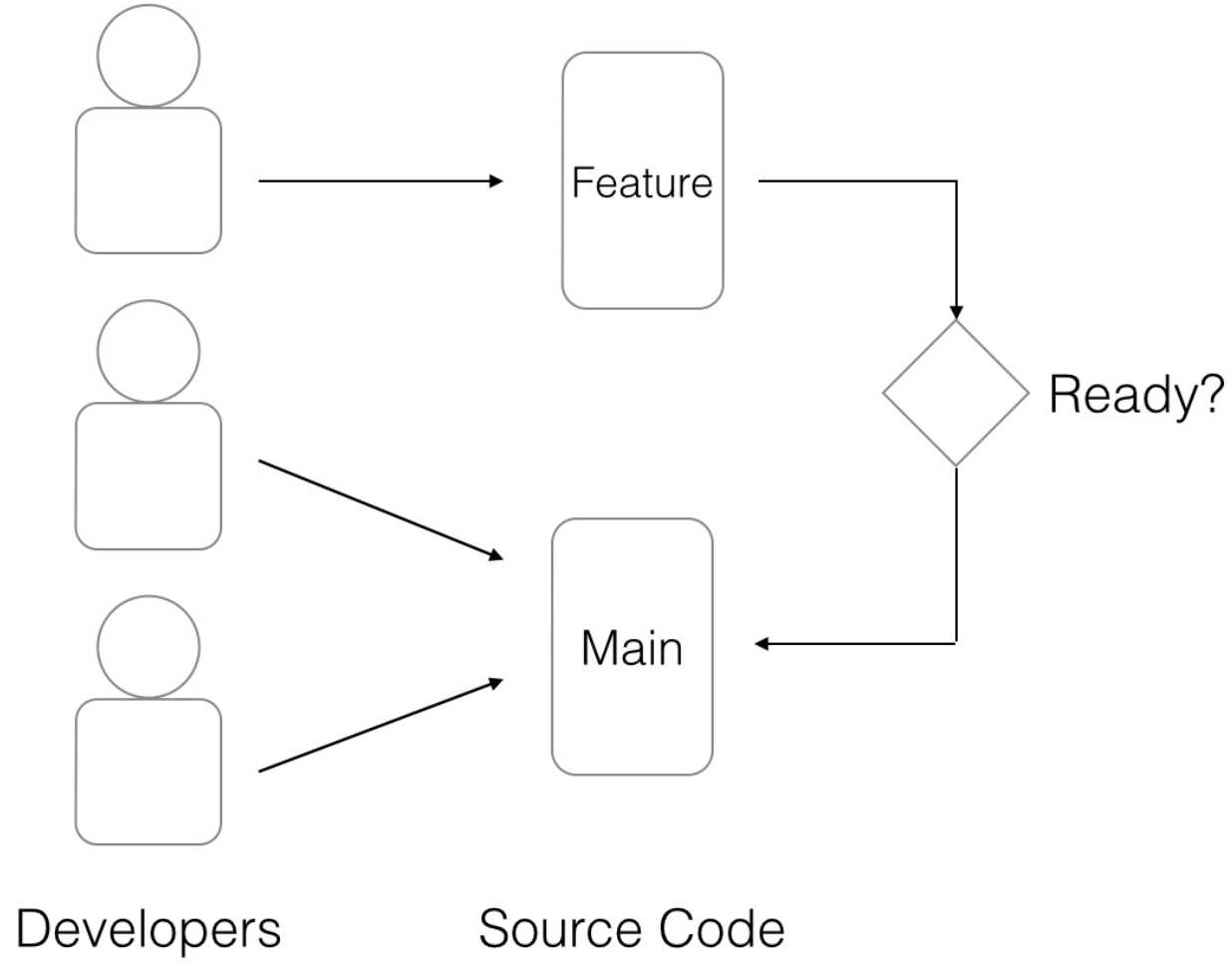


The benefits you get with a version control system, such as Git, in this scenario:

1. Historical record of what each programmer was working on and when
2. The ability to go back in time and view changes from a different day for any of the programmers on the team
3. Automatic merging of changes from each developer into one central project
4. One central project that a manager can download and send off to the client.

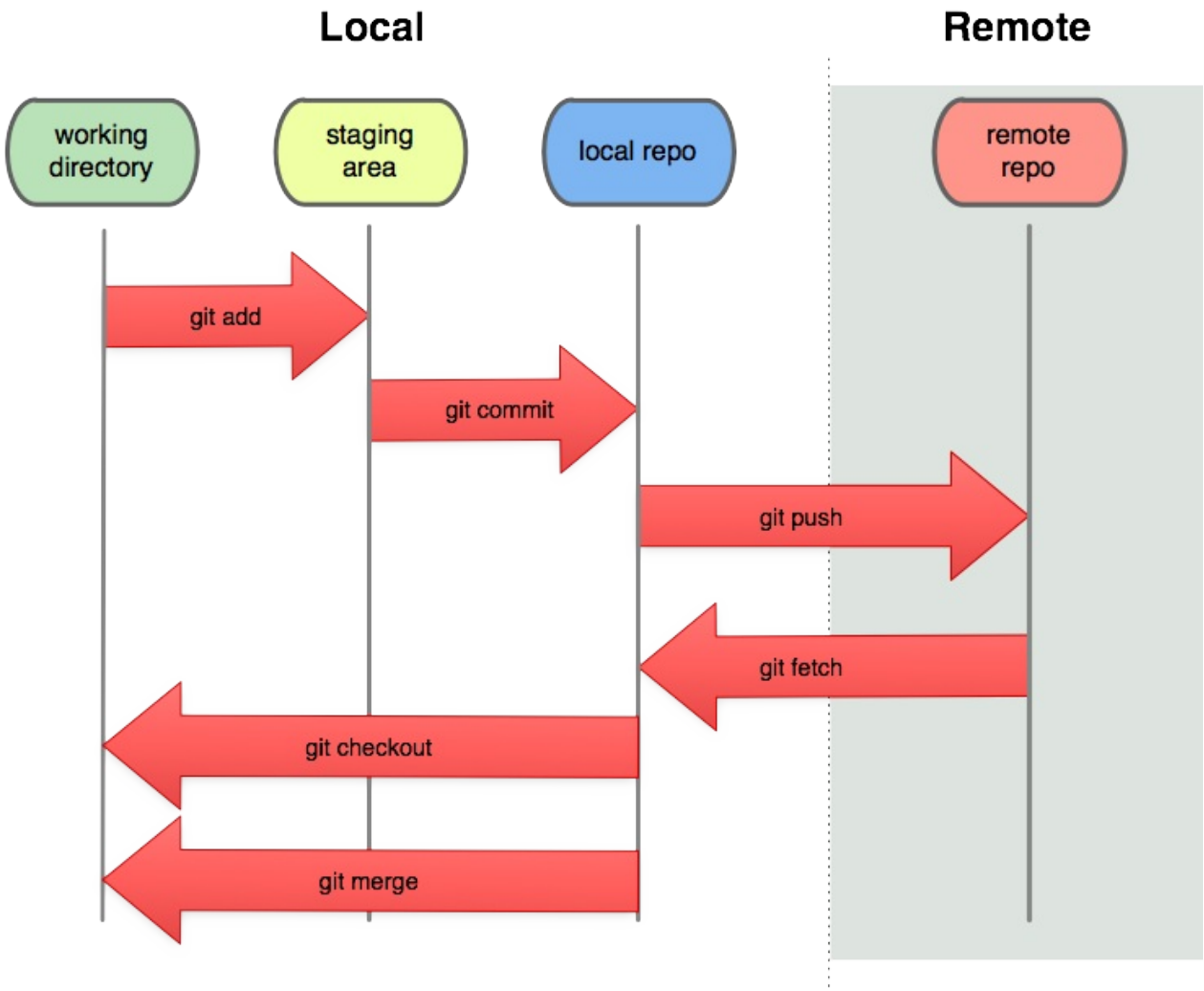
Benefit 3: Branching

Branches are commonly used to segregate new code from the existing project. Let's say your manager decides she wants to incorporate a new feature into your awesome app and only wants that feature to be available to the rest of the developers when it's complete and ready.



With Git you can work on your own branch and incorporate your changes at a later time, while retaining the other benefits of version control, as previously described.

Benefit 4: Local versus Remote - A Distributed System



When you're working with Git, you are working with your own copy. This is what we refer to as your **local** copy. With your local copy, you can do whatever you want. Add some new buttons to your app, change todo en español, get dangerous and experiment and tinker throughout the app.

> This is a great way to learn and experiment

This is your local copy and you're not affecting others. You can commit changes to your local copy (even without Internet access) and decide to abandon it all and get a fresh copy from the remote.

When you're ready, you can publish your changes from your local copy into the remote repository.

References

Why Git? - http://www.markus-gattol.name/ws/scm.html

Git User Interfaces

We are going to describe three different ways to interface into Git. Those are:

1. The GitHub website
2. SourceTree from Altassain
3. (optional) GitHub for Mac

Wrap Up

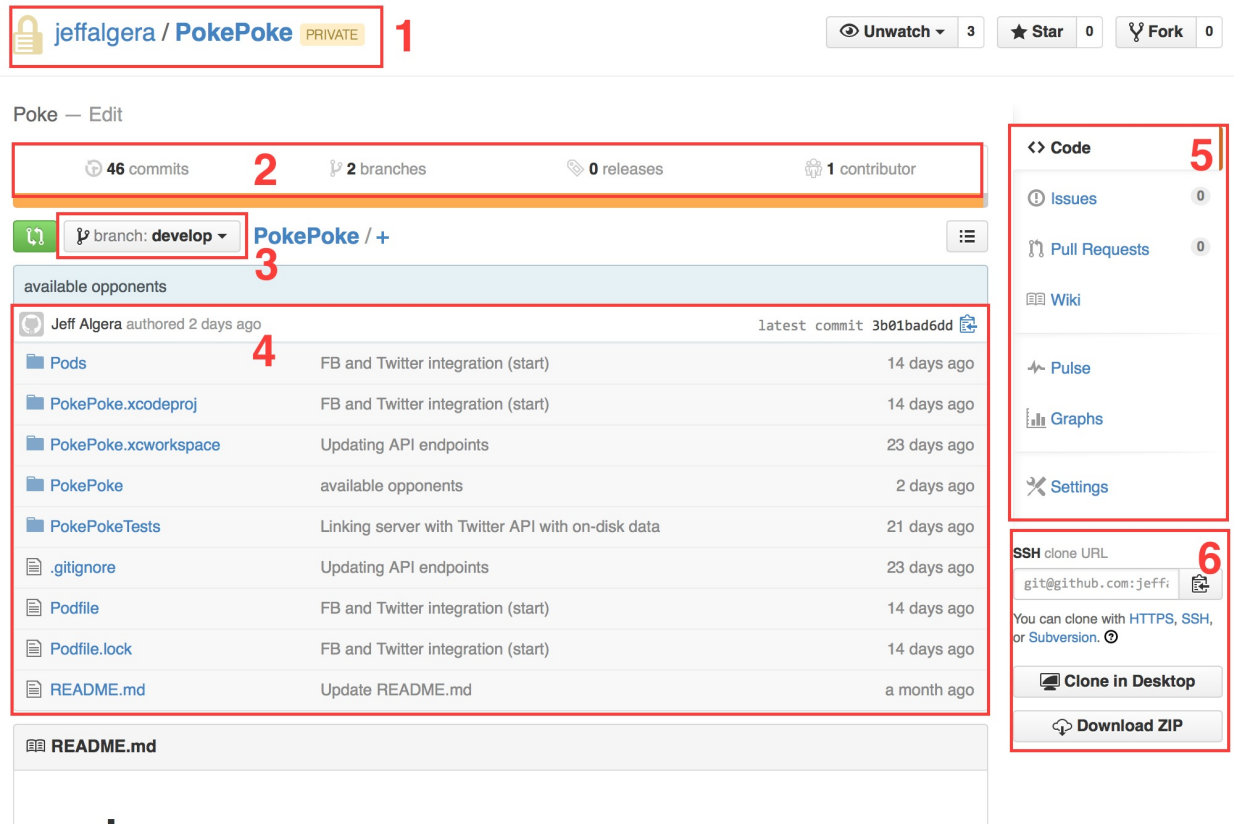
GitHub for web, SourceTree and GitHub for Mac are powerful tools that developers use each day. They enable the syncing of code across distributed systems and keep teams progressing a unified code base.

For further study, please consult this series of excellent YouTube videos published by Data School - https://www.youtube.com/playlist?list=PL5-da3qGB5IBLMp7LtN8Nc3Efd4hJq0kD

GitHub for Web

http://github.com

GitHub provides an excellent web interface for viewing and managing repositories.

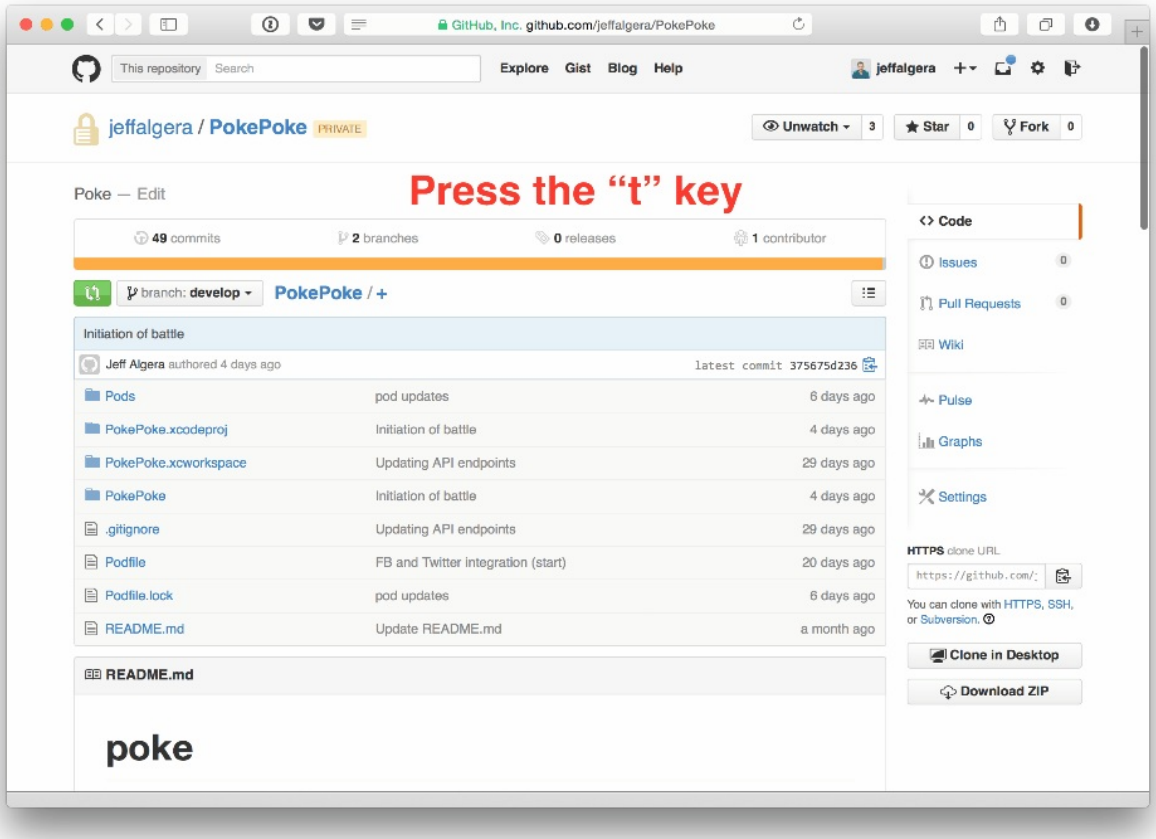


What you are seeing here is the remote representation of a repository. In detail:

1. **Repo name and type** - Describes who owns the repository, what the name of the repo is and whether the repo is public or private
2. **A selectable overview** of commits, branches, releases and contributors for a particular repository. Selecting any one of these options will bring a detail view of that selection.
3. A drop-down selection of the **currently viewed branch**. Changing the branch will update the source code to reflect that branch.
4. A selectable **source code list** organized within the committed file structure.
5. **Side bar** - use the side bar for responding to issues and creating pull requests. Also contained here are the settings per repository.
6. **Connection settings** - the lower right pane contains information on connecting to your repository. Typically, you will be wanting the SSH connection string to input into your Mac Github client. ZIP archive is good for downloading a backup of the repo.

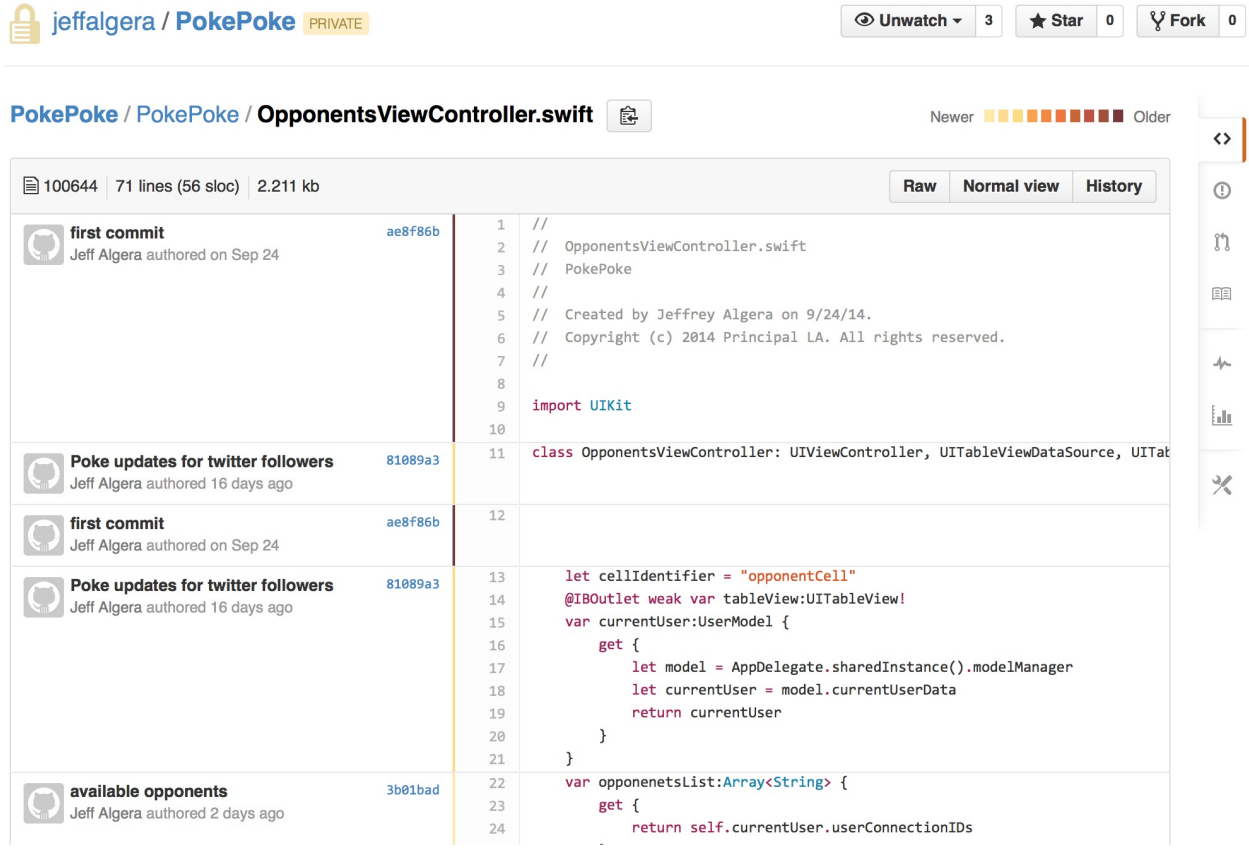
Blame

One of the great features of the GitHub website, is the ability to drill down and view specific information about individual source code files including commit logs.



The steps to view blame are as follows:

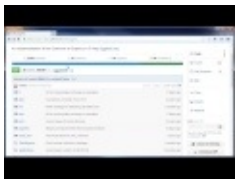
1. From the GitHub repository home page, hit the letter 't'
2. Being typing the filename. In our above example, we are typing AppDelegate
3. GitHub will filter in realtime file names, object and method names and display them in a clickable list
4. In our example, we were looking for AppDelegate.swfit, so we click on that
5. Then hit the **blame** button on the top-right toolbar
6. And behold, blame! A sequential list of your source code file with all changes made.



In the above graphic, we are looking at a blame of an individual source code file. Note the detail. You have a line-by-line listing of when that code was changed, by whom and a selectable commit entry to reference how that change was made. All items in blue are selectable.

[Videos] Navigating a GitHub Repository

Part 1:

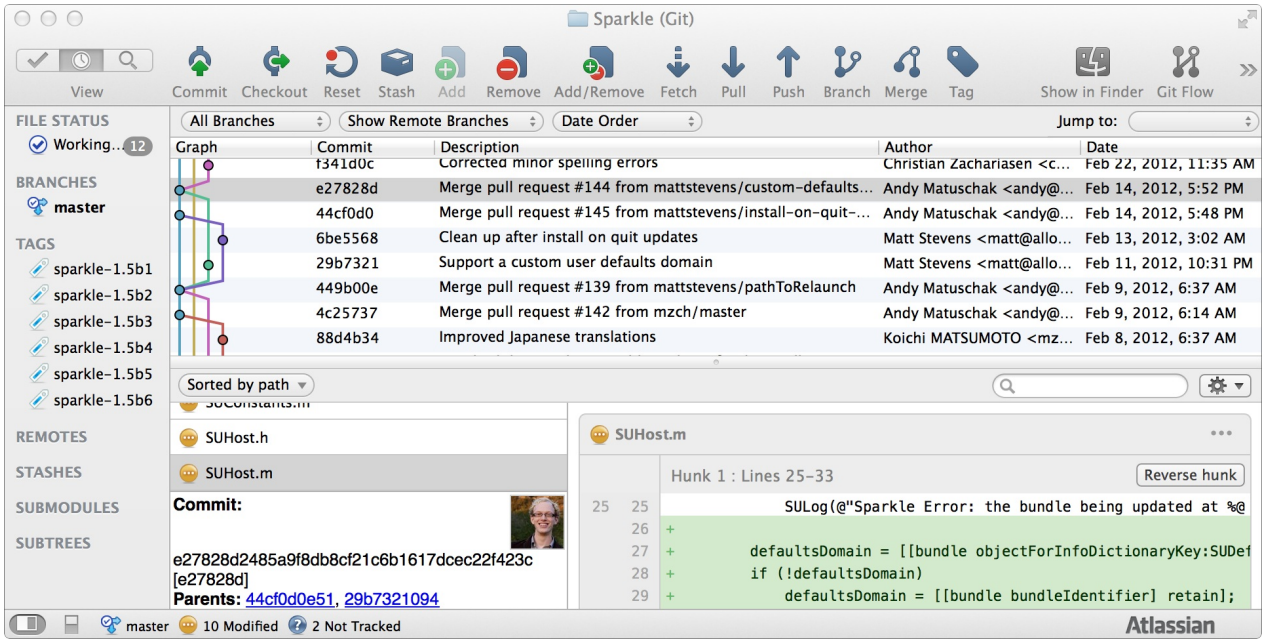


Part 2:

SourceTree for Mac

SourceTree is a Git client created by Altassian. They describe their product as, "(P)erfect for newcomers." Where "(c)reate, clone, commit, push, pull, merge, and more are all just a click away."

Download from http://www.sourcetreeapp.com



Window Layout

Below is a brief overview of SourceTree, its various controls and layout. This material is found within SourceTree itself by selecting Help from the file menu.

The Toolbar

The View Button This button allows you to switch between the 3 main views of the repository: File Status, Log, and Search.

Commit

Opens the Commit dialog so you can commit your changes. In Git, if you're using staging, the default is to open the dialog with the commit of staged changes selected.

Checkout

Allows you to switch your local working copy to a different point in history.

Reset

Use this to undo changes in your working copy.

Stash

Moves any uncommitted changes you have in your working copy to a storage area, and returns your working copy to a clean state.

Add

Adds any selected untracked files to source control - the next commit will start to track these files.

Remove

Stops tracking selected files and removes them from the working copy. The next commit will remove these files from being tracked in source control.

Add/Remove

A shortcut to stop tracking all files which are missing from the working folder, and to add any untracked files to source control.

Fetch

Download new commits from your remotes, but don't bring them in to your own branch yet.

Pull

Download new commits from your remotes and bring them in to your current branch, either by merging or rebasing.

Push

Upload new commits to a remote. This icon will have a number superimposed on it if you have commits which you haven't pushed yet.

Branch

Create a new branch (also includes a tab for removing branches)

Merge

Merge changes into your current branch.

Tag

Create and manage tags.

Create Patch

Create a patch file either from your current uncommitted changes, or from one or more commits that you've already made.

Apply Patch

Apply a patch file either to your working copy or directly to the commit log.

Terminal

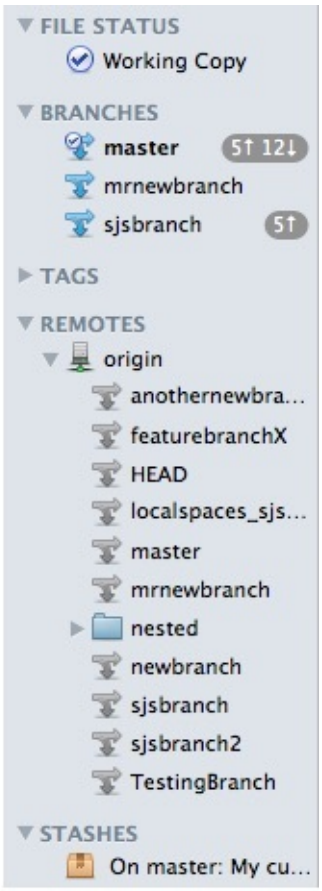
Open a Terminal at the selected location.

Settings

Access per-repository settings such as remotes.

The Sidebar

The sidebar is found on the left-hand side of the SourceTree application.



Git

The File Status Section

Selecting the "Working Copy" item behaves the same way as if you click on the File Status button on the View control on the toolbar.

The Branches section

This displays all your local branches. You can also right-click on the branch to get a number of context-sensitive options such as updating to/checking out the branch, merging it into your current branch, diffing that branch against your current branch. Double-clicking a branch will switch your current branch (after a confirmation dialog).

Your current branch is indicated with bold text and a 'tick badge' on the branch icon. The number of commits that your branch is ahead or behind the remote branch that it's tracking is displayed alongside the name, with an up arrow for ahead, and a down arrow for behind.

Note: if you create a branch with a name which includes a '/' character, SourceTree will display the branch as nested within a folder.

The Tags section

This section is not expanded by default but if you expand it you'll see one entry for each tag. Selecting one will jump to that tag in the Log view, and there are options available in the context menu such as updating to the tag, deleting the tag or diffing your current working copy against it.

The Remotes section This is where you find a list of remotes configured for this repository. You can pull from or push to a specific remote from the context menu on each item.

On Git, you will also find a list of remote branches, and you can perform actions on these via the right-click context menu or double-clicking to checkout.

The Stashes

A new item is created in this section, each one representing a set of changes that have been stored away for future use. You can click on these items to see a diff view of the changes they represent, and right-click an item to either apply it back to your working copy, or to delete it if you don't want it anymore.

The Submodules

If your project has nested repositories within it, called submodules. They will appear here. See the submodules section for more details.

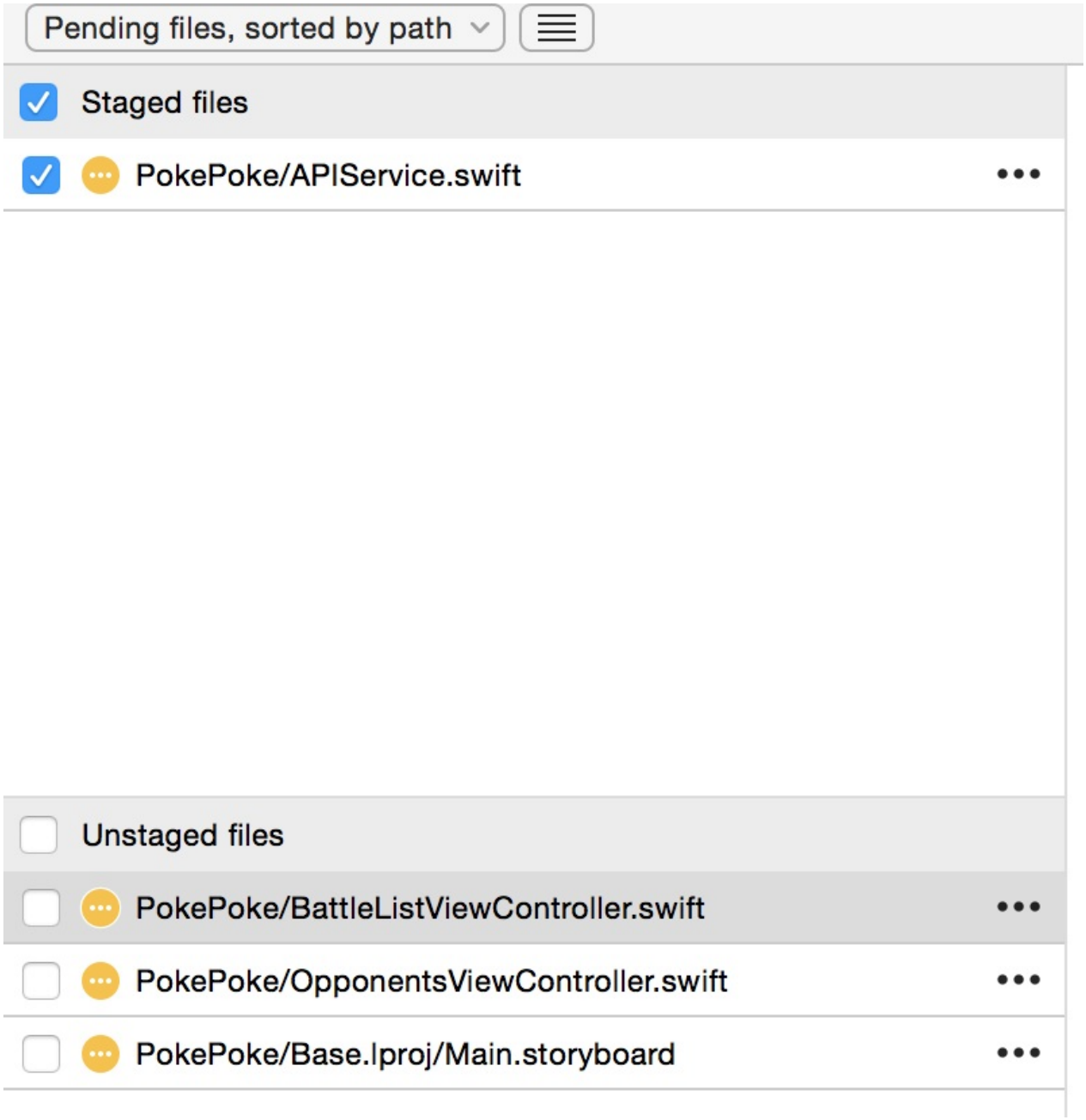
The Footer

Found at the bottom of SourceTree window is the footer.

You'll see here the name of the current branch, followed by either a 'Clean' marker if there are no local changes, or a summary of the number of files outstanding in each state if there are local changes. You will also see information here relating to any outstanding merges and conflicts.

File Status List

On the left-side of the SourceTree window, after the sidebar, is the File Status list.



This area shows files in your working copy which have been modified in some way, although you can use the checkboxes at the bottom to filter the statuses that are displayed. Selecting one of these files will display those changes in the differences pane.

Differences Pane

Colorful green/pink view on the right side of the SourceTree application.

PokePoke/APIService.swift

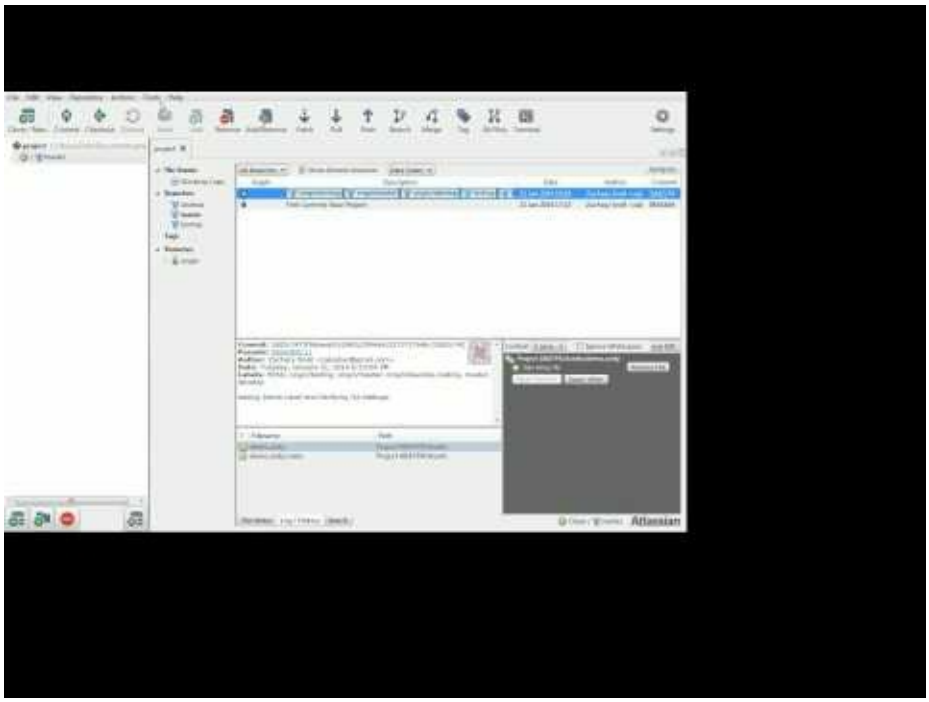
Hunk 1 : Lines 85-119 Reverse hunk

```
85  85                          user.userEmail = user_email
86  86                      }
87  87                  }
88      -
89      -                   if let userConnections = userData["user_connection
90      -                       user.userConnectionIDs = userConnections
91      -                   }
        88  +
        89  +                   completion(result: true)
        90  +                   return
        91  +               }
        92  +               completion(result: false)
        93  +           }
        94  +       }
        95  +
        96  +       func getUserBattles(user:UserModel, completion:(result:Bool) -
        97  +           var url = self.constants.serverURL + "get_active_battles"
        98  +           var params = Dictionary<String, String>()
        99  +
       100  +           if let userID = user.userID {
       101  +               params["user_id"] = userID
```

This is a color-coded view with green describing additions, pink describing deletions and white describing no changes.

[Video] Git + SourceTree Tutorial

The following video provides an introduction to Git using SourceTree:



https://www.youtube.com/watch?v=gdJ01t3KYH0

SourceTree Resources

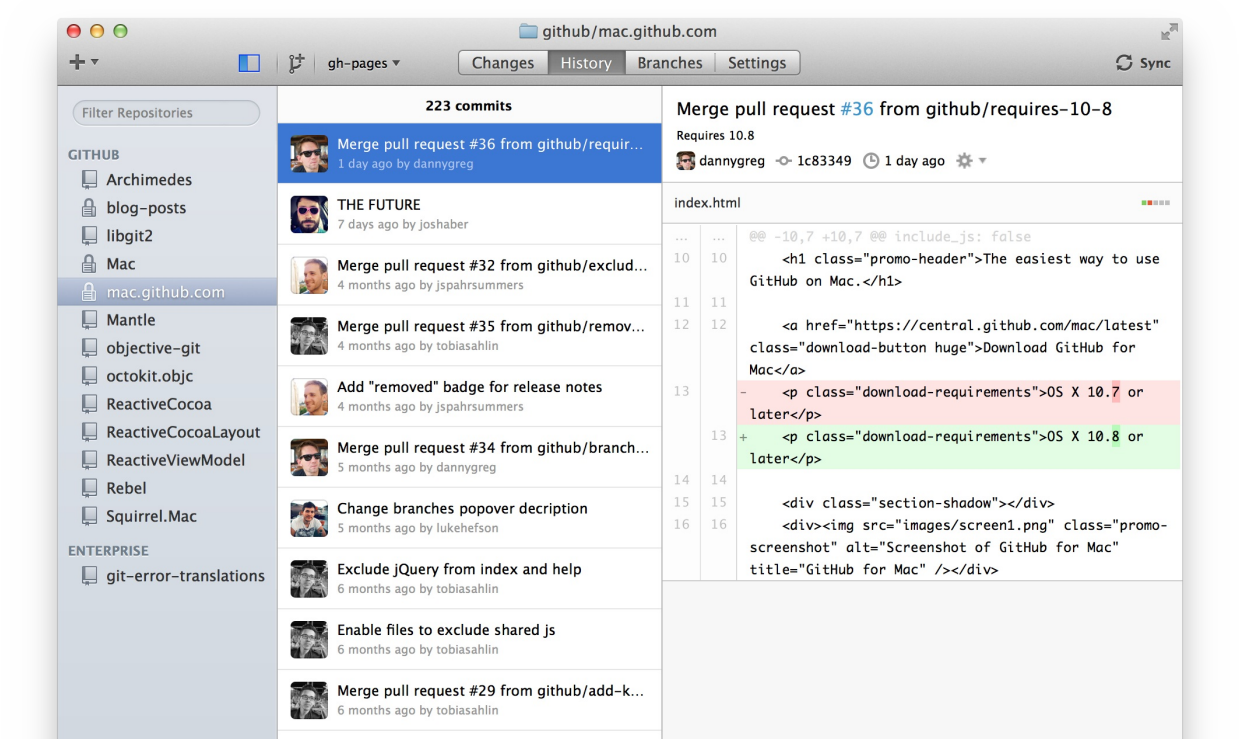
FAQ from SourceTree - http://www.sourcetreeapp.com/faq/

Support for SourceTree - http://www.sourcetreeapp.com/support/

BitBucket? It is important to note that there are competitors to GitHub. Altassian is one of those competitors. In addition to this Mac app, they also make a web interface called BitBucket, which is Altassian's answer to GitHub.

GitHub Mac App

Download GitHub for Mac here: https://mac.github.com



The GitHub for Mac client provides a easy-to-use interface for Git and GitHub. GitHub has a fantastic overview of their product which we will outline below:

1. **First Launch** - Just getting started with a fresh download of GitHub for Mac? https://help.github.com/articles/first-launch/
2. **Working with Repositories** - Want to fork your first repository? Use this resource: https://help.github.com/articles/working-with-repositories/
3. **Making Changes** - View your source, commit locally and sync your changes with the server - https://help.github.com/articles/making-changes/
4. **Viewing History** - Want to see what's changed and when? https://help.github.com/articles/viewing-previous-commits/
5. **Branches** - Create a branch after a successful clone - https://help.github.com/articles/branching-out/
6. **Merging branches** - Once you're ready to merge your branch into another branch, check out - https://help.github.com/articles/merging-branches/

Mac App Resources

Keyboard shortcuts and further informative resources can be found here: https://mac.github.com/help.html

Common Terms

After using Git for a number of years, developers have created a language around the tool and their process. This list can be used as a quick reference to decipher those messages and assimilate this language into your own.

Terms

- **Add** - Creating a new file within the repository
- **Blame** - A term used to find who changed what line of code and when.
- **Branch** - describes a separate line of development from another branch
- **Checkout** - retrieve the latest copy of a repository into your local branch
- **Clone** - to initially copy a remote repository onto your computer.
- **Commit** - to take your changes and submit them to the repository
- **Conflict** - when two commits affect the same code and Git has trouble automatically merging the two
- **Git** - pronounced with a hard "G", rhymes with spit
- **GitHub** - A company in San Francisco that made an excellent interface to Git repositories
- **Local** or **Local changes** - a cloned copy of a repository that may have changes not yet pushed to the remote repository.
- **Master** - Typically, the base branch of which all other brances are derived
- **Merge** - Take a series of commits from one branch and put them into another branch
- **Merge conflict** - Same as conflict
- **Publish** - To submit your changes to the remote repository. May also refer to the creation of a branch on the remote repository.
- **Push** - Same as publish without the creation connotation.
- **Pull** - To retrieve changes on a remote repository into your own local repo.
- **Repo** or **Repository** - Where the source code lives. For our purposes, this is GitHub
- **Stage** - The precursor to commit. Observe changes about to be commited into your branch before those changes go into the local copy of your repo.
- **Stash** - To quickly push aside your changes into a temporary state which can be retrieved later or abandoned.
- **Version Control** - A system that handles keeping track of changes and provides automation around merging changes from multiple streams of input, such as people.

References

GitHub Glossary - https://help.github.com/articles/github-glossary/

Git Manual (*Warning: Heavy material, approach with coffee*) http://gitref.org

Simple guide to forks in GitHub and Git http://www.dataschool.io/simple-guide-to-forks-in-github-and-git/

Stack Overflow Q&A - http://stackoverflow.com/questions/7076164/terminology-used-by-git

Getting Started

On behalf of the developer community, welcome! We're glad you're here. Let's get started.

This section contains a number of chapters which will provide an overview of the tools you will use to create your next mobile app. Each provides references and useful resources to guide you in your learning journey.

It is important to note, as you work with Xcode you will become more familiar. Start with an overview, observing controls and tool tips. Move into detail when comfortable.

Quickstart Tool Overview

1. Xcode - An overview of Xcode IDE, how to navigate Xcode and associated resources.
2. Inteface Builder - Visually create and debug views with Interface Builder.
3. Playgrounds - Playgrounds are a way to experiment with code in a sandbox environment.

Further resources

Apple provides a searchable directory of informative resources including source code tutorials available for download.

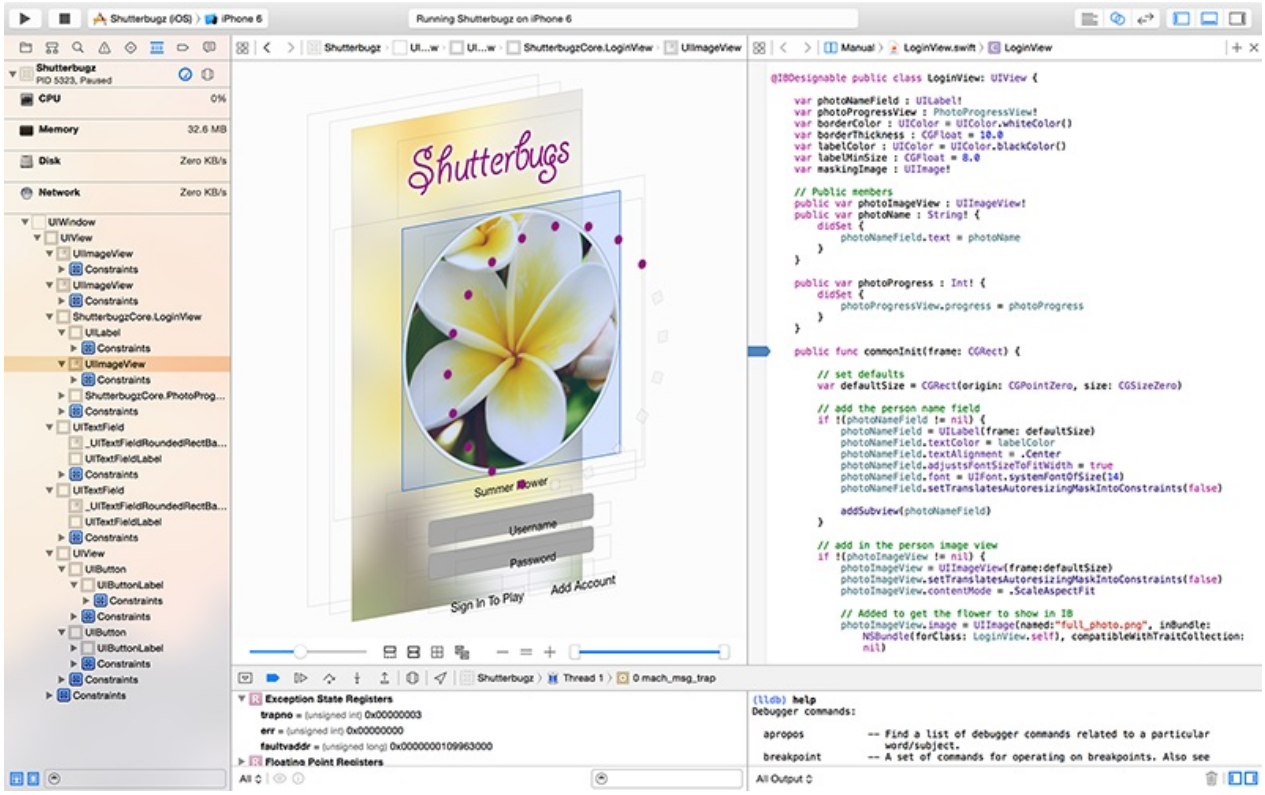
iOS Developer Library - https://developer.apple.com/library/ios/navigation/

Xcode

Xcode is a powerful **I**ntegrated **D**evelopment **E**nvironment (or **IDE**) created by Apple for developers. Xcode is used to build native mobile applications for iOS devices. You will be using Xcode exclusively for writing source code and debugging your apps. Xcode comes bundled with a number of applications including:

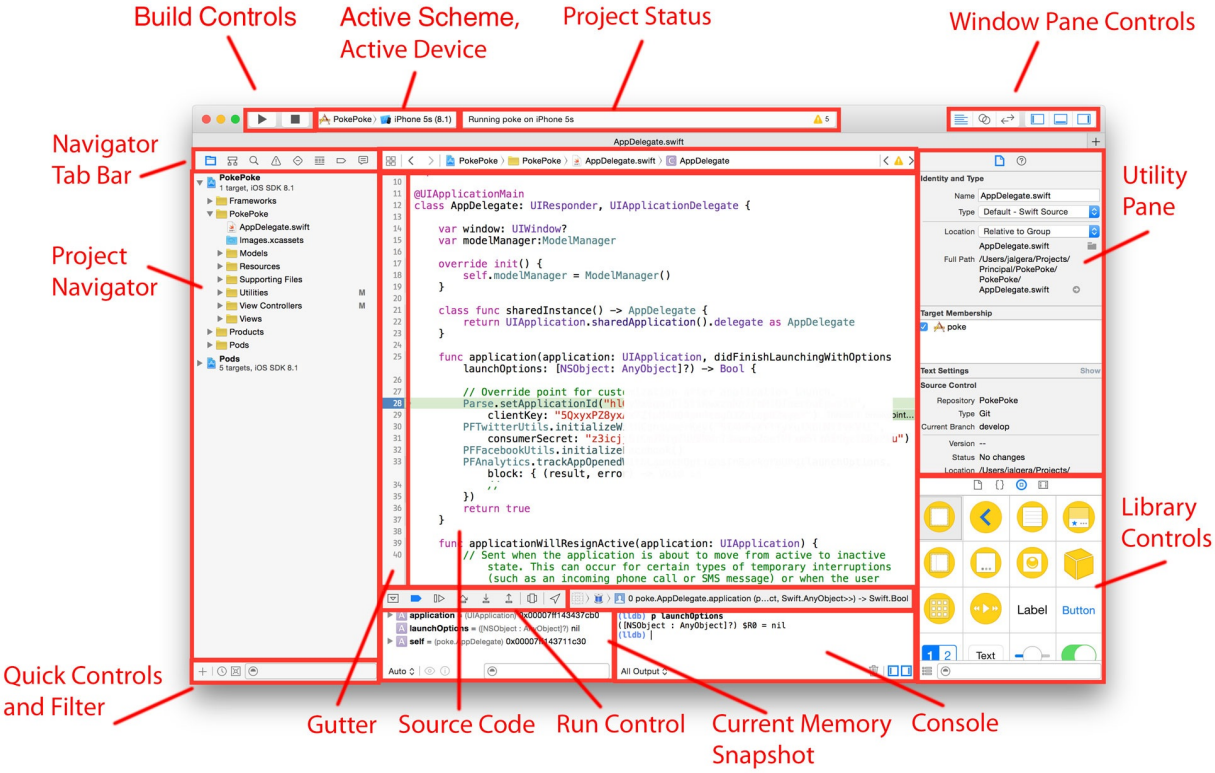
1. **Source code editor**, compiler and debugger - used to write and test code
2. **Interface Builder** - Used to build visual interfaces for your users
3. **iPhone and iPad Simulator** - Used to run your apps easily and quickly without the need to connect your device.

The latest version of Xcode is **Xcode 6** and can be downloaded by visiting the following link:
https://developer.apple.com/xcode/



Overview of Xcode

Below is a labeled diagram describing Xcode - its panes, toolbars and general layout. Since you will be using Xcode on a regular basis, it is important to know where the buttons, views and toolbars are located. In the below diagram, Xcode is currently debugging an iPhone app:



1. **Build Controls** - Runs and stops the current build
2. **Active Scheme, Device** - Selectable list of schemes and available simulator and connected devices to run the build.
3. **Project Status** - Current status of the project including active errors and warning counts.
4. **Window Pane Controls** - Configures Xcode window panes into a variety of options.
5. **Navigator Tab Bar** - Optional states for the navigator tab bar such as symbols, search and breakpoint list.
6. **Project Navigator** - List of files contained in your project or workspace.
7. **Quick Controls** - Contains a filter for the above Project Navigator and a quick add files button ("+")
8. **Gutter** - Can contain line numbers associated with your source code and active, non-active breakpoints.
9. **Source Code** - Editor for source code
10. **Run Control** - Options for debug including toggling breakpoints, step controls and run and pause.
11. **Memory Snapshot** - During debug, shows active, recently changed memory in a read-only format.
12. **Console** - During debug, provides NSLog output as well as the ability to input commands directly to the debugger.
13. **Library** - UI Library available for Interface Builder.
14. **Utility Pane** - Configuration and utility functions available per file or control within Interface Builder.

[Video] - What's New in Xcode (WWDC 2014)

Each iteration of Xcode, Apple releases an informative video describing new and important features of their IDE. The following video will walk you through these features.

Now that we have an overview of Xcode, our next chapter will discuss Interface Builder.

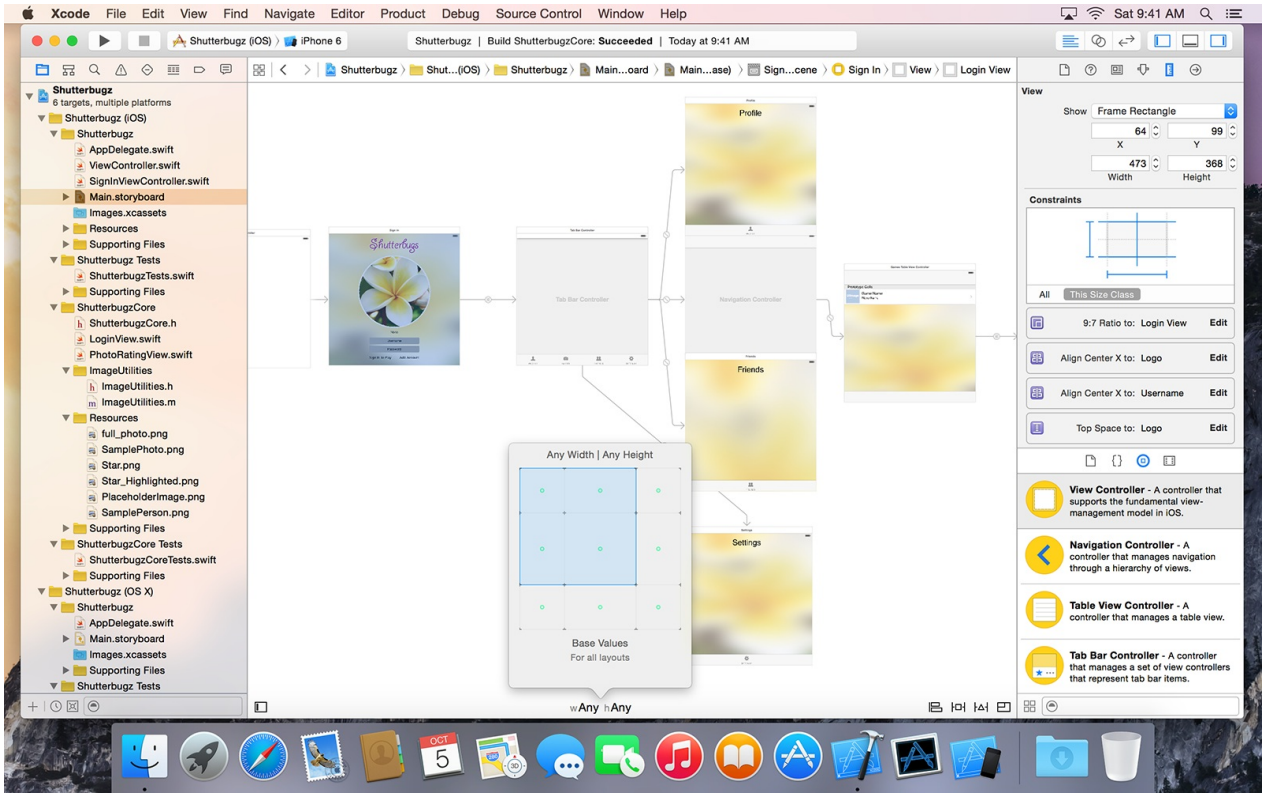
References

Xcode Overview - https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/chapters/about.html

Xcode: Learn More - https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/chapters/LearnMoreAboutXcode.html

Interface Builder

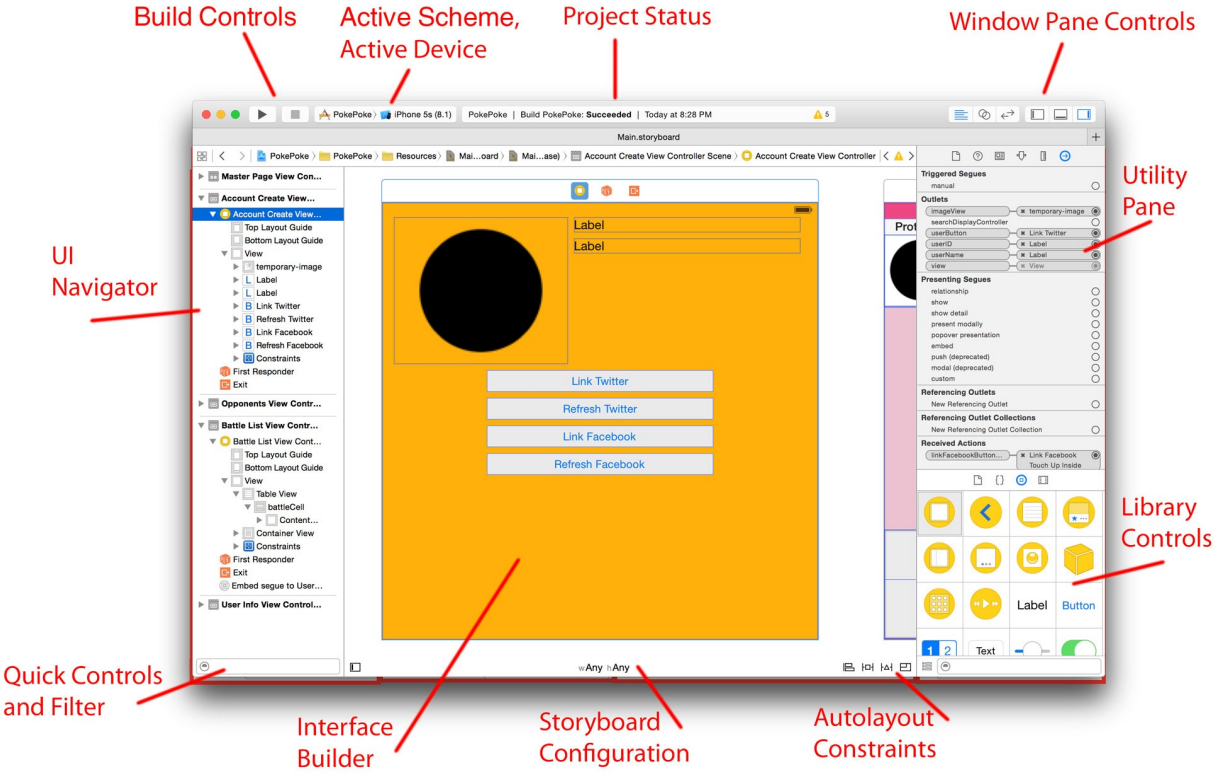
Interface Builder (aka **IB**) is included within the Xcode IDE. You will use it to drag and drop visual elements onto a canvas and build a compelling user interface.



Let's begin with an overview of the tool, where the various buttons, toolbars and view are and why you would use them.

Overview of Interface Builder

Below is a labeled diagram describing Xcode - its panes, toolbars and general layout. In the below diagram, Xcode is currently building an example iPhone app:



1. **Build Controls** - Runs and stops the current build
2. **Active Scheme, Device** - Selectable list of schemes and available simulator and connected devices to run the build.
3. **Project Status** - Current status of the project including active errors and warning counts.
4. **Project Navigator** - List of files contained in your project or workspace.
5. **Quick Controls** - Contains a filter for the above Project Navigator
6. **Interface Builder** - Represents an abstract view used for positioning and layout. Meat and potatoes. Where you will spend most of your time when buliding UI.
7. **Storyboard Configuration** - Selectable variants of the above storyboard used to differeniate specific layouts for varying devices.
8. **Autolayout Constraints** - Add new constraints, and respond to constraint errors. Also contains the ability to remove contraints.
9. **Library** - UI Library available for Interface Builder.
10. **Utility Pane** - Configuration and utility functions available per file or control within Interface Builder.

As with other parts of Xcode, the more you use the tool, the more familiar you will become with it.

[Video] Universal Storyboards

This is a little advanced in detail, but from a cursory overview, you can watch how this developer moves through Interface Builder, configuring Xcode and navigating various controls and utilities.

Although some of that video was advanced, have no fear. We will be returning to Interface Builder in later chapters.

References

Overview of Interface Builder - https://developer.apple.com/xcode/interface-builder/

Build a User Interface - https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/chapters/edit_user_interface.html

Advanced - Visual Debugging with Xcode http://www.macstories.net/mac/xcode-6-live-rendering-visual-view-debugging-and-swift/

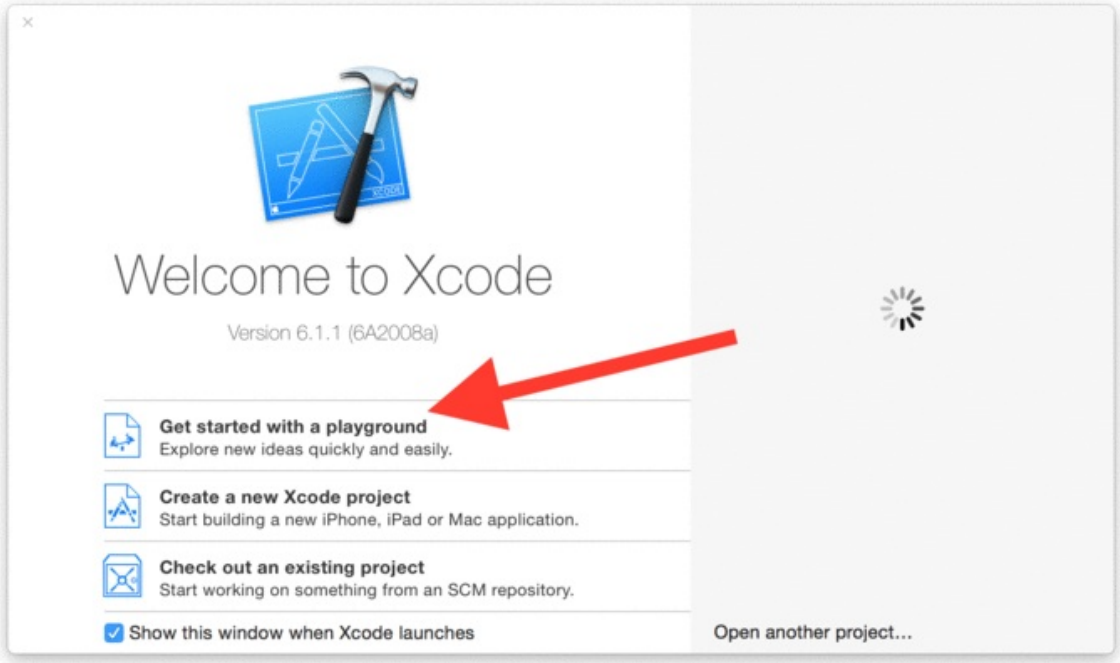
Playgrounds

Playgrounds are a fantastic way to learn programming in Swift. They allow you to quickly experiment and test sections of code without having to build and compile to a device or simulator.

Creating a new Playground with Xcode

We will go through the quick process of opening a new Playground and start editing some Swift code:

1. Open Xcode
2. Select "New Playground" or select "File" -> "New" -> "Playground"
3. Create a name for your Playground and save it to disk
4. Update code in the left window to see it update on the right!



It's really simple to get started and there are powerful things you can do. Use Playgrounds to test out new code, experiment with someone else's code and **isolate problem areas from your existing projects.**

Let's see how we can get started in Playgrounds with an example project.

Example

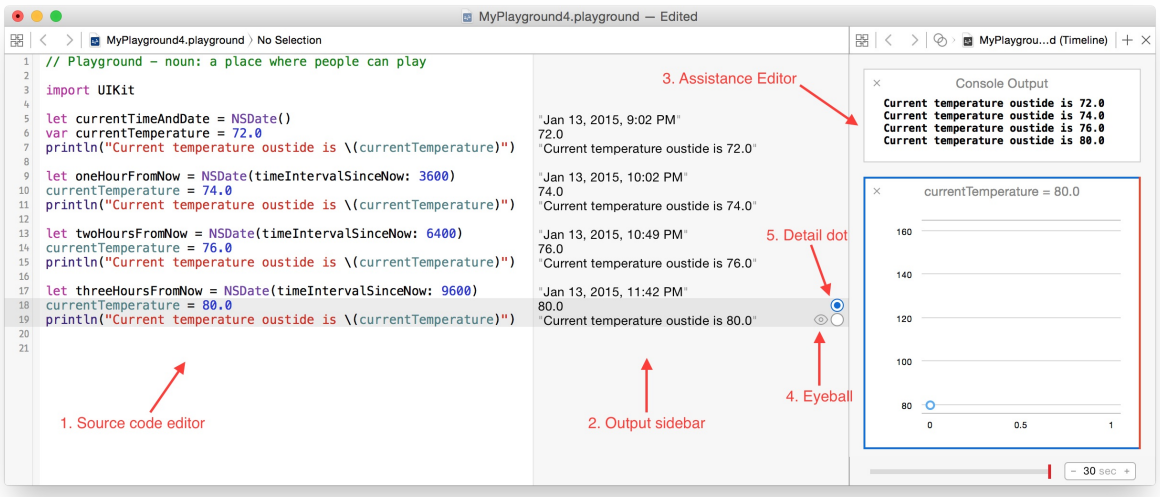
Following the steps above, open a new Playgrounds project. Copy and paste the following code into your source code view:

```
import UIKit

let currentTimeAndDate = NSDate()
var currentTemperature = 72.0
println("Current temperature oustide is \(currentTemperature)")

let oneHourFromNow = NSDate(timeIntervalSinceNow: 3600)
currentTemperature = 74.0
println("Current temperature oustide is \(currentTemperature)")

let twoHoursFromNow = NSDate(timeIntervalSinceNow: 6400)
currentTemperature = 76.0
println("Current temperature oustide is \(currentTemperature)")

let threeHoursFromNow = NSDate(timeIntervalSinceNow: 9600)
currentTemperature = 80.0
println("Current temperature oustide is \(currentTemperature)")
```

The above code will update a temperature variable based on new times throughout the day. In later chapters we will go over the syntax of Swift, but for now, we are just looking at Playgrounds from a tool perspective.

Your playgrounds file should now look like the below. In our diagram, you can see

1. Where source code is written and modified
2. Where the current output of code is shown
3. Where you can view specific objects in detail
4. Eyeball - selecting the eyeball will show detail about that specific line of output. Use this to quickly see detail about the current variable.
5. Assistance pin - Use this circle to pin a new window into the assistance editor. This is helpful for viewing changes to a particular variable over time.

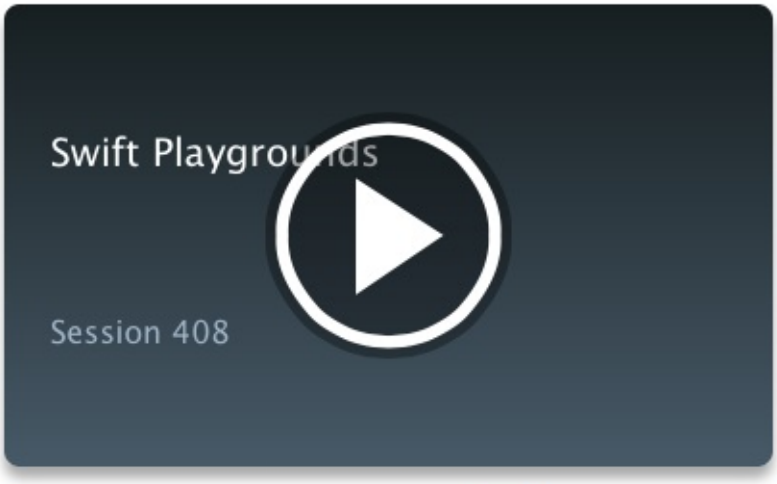


The majority of our examples in this Gitbook will be shown in Playgrounds. It's a good idea to start getting more comfortable now. Part of this process will be watching Apple introduce Playgrounds at WWDC 2014.

[Video] Swift Playgrounds

Apple introduce Playgrounds as part of the new Xcode 6 at this year's WWDC event. Below is an overview to getting

started with Playgrounds straight from the horses mouth at this year's WWDC event.



Explore how Playgrounds provide new workflows, enable rapid development, help you conveniently step through your code to diagnose bugs, and make it easier than ever to learn new concepts. (Apple WWDC Session 408)

References

Exploring and Evaluating Swift Code in a Playground

Experiment with Swift code in an interactive environment known as a playground.

https://developer.apple.com/library/ios/recipes/xcode_help-source_editor/chapters/ExploringandEvaluatingSwiftCodeinaPlayground.html

Intro to Swift

Now that we went over the tools needed to create mobile apps, let's move into the programming language we will be using to code. These chapters will go over the **Swift** programming language, concepts of software programming and then we'll return to Interface Builder with some real-world examples.

> Swift is a new programming language for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility. Swift adopts safe programming patterns and adds modern features to make programming easier, more flexible, and more fun.

Source

This section will begin to outline concepts of the Swift programming language including:

1. Swift - The programming language
2. Key Programming Concepts of programming languages with examples in Swift
3. More Interface Builder - Overview of Cocoa controls and making connections to code

References

About Swift

https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/index.html

Swift - Overview

https://developer.apple.com/swift/

Swift Reference Book

https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

Style Guide Reference

https://github.com/github/swift-style-guide

Trending Swift Repositories on GitHub

https://github.com/trending?l=swift&since=monthly

Welcome to Swift

Hello and welcome to Swift, Apple's programming language designed to for creating iOS and Mac applications.

Swift is a new programming language developed by Apple for the development of iOS and OSX applications. Like many other high-level languages, Swift facilitates the creation of complex objects and modern programming patterns. This is different from the C-based paradigm of Swift's predecessor, Objective-C. Swift also aims reduce overall amounts of code from Objective-C by providing a more succinct language and file structure.

Swift was introduced by Apple at the WWDC 2014 event.

[Video] Introduction to Swift

The following gives a great overview of the language of Swift and why developers should use it:

The Basics

Below, we will list a few basics of the Swift programming language. These examples can be pasted into a Playgrounds window and changed there. By using Playgrounds, you will be able to ensure the code compiles, runs and see the output of each line in real-time.

Declaring a variable

Variables represent data within a Swift application:

```
let staticVariable = "This won't ever change"
var dynamicVariable = "This can change"

dynamicVariable = "... you've changed!"
```

Optionals

In Swift, you can define a variable that may be uninitialized (or nil) by using the "?" operator.

```
var newClientName: String?
// newClientName is initialized as nil
```

If you know that an optional variable is currently not nil, you can force unwrapping by using the "!" operator

```
var newClientName: String?
newClientName = "Apple"
let client = newClientName!
```

Performing arithmetic

You can then use variables to perform actions, like math. Swift defines a variety of operations for mathmatical computation including:

- = Equals
- + Addition
- - Subtraction
- / Division
- * Mulitplication
- % Remainder
- ++ Increment
- -- Decrement

```
let a = 1
let b = 2
var c = a + b
print(c) // prints 3

var d = c + a
print(d) // prints 4
```

For more examples on mathmatical operations, please visit: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/BasicOperators.html

Control flows

There are a variety of control flows you can use to iterate over data and repeat a particular set of instructions.

These examples will be adpated from the Swift Programming Langugae book [1]

For-in

The For-In loop will repeat its control block until the condition is met.

```
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    println("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

For

Swift also supports the more traditional For control loop.

```
// Format
for initialization; condition; increment {
    statements
```

```
}
```

Refactoring our last example, we can come up with the same result.

```
let names = ["Anna", "Alex", "Brian", "Jack"]

// initial; continue until;
for var index = 0; index < names.count; ++index {
    println("name at index \(index) is \(names[index])")
}
// name at index 0 is Anna
// name at index 1 is Alex
// name at index 2 is Brian
// name at index 3 is Jack
```

While

Another type of control is the while loop. This loop will continue until the condition is met. Special care should be taken to avoid running an infinite loop.

```
while condition {
    statements
}
```

For example:

```
var counter = 1
while counter <= 5 {
    println("we've counted to \(counter)")
    counter++
}
//we've counted to 1
//we've counted to 2
//we've counted to 3
//we've counted to 4
//we've counted to 5
```

Conditionals

Swift provides a standard set of conditional statements that can be used for checking states of variables and performing optional chains of events.

If Else

The most common conditional statement is evaluating a variable for a particular condition. For example

```
var composerType: String?

if composer == "Bach" {
    composerType = "Baroque"
}
else if composer == "Beethoven" {
    composerType = "Classical"
}
else if composer == "Schumann" {
    composerType = "Romantic"
}
else { // if the above conditions are not met, the execution will come here
    composerType = "Uknknown"
}

print("\(composer) is \(composerType)")
// Beethoven is Classical
```

Switches

> A switch statement considers a value and compares it against several possible matching patterns [1]

The patterns in this case can be a single value, multiple values and ranges of values.

```
switch value to consider {
case value 1:
    respond to value 1
case value 2,
     value 3:
    respond to value 2 or 3
default:
    otherwise, do something else
}
```

As we fill out our example below based on the above format, we are also introducing the range operator **(...)** which is a quick way to include a range of values from x to y.

```
let breakfastCost = 10

switch breakfastCost {
case 0:
    print("Free?!  I'll take it!")
case 1...4:
    print("Sounds good.  I'll buy it!")
case 5...7:
    print("Grumble.  I guess.")
case 8...10:
    print("I'm going elsewhere")
default:
    print("Highway robbery!")
}
```

Storage

Swift provides a variety of storage types (aka **variables**). Let's go over some of them now.

Swift types

We have already used quite a few of these up to this point.

```
let likeCats: Bool = true
let numberOfCatsOwned: Int = 25
let dollarsCatsCostMe: Double = Double(numberOfCatsOwned) * 500.0
let name: String = "Batman"
```

The second to last example **casts** (converts) the numberOfCatsOwned which was an Int to a Double with the syntax Double(Int).

Tuples

Tuples are a group of two or more variables coupled together. They can be different types. Tuples provide a convenient way of passing around a set of information in code.

```
let balloon = (12.0, "Red")

let (diameter, color) = balloon
println("The balloon diameter is \(diameter)")
println("The balloon color is \(color)")
```

```
// Output
// The balloon diameter is 12.0
// The balloon color is Red
```

Arrays

> An array stores multiple values of the same type in an ordered list. [1]

Arrays are useful for storing and enumerating collections of similar types.

```
var lbNeighborhoods = ["Downtown", "Bixby Knolls", "North Long Beach", "Wilmore", "California Heights", "Cambodia Town"

for hood in lbNeighborhoods {
    if hood == "North Long Beach" {
        println("\(hood) is home!")
    }
    else {
        println(hood)
    }
}

// Create a new, blank array which will contain Strings
var placesLived = Array<String>()

// Add some data
placesLived.append("Downtown")
placesLived.append("North Long Beach")
```

Dictionaries

Dictionaries are a set of key-value pairs. Each key must be unique.

```
var contacts = ["Jeff" : "201-555-5555", "Joe" : "212-555-6666", "Sarah" : "917-555-7777", "Obama" : "202-555-8888"]

var obamaPhone = contacts["Obama"]
print(obamaPhone)
// 202-555-8888
```

Defining a function

Organize actions into functions:

```
func exampleFunction(thisIsFun:Bool) -> String {

    var returnValue = "No, this isn't fun."
    if thisIsFun {
        returnValue = "Yes, this is fun!"
    }

    return returnValue
}

let enjoyment = exampleFunction(true)
print("Is Swift fun? \(enjoyment)")
```

These are some of the basic tenants of the Swift programming language. As we move the course, you will experience more of the Swift programming language expanding your knowledge and repertoire.

References

[1] Swift Programming Language
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language

Swift Resources https://developer.apple.com/swift/resources/

Style Guide Reference https://github.com/github/swift-style-guide

Trending Swift Repositories on GitHub https://github.com/trending?l=swift&since=monthly

Key Programming Concepts

Below is a list of concepts associated with Swift and other similar programming languages. The goal of this chapter is to frame your experience and provide a grounding from which to launch.

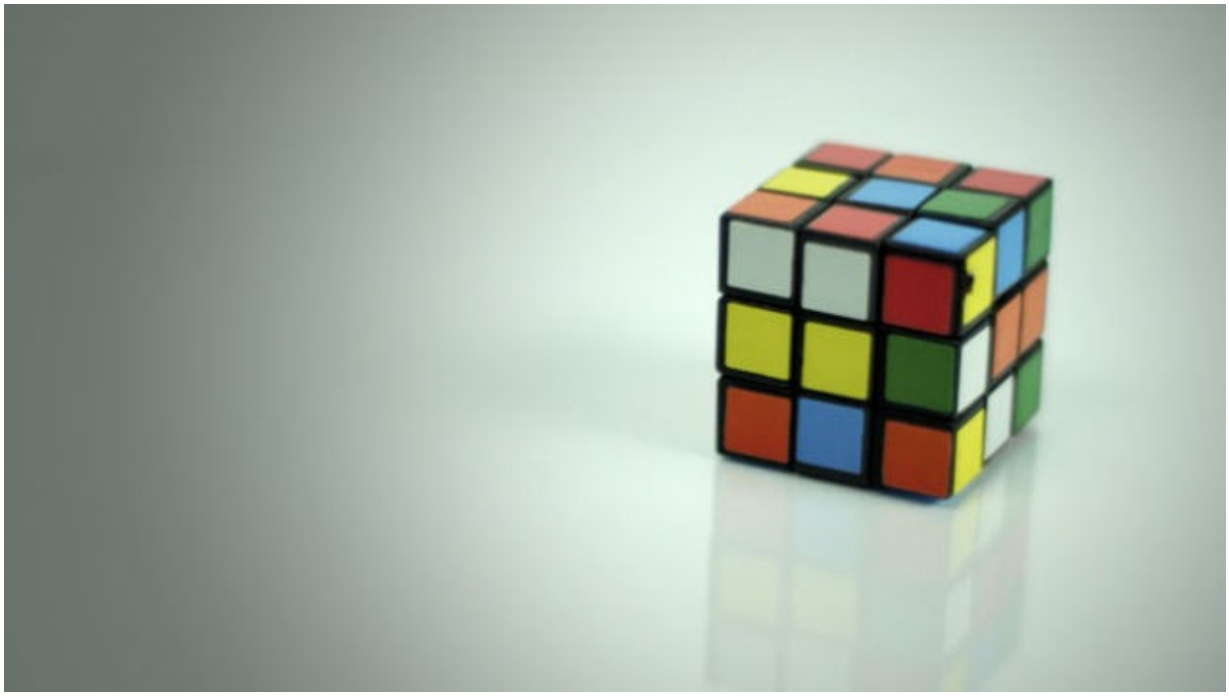
Programming Mindset

Below are three concepts to consider when getting into the programming mindset. These concepts will help you stay focused and be productive.

Break it down

It's easier to program when you understand what you are trying to accomplish. What do you want to happen? Can you break it down further?

A Systematic Approach to Solving Just About Any Problem



Having Fun

When you choose what to program, what to build and what to release to the world, realize that it takes time. *A lot of time*. This is all part of the process. Keep at it, try and have fun. The more experience you have the easier it becomes.

Make it easy

As a developer, you'll spend a lot of time reading code. Making code easy to read helps the process of reading and understanding code. Not just for you, but future you. And future employee, too.

```
// This is harder to read
let x = 1 + 2
let y = 3 + x + 2
let z = 1 + 1 + 1 + 1 + 1 + 1 + 6 / 2

// This is easier to read
let x = 3
let y = 5 + x
let z = 9
```

Choose easy. Follow the above concepts and future you will thank you with a high five.

Key Programming Concepts

Fundamental actions and terms that are central to executing code.

Commenting

Leaving comments helps future you understand your thought process. As a general rule, keep it short and meaningful. Code should be commented where reading the code may cause confusion.

```
// Example single line comment.  Preferred style.

/*
    Muli-line comment
    Use for commenting out large blocks of code temporarily.
    Don't use long-term.  Makes reverting comments difficult.
*/
```

Importing

There are numerous libraries that provide added functionality to your code. This is a clean way to import other's code into your own, effectively extending the possibilities of your code with minimal effort.

```
// Example iOS imports
import UIKit // Imports iOS UIKit library for various controls such as UIButton, UILabel, etc.
import Foundation // Imports foundation objects, typically prefaced with "NS" such as NSString, NSData, etc.
```

Storing Data

Storing data is complicated. Data can be structured in a variety of ways. Data can be sent and received in a variety of ways. Swift provides a variety of buckets you can use for storing data. The comments in the below code delineate various types of in-memory data storage. It's important to start thinking about how you will store and manipulate data.

```
// Storing one piece of data
let myName = "Jeff"

////
// Storing an array of data
let dangerousSnakes: [String] = ["Rattlesnake", "Cobra", "Python"]

////
// Storing a rectangle
struct Rect {
    var length:Int
    var width:Int
    var color:String
    var area:Int {
        get {
            return length * width
        }
    }
}

let rectangle = Rect(length: 10, width: 10)
let area = rectangle.area // 100

////
// Storing a complex object
class ComplexObject {
    var objectName:String
    var objectType:String
```

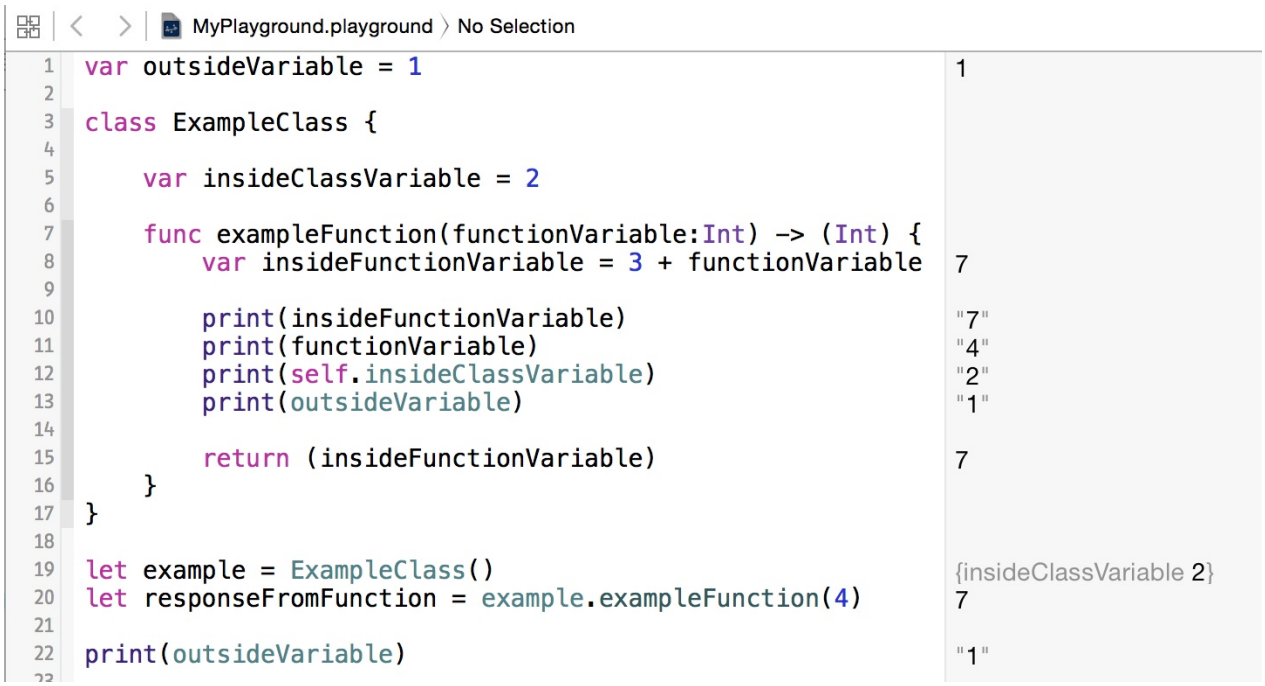
```
    var objectAge:Int
    var spouse:ComplexObject?

    init(name:String, type:String, age:Int) {
        self.objectName = name
        self.objectType = type
        self.objectAge = age
    }

    func becomeMarried(partner:ComplexObject) -> () {
        self.spouse = partner
    }
}

let me = ComplexObject(name: "Jeff", type: "Human", age: 34)
me.objectType

let wife = ComplexObject(name: "Katie", type: "Human", age: 32)
me.becomeMarried(wife)
wife.objectType
```

Scope

Scope of a variable refers to where a particular variable's binding is valid. For example, open Playgrounds and paste the following. You can trace the scope in the right-hand panel.

```
var outsideVariable = 1

class ExampleClass {

    var insideClassVariable = 2

    func exampleFunction(functionVariable:Int) -> (Int) {
        var insideFunctionVariable = 3 + functionVariable

        print(insideFunctionVariable)
        print(functionVariable)
        print(self.insideClassVariable)
        print(outsideVariable)

        return (insideFunctionVariable)
    }
}

let example = ExampleClass()
let responseFromFunction = example.exampleFunction(4)

print(outsideVariable)
```

And the output from the right-pane should be similar to this:



References

Swift Programming Guide - https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

Scope (Programming) - http://en.wikipedia.org/wiki/Scope_(computer_science)

The Swift Programming Language - A Swift Tour https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/GuidedTour.html

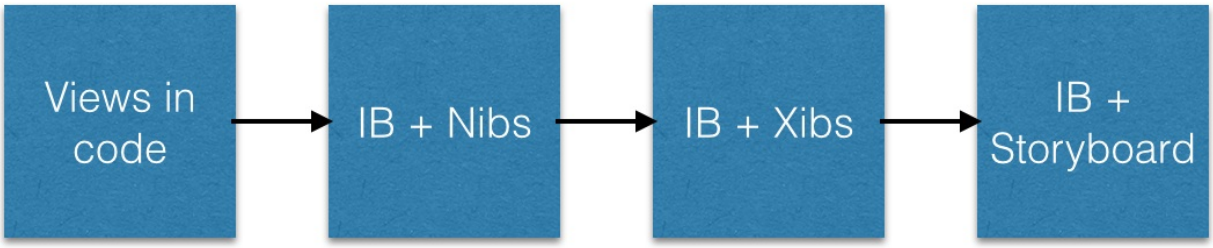
More Interface Builder

The Interface Builder has a variety of tools that allow you to build amazing experience in your app. This section will talk about how to combine IB with source code.

Since we are jumping into iOS now, it's important to realize a little of where we once where.

History

Once upon a time, there wasn't an Interface Builder and view programming was done in directly code.



Then along came something called Nibs. Nibs were files that allowed the programmer to accomplish visual editing and were somewhat complicated and not easy to edit outside of Interface Builder (or IB).

Then XIBs came along.

XIBs

Even though Apple pushes the use of Storyboards, XIBs are still available and are commonly used for reusable views that occur across multiple storyboards or projects.

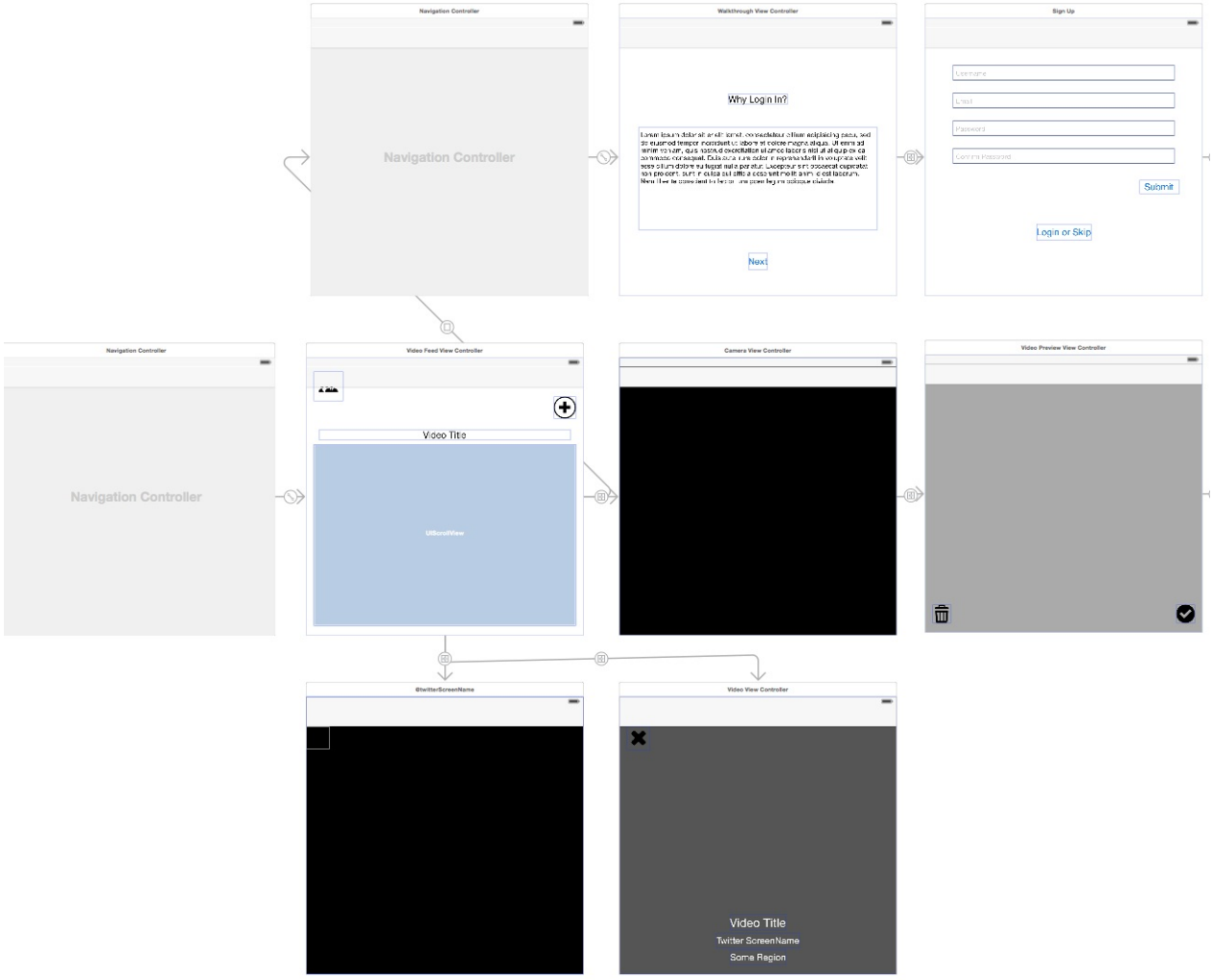
Xibs showed up later. They were more refined and featured XML behind the scenes so you could (if you wanted) view the files with a text editor.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="5056" systemVersion="13D65" targetRuntime
    <dependencies>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3733"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner" customClass="PLSmallEngagementRadial">
            <connections>
                <outlet property="descriptionLabel" destination="mEO-Dp-ZrE" id="eBD-08-0OC"/>
                <outlet property="percentLabel" destination="2zC-Af-ldS" id="uWd-2m-uso"/>
                <outlet property="radialViewHighlight" destination="3gt-Me-FVX" id="XOg-jc-GNn"/>
                <outlet property="view" destination="iN0-l3-epB" id="u0U-jN-n1f"/>
                <outletCollection property="viewsToMakeRegular" destination="2zC-Af-ldS" id="jgA-nJ-fYS"/>
                <outletCollection property="viewsToMakeRegular" destination="mEO-Dp-ZrE" id="WKs-Pu-26h"/>
            </connections>
        </placeholder>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
```

Finally, Storyboards came along.

Storyboards

Last, we have Storyboards - a collection of Xibs within one single file (also XML).



Storyboards is where we will be spending the majority of our time. They give an overview of several XIB-like files within one view. They are helpful for seeing entire sections of your UI at a glance.

Building UI isn't complete with making some connection back into your code.

Connections

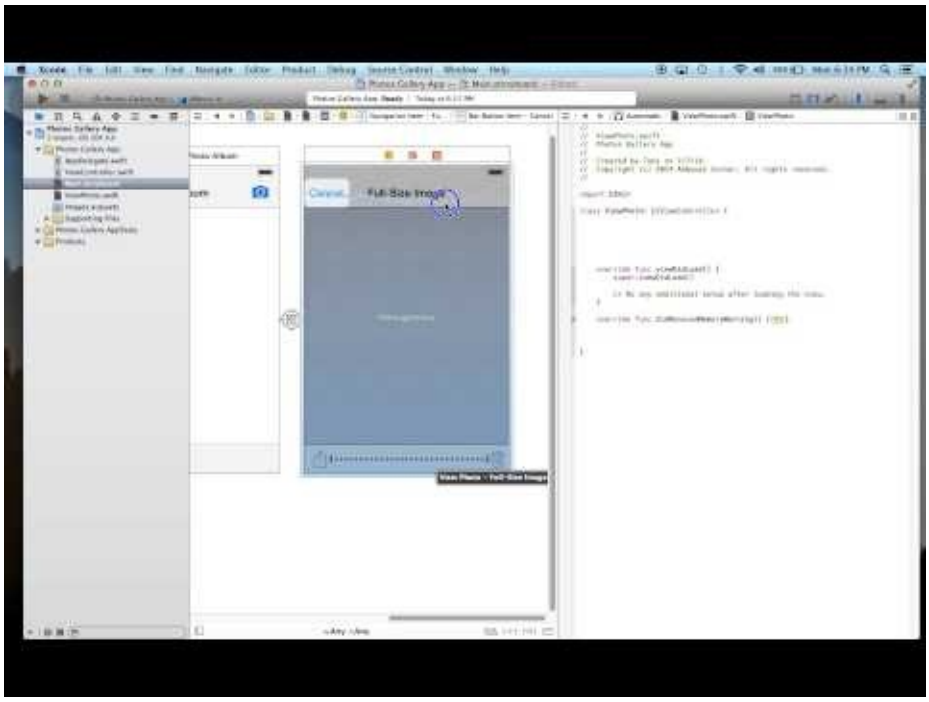
Making connections from the Storyboard into your View Controller code involves dragging outlets from the Utility pane to your control. Make sure you have the parent View Controller selected in the Document Outline.

To begin, you'll need to use your View Controller code and create outlets and actions. Swift uses a special character to denote objects and functions that are available within Interface Builder. That is the @ **symbol.**

```
import UIKit

class YourViewController: UIViewController {
    @IBOutlet var userName:UITextField!
    @IBOutlet var userPassword: UITextField!

    /////

    @IBAction func submitButtonPressed:(sender: AnyObject) {

        // perform some action
        // perhaps with self.userName or self.Password
    }
}
```

Once we have the outlets, we can enter IB, using the Utility pane. We will demonstrate this with a video from YouTube.

[Video] Hooking up outlets and actions with Interface Builder



References

Working with Interface Builder - https://developer.apple.com/xcode/interface-builder/

What's new in Interface Builder - https://developer.apple.com/videos/wwdc/2014/?include=411#411

Deep Dive with Swift

Last week, you started to get comfortable with Swift, concepts that distinguish good developers and were able to work with IB and hook into your code. Well, this week, we're going to go deeper with Swift so buckle up and let's check out:

1. Object Oriented Programming
2. Common iOS Controls
3. Common View Controllers

References

Object Oriented Programming with Swift - http://blog.codeclimate.com/blog/2014/06/19/oo-swift/

Class and Structs - https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

Structs, and tuples, and enums, oh my! - https://medium.com/swift-programming/structs-and-tuples-and-enums-oh-my-1b97f82a7339

Object Oriented Programming

A tenant of good source code is something that is reusable and easy to read. If you can create something once and reuse it, you can:

1. Write less code
2. Have an easier time maintaining your code base
3. Reuse your code in other projects

Writing code using the principles of OOP will allow you to accomplish the above tasks while maintaining logical separation of roles between your objects.

> Object oriented programming (**OOP**) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. [1]

Definitions

Let's frame out a few terms that we will use to describe Object Oriented Programming and associated concepts.

- **Base Class** - Parent class. For example, we are all human. Human would be our base class.
- **Class** - A definition of a set of properties and functions that are related.
- **Inheritance** - When a class inherits from a base class, it also inherits all of its public properties and functions. A human typically consists of a set of organs plus functions such as sleep and wake.
- **Instance** - A single occurance of a class.
- **Object** - A class that has been created and initialized.
- **Object Oriented** - A programming concept where collections of objects are used to define an application.
- **OOP** - An acronoym referring to Object Oriented Programming
- **Protocol** - A set of functions and properties that a class will need to implement. Differs from inheritance in that the implementation has not yet been defined. Now that we have a vocabulary, let's move into some detail.
- **Superclass** - Another term for Base Class.

Details

Let's talk about the ways Swift can help you rock out OOP.

Defining a class

And organize your code into reusable objects with a class:

```
class ExampleClass {

    var memberVariable:String

    init() {
        self.memberVariable = "Initial value."
    }

    func exampleFunction(thisIsFun:Bool) -> String {

        var returnValue = "No, this isn't fun."
        if thisIsFun {
            returnValue = "Yes, this is fun!"
        }

        return returnValue
    }
}
```

```
var exampleClass = ExampleClass()
print("We created a class and its member variable contains \(exampleClass.memberVariable) )"
```

Inheritance

A class can inherit another class and therefore capture its methods as its own. A subclass inherits the superclass.

```
import UIKit

class YourViewController: UIViewController {

}
```

In this simple example above, we are using Swift to describe YourViewController which inherits all the methods of UIViewController. When we inherit the superclass, we can not only obtain functionality but can override that functionality as we want.

```
import UIKit

class YourViewController: UIViewController {
    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)
        view.backgroundColor = UIColor.blackColor()
    }
}
```

It is important to note that when calling functions from the superclass, it is often the case you will be wanting to call super of the same method. This allows for the super's function to also be called. In the above example, we are calling super.viewWillAppear(animated) to allow for the UIViewController's method (viewWillAppear) to also be called and perform any necessary initialization.

Protocols

Protocols are similar to inheritance in that we obtain a set of functions and properties through declaration. They differ in that you are required to define these methods when implementing the protocol.

```
import Foundation

// Defines a protocol
protocol ExampleProtocol {

    var maxValue:Int {get set}

    func generateRandomNumber() -> Int
}

// Defines the class which implements the protocol
class ExampleImplementation: ExampleProtocol {

    var maxValue:Int

    init (initialMaximumValue:Int) {
        maxValue = initialMaximumValue
    }

    func generateRandomNumber() -> Int {
        return random() % maxValue;
    }
}

// Example usage
let example = ExampleImplementation(initialMaximumValue: 10)
example.generateRandomNumber()
```

A real-world example, extending our previous, would be to conform YourViewController to the UITableViewDataSource protocol:

```
import UIKit

class YourViewController: UIViewController, UITableViewDataSource {

    //////////
    // Inheritance
    override func viewWillAppear(animated: Bool) {
        view.backgroundColor = UIColor.blackColor()
    }

    //////////
    // Protocol
    // Required: filling out protocol functions

    // protocol definition 1
    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    // protocol definition 2
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 1
    }

    // protocol definition 3
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

        var tableCell = UITableViewCell()
        return tableCell
    }
}
```

Polymorphism

Polymorphism allows you to write more generic code that works with families of objects, rather than writing code for a specific class. [2]

```
var x = UILabel()
var y = UITextView()
var z = UIButton()

func updateTheTag(control:UIView, updatedTag:Int) {
    control.tag = updatedTag
}

updateTheTag(x, 1)
updateTheTag(y, 2)
updateTheTag(z, 3)
```

In this above example, UILabel, UITextView and UIButton all derive from the base class of UIView and that base class defines a tag property. Although each of these classes may have a specific purpose, they all can access their base classes properties in a generic way.

Swift types

Enumerations

An enumeration defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code. [3]

```
enum CompassPoint {
    case North
    case South
```

```
    case East
    case West
}

// Int type
enum AgesOfChildren:Int {
    case John = 20
    case Jacob = 25
    case Jingleheimer = 30
    case Schmidt = 35
}

// String type
enum SectionHeader : String {
    case FirstColumn = "Column 1"
    case SecondColumn = "Column 2"
    case ThirdColumn = "Column 3"
    case FourthColumn = "Column 4"
    case FifthColumn = "Column 5"
}
```

Structures vs Classes

Apples documentation goes into detail defining structures and classes. That resource is listed as number 5 below, and is summarized here.

> Classes and structures are general-purpose, flexible constructs that become the building blocks of your program's code.
>
> Similarities:
>
> - Define properties to store values
> - Define methods to provide functionality
> - Define subscripts to provide access to their values using subscript syntax
> - Define initializers to set up their initial state
> - Be extended to expand their functionality beyond a default implementation
> - Conform to protocols to provide standard functionality of a certain kind

Structures

```
struct Beer {
    var name:String
    var ounces:Float
    var alcoholContent:Float
    var bitterness:Int
}

var stoneIPA = Beer(name:"Stone IPA", ounces:22.0, alcoholContent:6.9, bitterness:85)
```

[Video] Value versus References

When talking about classes there is one key difference and that is **value** versus **reference**. Structures are passed by value where classes are passed by reference.

Classes

Now classes are similar to structures with a few exceptions, including being passed as reference types. Classes also allow

for inheritance, which enables us to define more complex object schemes, such as the following:

```
class Human {
    var name:String
    var type:String
    var age:Int
    var spouse:Human?

    init(name:String, type:String, age:Int) {
        self.name = name
        self.type = type
        self.age = age
    }

    func becomeMarried(spouse:Human) -> () {
        self.spouse = spouse
    }
}

let me = Human(name: "Jeff", type: "Human", age: 34)
me.spouse

let wife = Human(name: "Katie", type: "Human", age: 32)
me.becomeMarried(wife)
wife.spouse

class Baby : Human {
    var diaperChangesToday:Int

    override init(name: String, type: String, age: Int) {
        diaperChangesToday = 0
        super.init(name: name, type: type, age: age)
    }

    func incrementDiaperChanges() -> String {
        diaperChangesToday++
        return("\(diaperChangesToday) diapers? C'mon now!")
    }
}

let newBaby = Baby(name: "Raleigh", type: "Human", age: 0)
newBaby.incrementDiaperChanges()
newBaby.incrementDiaperChanges()
newBaby.incrementDiaperChanges()
```

References

An Introduction to Object-Oriented Programming - http://blog.codeclimate.com/blog/2014/06/19/oo-swift/

[1] Object-oriented Programming http://en.wikipedia.org/wiki/Object-oriented_programming

[2] Swift Programming 101: Inheritance & Polymorphism http://www.iphonelife.com/blog/31369/swift-programming-101-inheritance-polymorphism

[3] Enumrations
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Enumerations.html

[4] Protocols
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html#//apple_ref/doc/uid/TP40014097-CH25-XID_402

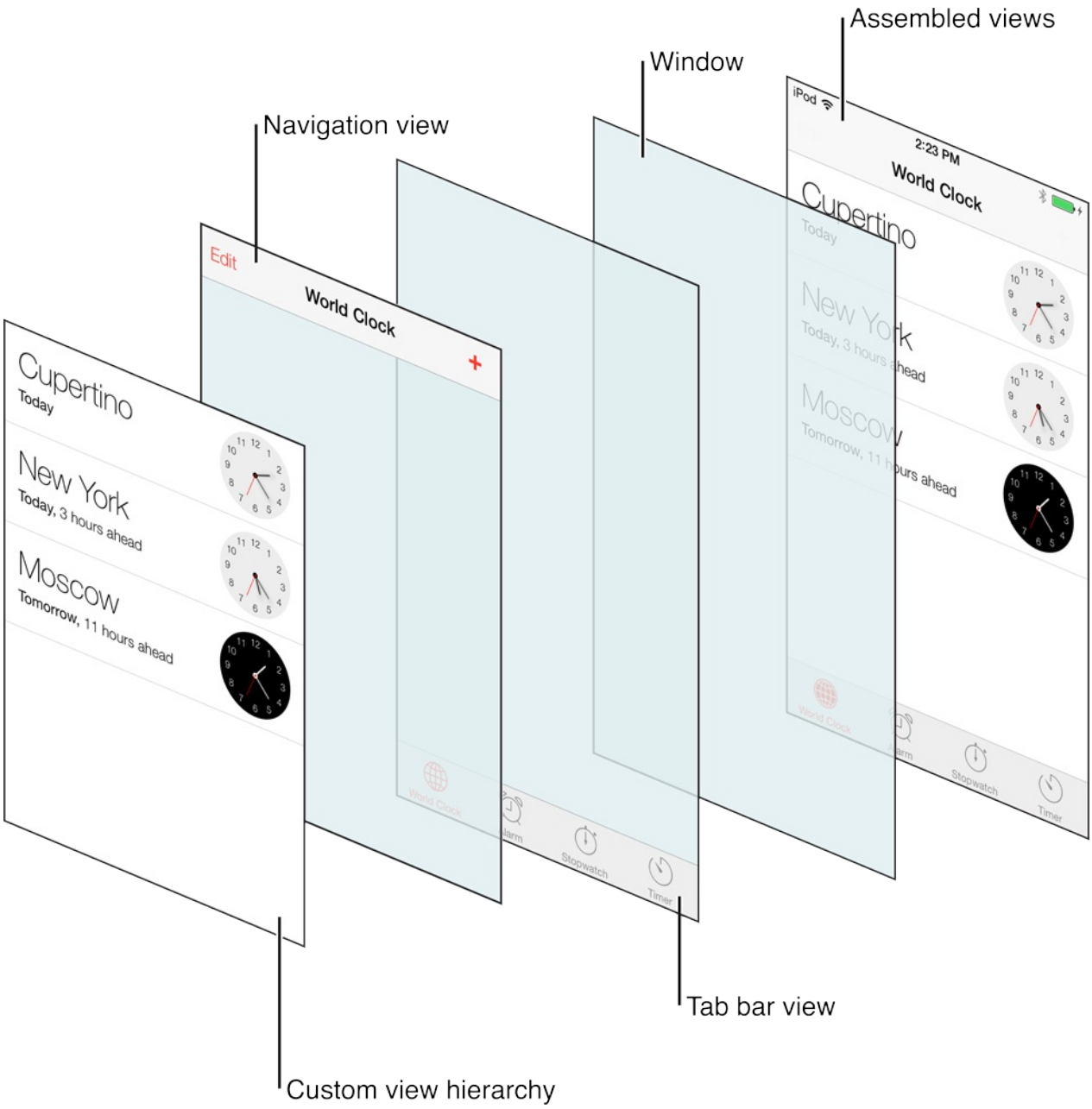
[5] Structures and Classes
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

[6] Protocols
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

UIKit

Apple includes a variety of controls that serve very well in the creation of UI through Interface Builder. These controls are found in the Object Library within Xcode

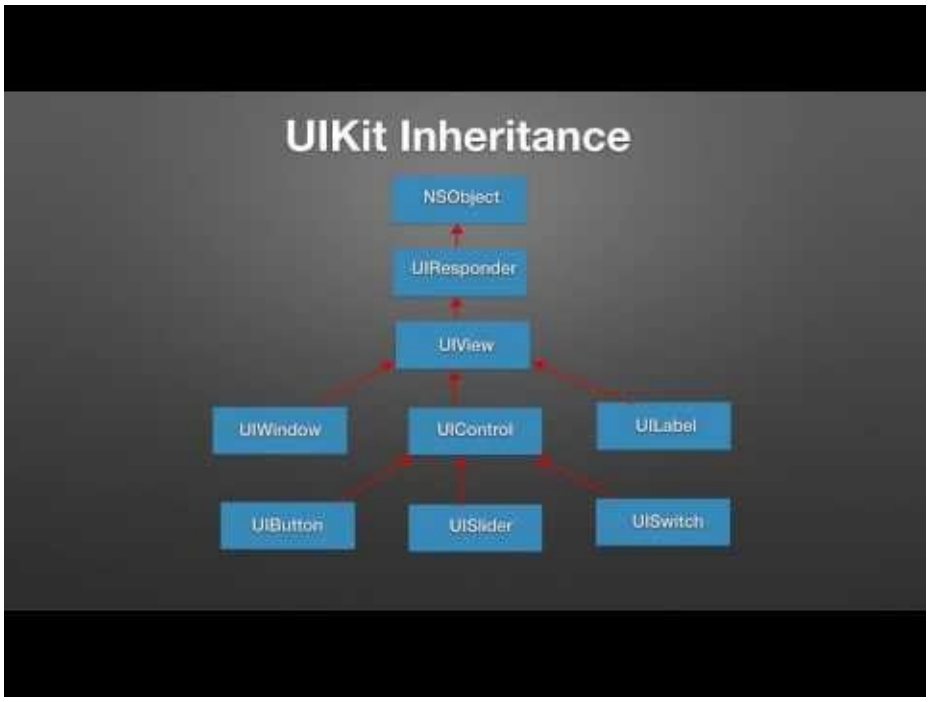
Overview



Apple provides excellent overview documentation on UIKit here

> Views are the building blocks for constructing your user interface. Rather than using one view to present your content, you are more likely to use several views, ranging from simple buttons and text labels to more complex views such as table views, picker views, and scroll views. [1]

[Video] Introduction to UIKit

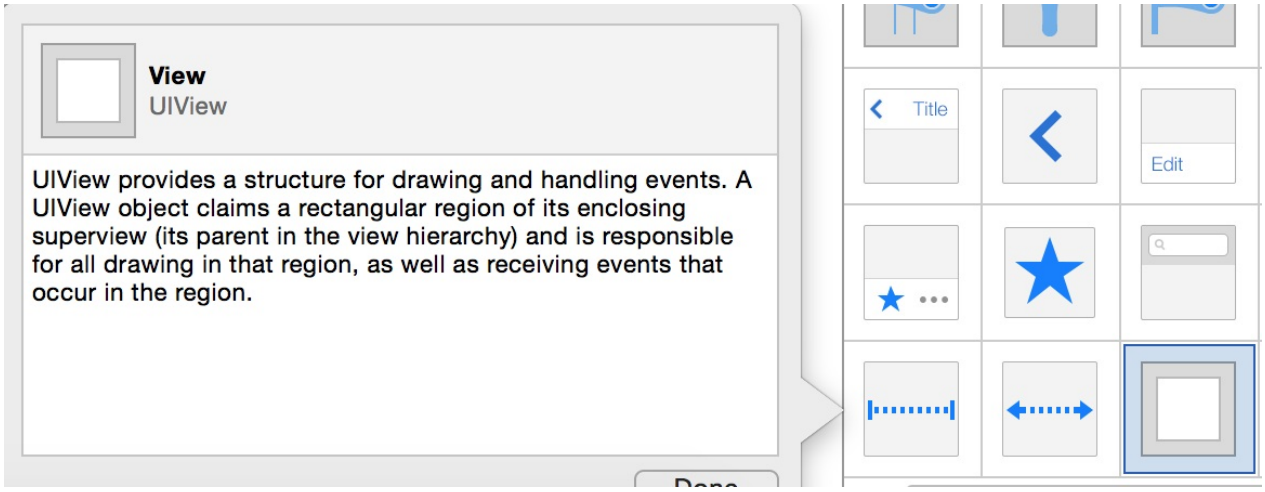


(http://www.youtube.com/watch?v=DFsENma-PAk

Common Controls

UIView

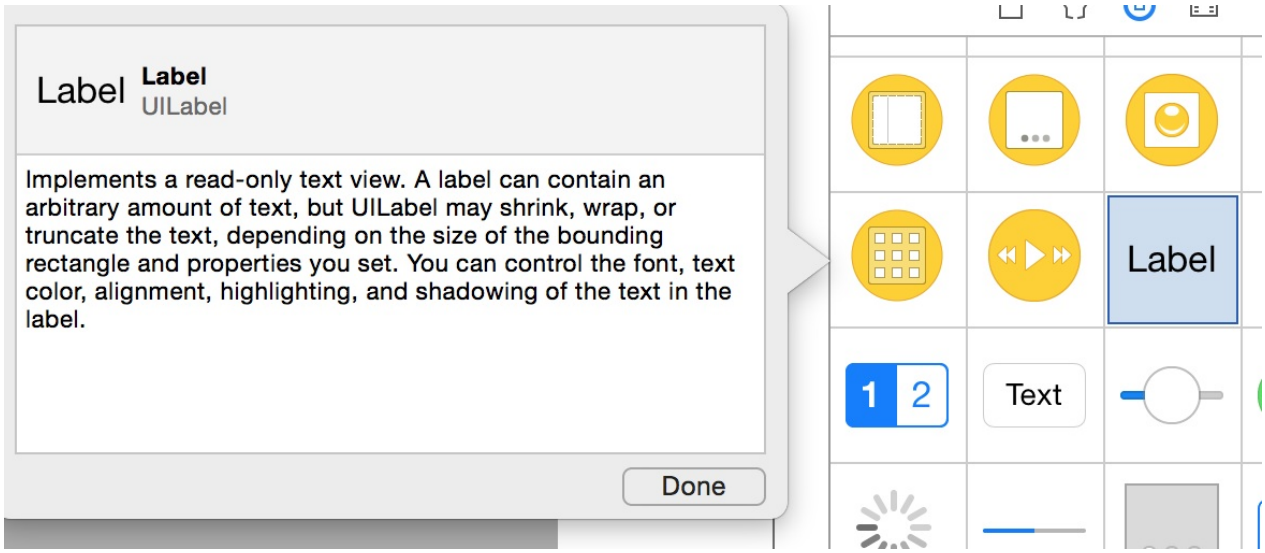
UIView is the basis for all views within iOS. Buttons, labels, tables are all subclasses of UIView.



> The UIView class defines a rectangular area on the screen and the interfaces for managing the content in that area.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/

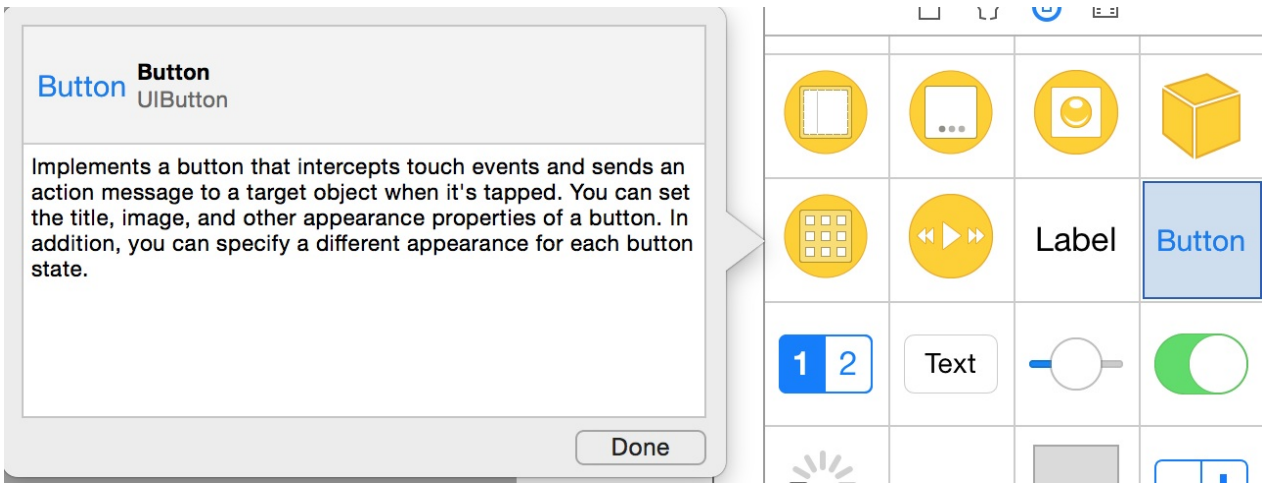
UILabel



> 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.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILabel_Class/

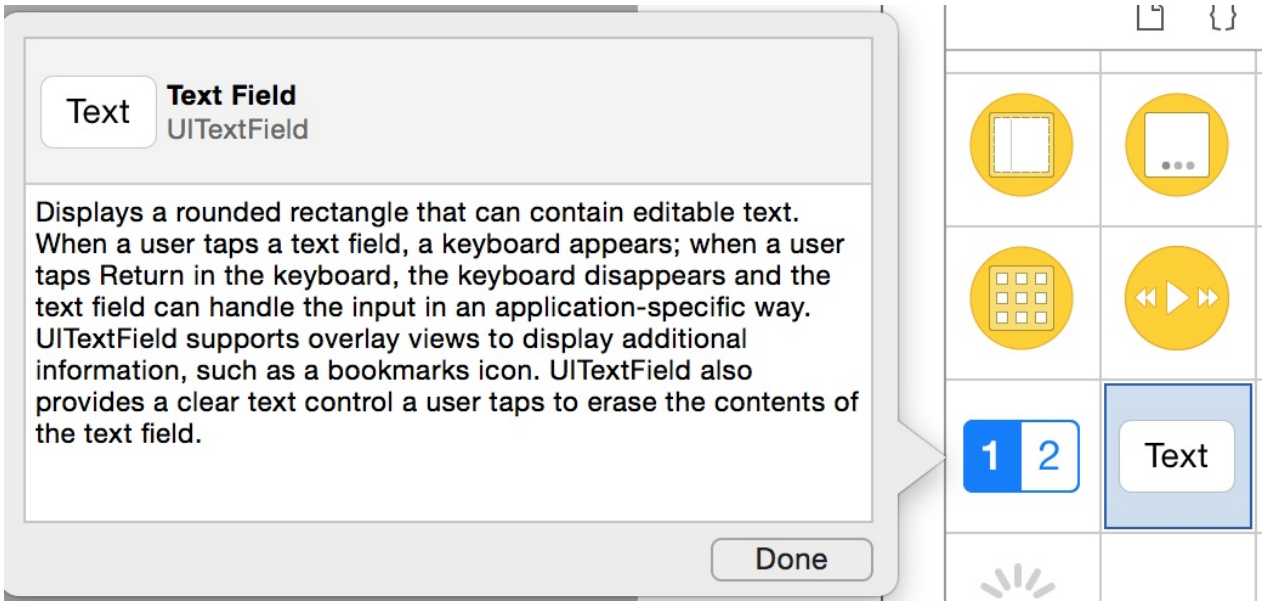
UIButton



> A button intercepts touch events and sends an action message to a target object when tapped.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html

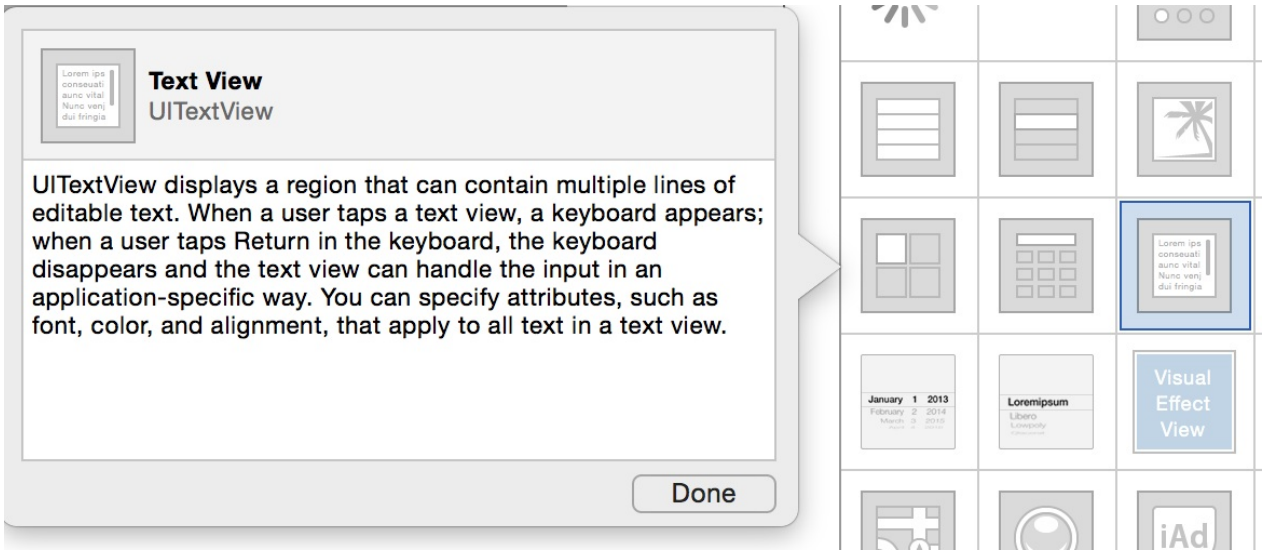
UITextField



> A UITextField object is a control that displays editable text and sends an action message to a target object when the user presses the return button

Reference: https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UITextField_Class/index.html

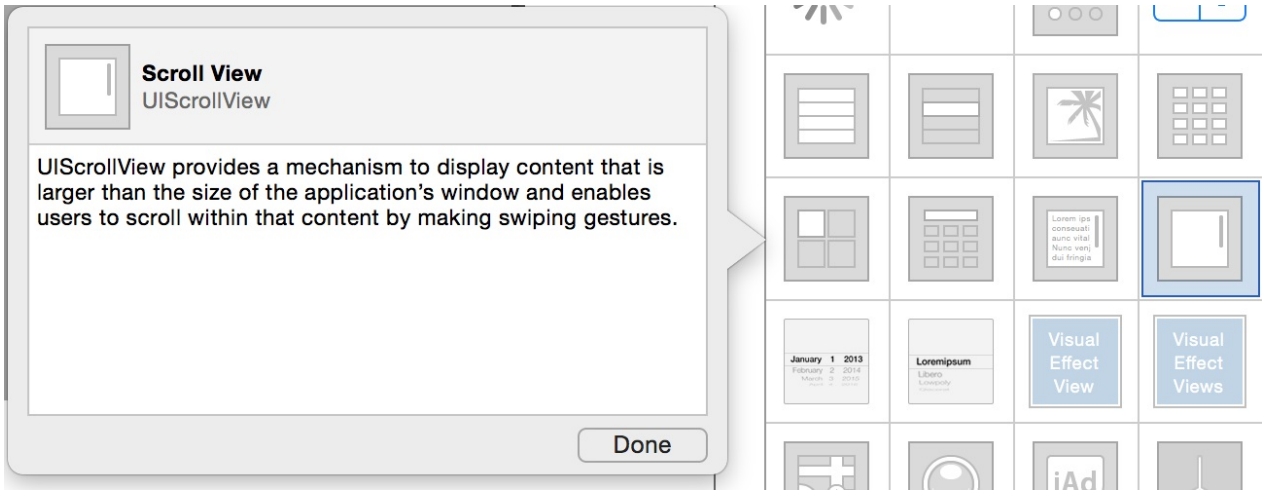
UITextView



> The UITextView class implements the behavior for a scrollable, multiline text region

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextView_Class/index.html

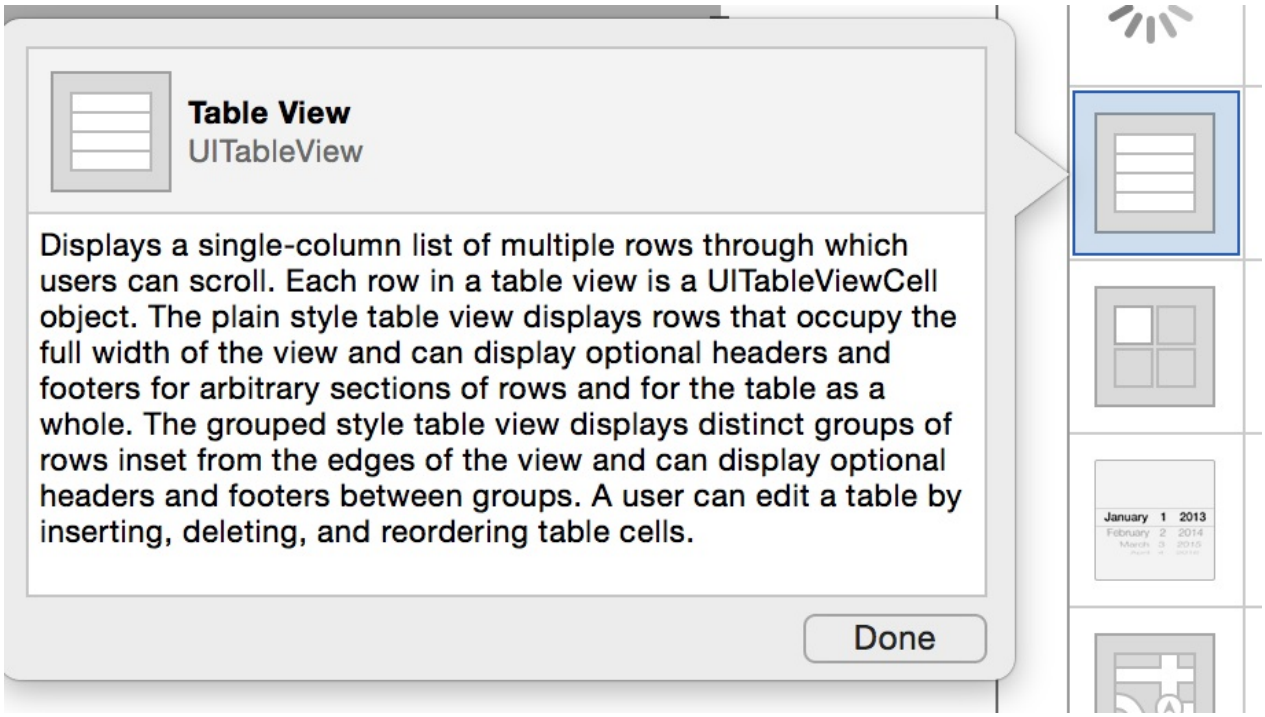
UIScrollView



> The UIScrollView ... enables users to scroll within that content by making swiping gestures, and to zoom in and back from portions of the content by making pinching gestures

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScrollView_Class/index.html

UITableView

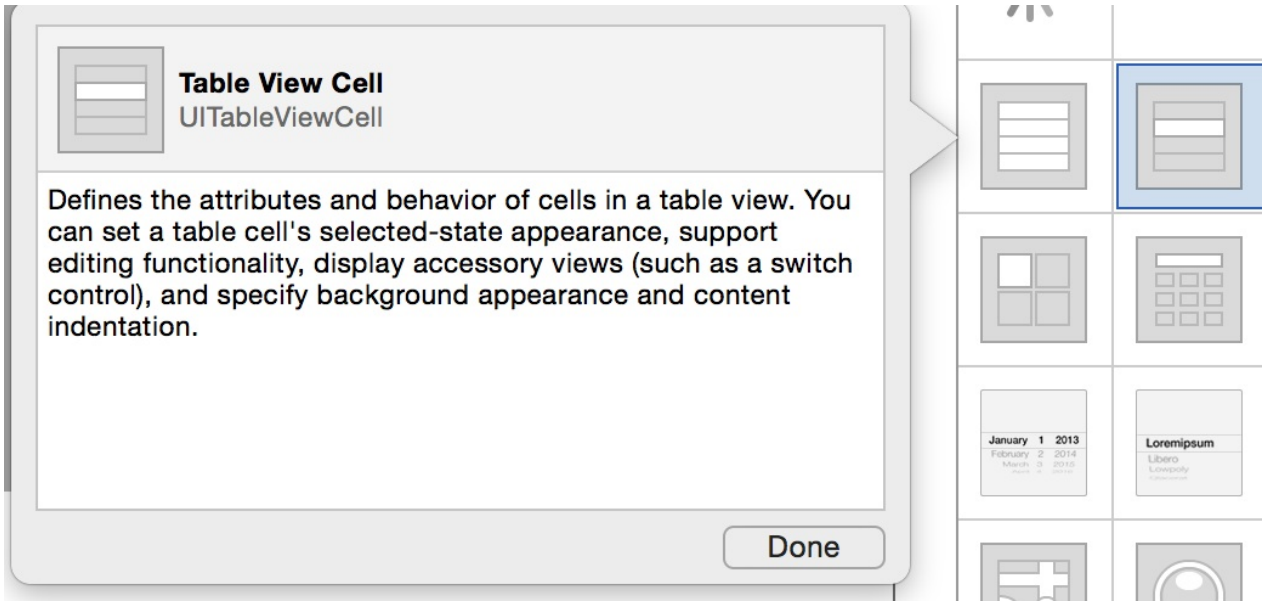


The UITableView allows for a vertical, scrolling list of cells to be displayed.

> A table view is made up of zero or more sections, each with its own rows. Sections are identified by their index number within the table view, and rows are identified by their index number within a section.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableView_Class/index.html

UITableViewCell

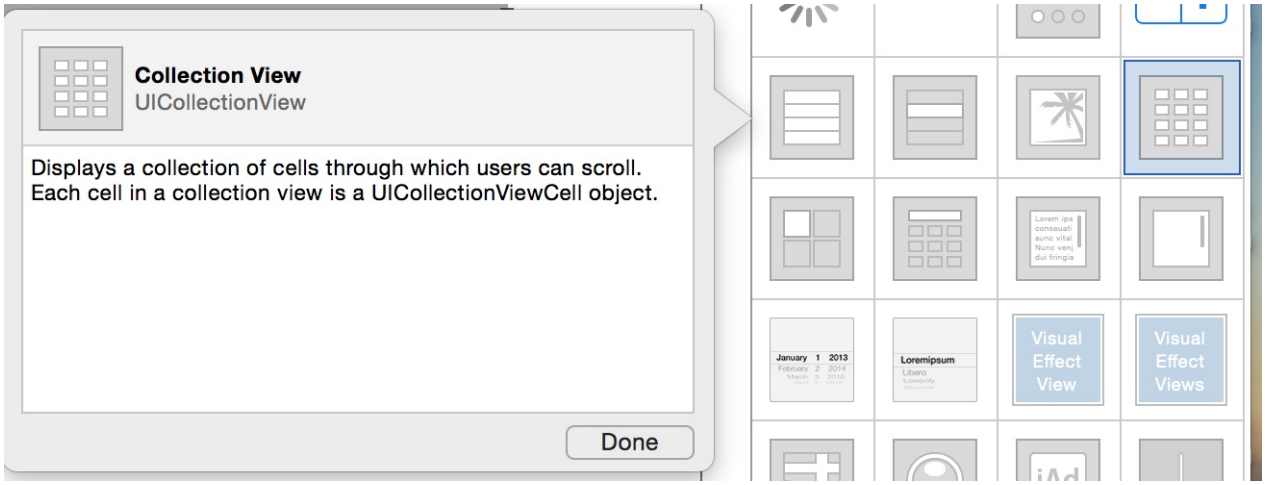


Manages content within UITableView.

> Includes properties and methods for setting and managing cell content and background (including text, images, and custom views), managing the cell selection and highlight state, managing accessory views, and initiating the editing of the cell contents

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/

UICollectionView

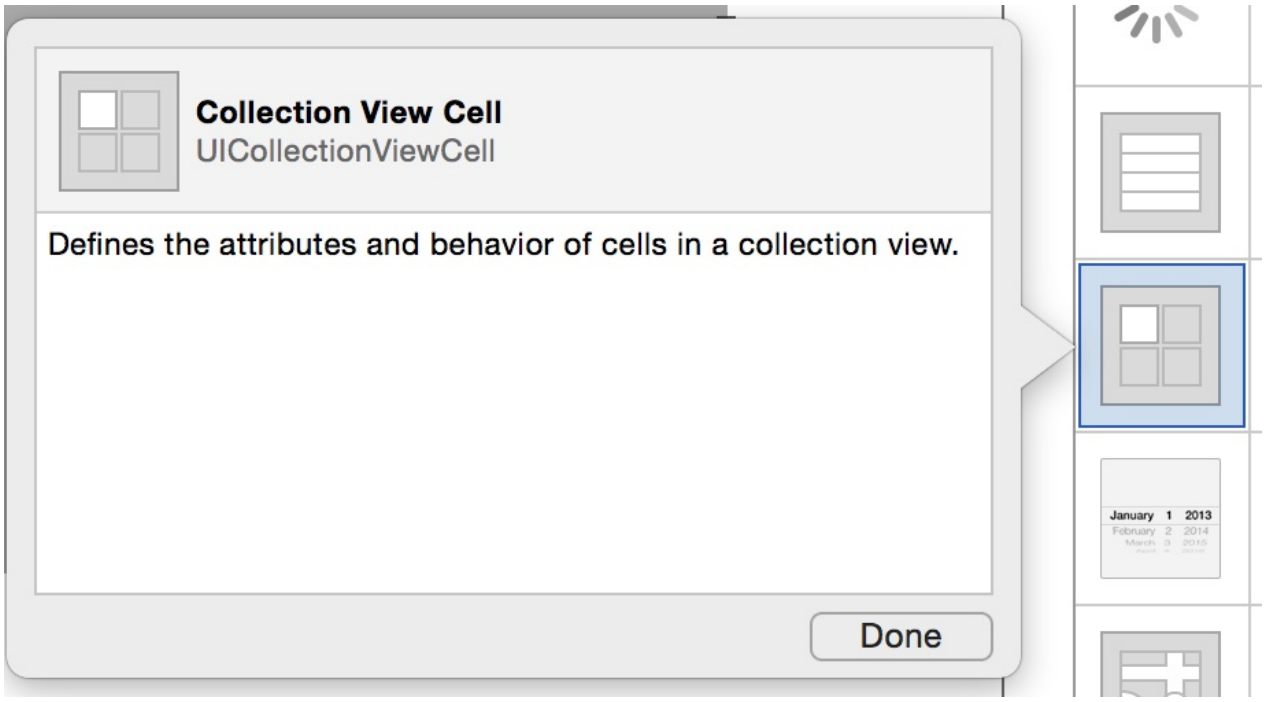


UICollectionViews are similar to UITableViews in that they display a series of cells. The extend the limitations of UITableView:

> Collection views support customizable layouts that can be used to implement multi-column grids, tiled layouts, circular layouts, and many more. You can even change the layout of a collection view dynamically

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UICollectionView_class/index.html

UICollectionViewCell



> A UICollectionViewCell object presents the content for a single data item when that item is within the collection view's visible bounds.

Reference:
https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UICollectionViewCell_class/index.html

---

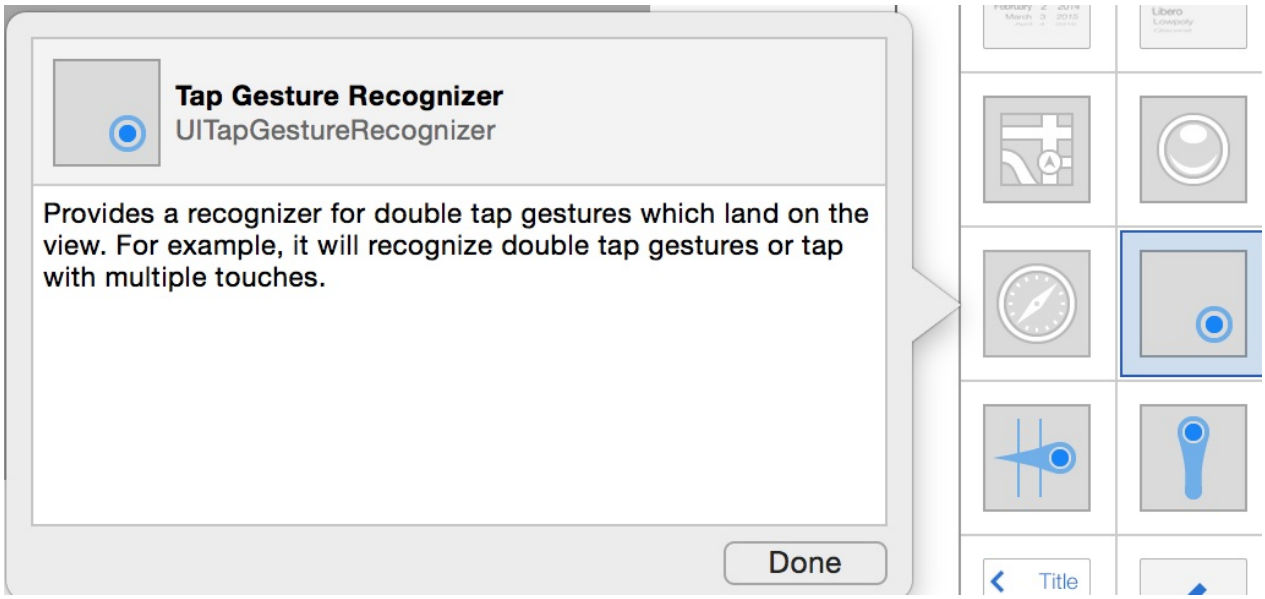
Common Gestures

Below are three of the most common gesture recognizers Apple provides in UIKit. They are all derived from a common base class of UIGestureRecognizer.

[Video] iOS Tutorial - UIGestureRecognizer

https://www.youtube.com/watch?v=KnfQRy6xOPk

UITapGestureRecognizer



> [UITapGestureRecognizer] looks for single or multiple taps. For the gesture to be recognized, the specified number of fingers must tap the view a specified number of times.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITapGestureRecognizer_Class/index.html

UIPanGestureRecognizer

> [UIPanGestureRecognizer] looks for panning (dragging) gestures. The user must be pressing one or more fingers on a view while they pan it.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPanGestureRecognizer_Class/index.html

UILongPressGestureRecognizer

> [UILongPressGestureRecognizer] looks for long-press gestures. The user must press one or more fingers on a view and hold them there for a minimum period of time before the action triggers.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILongPressGestureRecognizer_Class/index.html

---

References

[1] Apple - About Views https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/

[2] Apple UIKit Documentation https://developer.apple.com/library/ios/documentation/UIKit/Reference/

[3] Designing for iOS https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

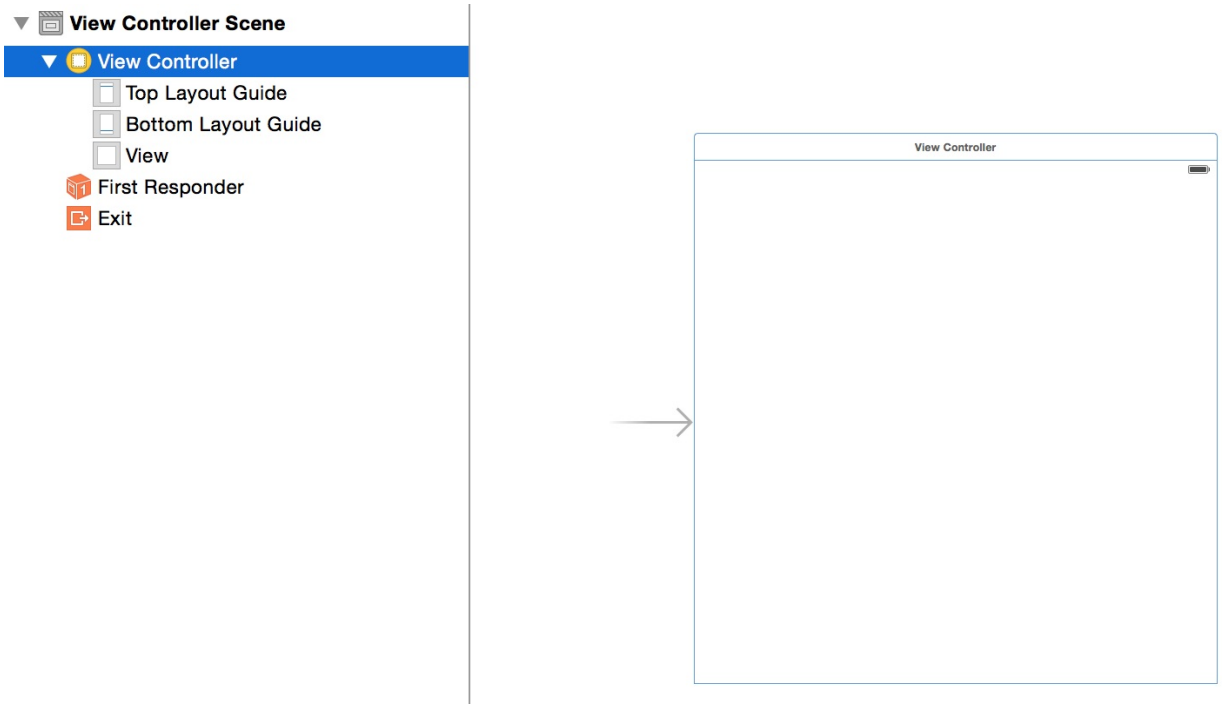
[4] Start Developing for iOS - User Interfaces
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesigningaUserInterface.html

View Controllers

Apple includes a variety of View Controllers which organize your views and provide a controller class for delegates and other interactions.

UIViewController

UIViewController is the building block for user interfaces. You will typically subclass UIViewController in the development of your own view controllers.

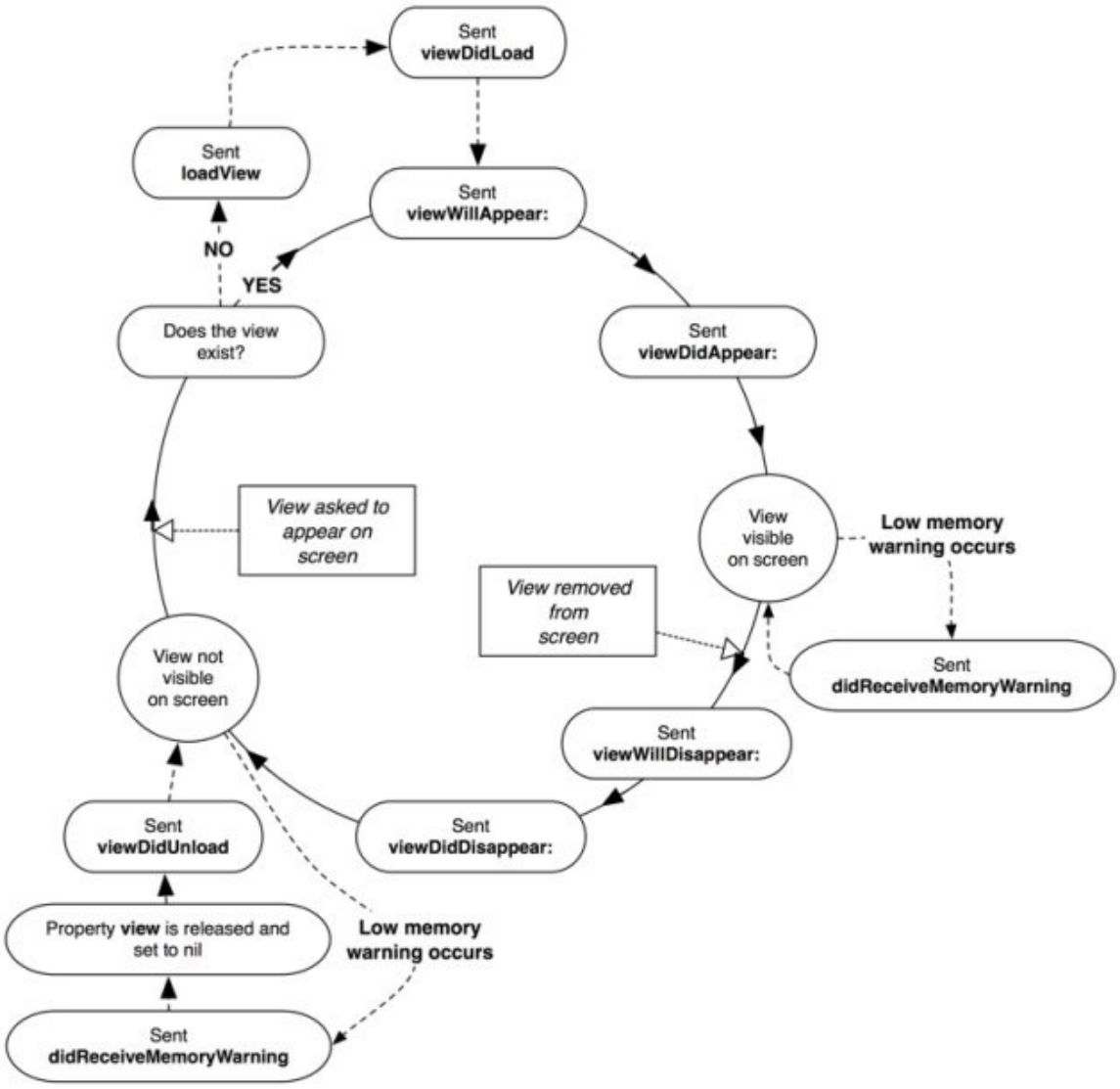


> The UIViewController class provides the fundamental view-management model for all iOS apps

https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html

View Controller Life Cycle

UIViewController provides a number of methods that you can override at various points of the view controller's life cycle.



Source: http://rdkw.wordpress.com/2013/02/24/ios-uiviewcontroller-lifecycle/

Common methods to overide

Depending on what you want to accomplish, the following methods can be overridden. These are all optional overrides.

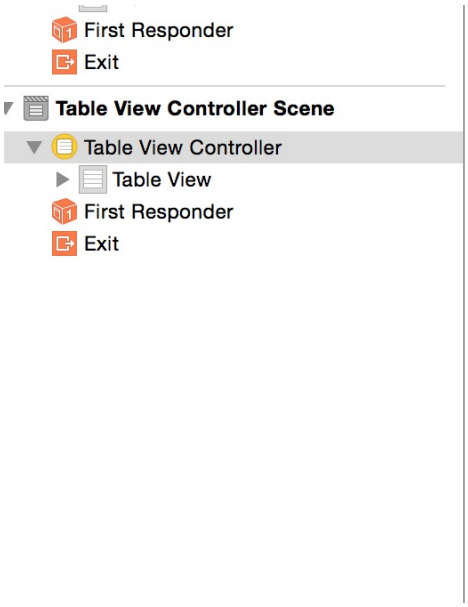
Method name	Common Use Case
viewDidLoad:	One-time setup, called when app has already loaded the XIB but has not started to display
viewWillAppear:	Called each time the view will begin to appear on screen. Setup data on screen such as labels.
viewDidAppear:	Called when the view has completed being displayed to the user. Use for custom animations on hidden views.
viewWillDisappear:	Called each time the view is about to begin disappearing. Use for more custom animations and for saving data to disk.
viewDidDisappear:	View has disappeared and view controller may be removed from memory.
didReceiveMemoryWarning	iOS will begin trying to reclaim memory. Release memory that can be recreated easily.

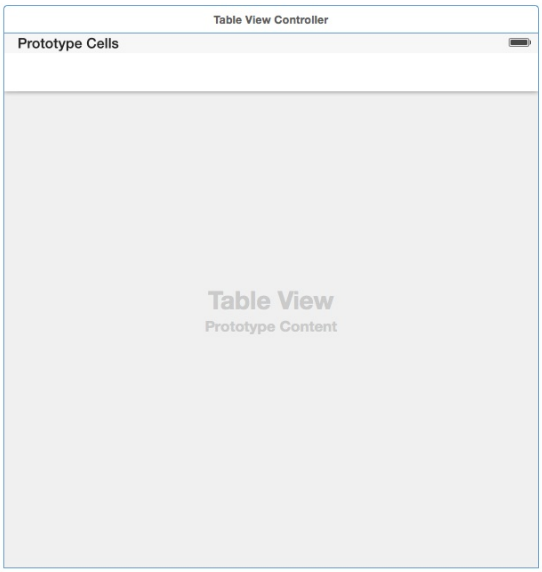
Derivatives

UIViewController is used as the base class for other common view controllers including **UITableViewController** and **UICollectionViewController**. These two additional classes will include other protocols that will need to be defined within

the implementation.

UITableViewController

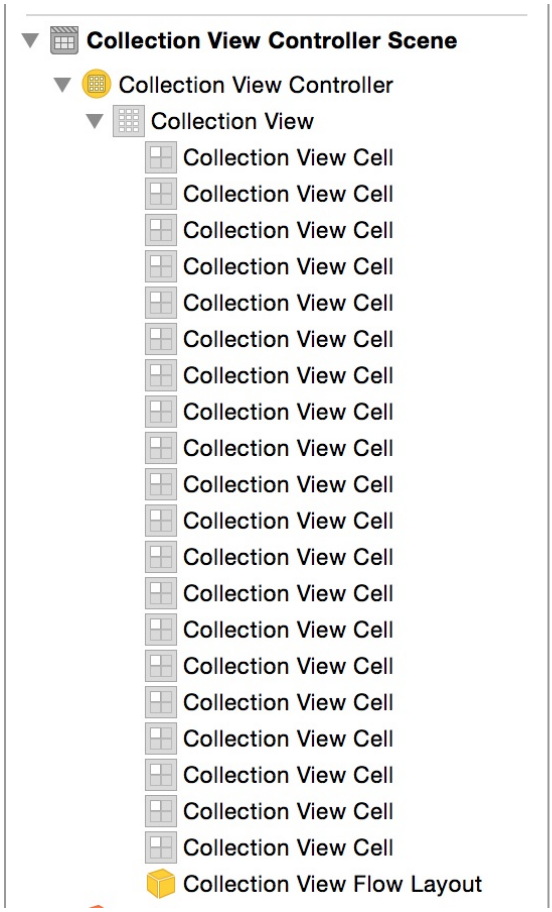


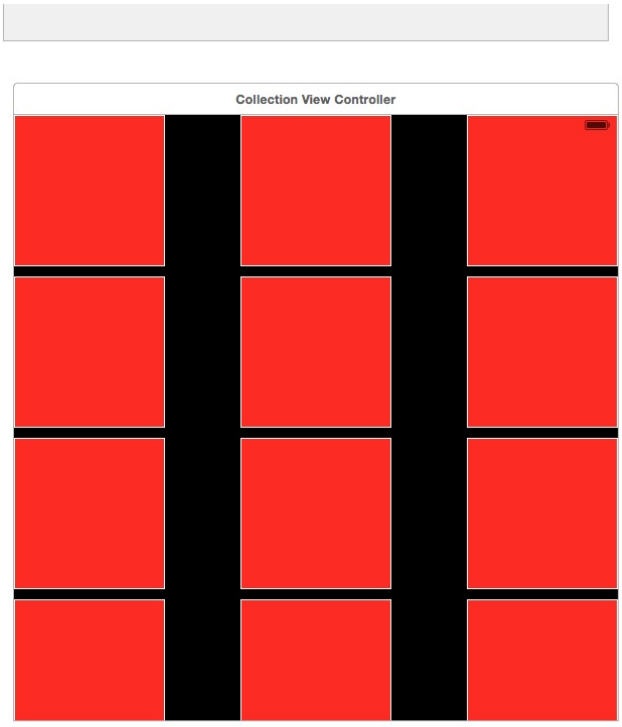


> The UITableViewController class creates a controller object that manages a table view.

https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UITableViewController_Class/index.html

UICollectionViewController





> The UICollectionViewController class represents a view controller whose content consists of a collection view

https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UICollectionViewController_clas/index.ht

ml

References

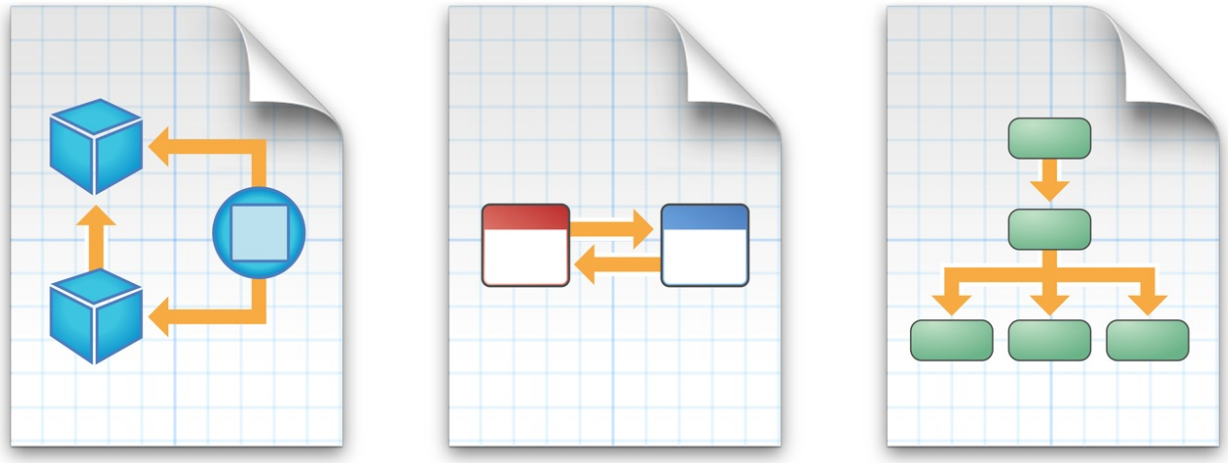
Apple UIKit Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/

View Controller Programming Guide
https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/CreatingCustomContainerViewControllers/CreatingCustomContainerViewControllers.html

Design Patterns

There are a few standard design patterns that you will use in coding mobile applications. These are not necessarily specific to Swift or Apple, but are general templates you can utilize in your programming endeavors.

- MVC - Model View Controller pattern
- Singleton - Same instance of a class for all that request it.
- Delegate - Messaging system between two separate objects
- Notification - Messaging broadcast to all listeners



The above diagram references an example of MVC, delegation and notification. From Apple's Using Design Patterns [1].

References

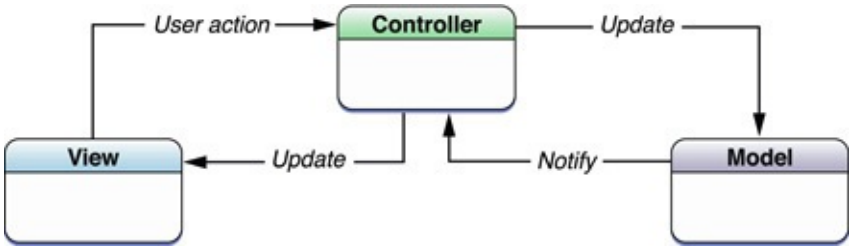
(1) Start Developing iOS apps today - Using Design Patterns
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

iOS Core Competencies - https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/

iOS Design Patterns - http://www.raywenderlich.com/46988/ios-design-patterns

Model - View - Controller

iOS templates new projects in Xcode with the MVC pattern. That is to say, they give you a separate entity for each part of the MVC pattern. Keeping each part of MVC as separate as possible allows for reusability, maintenance and clean code.



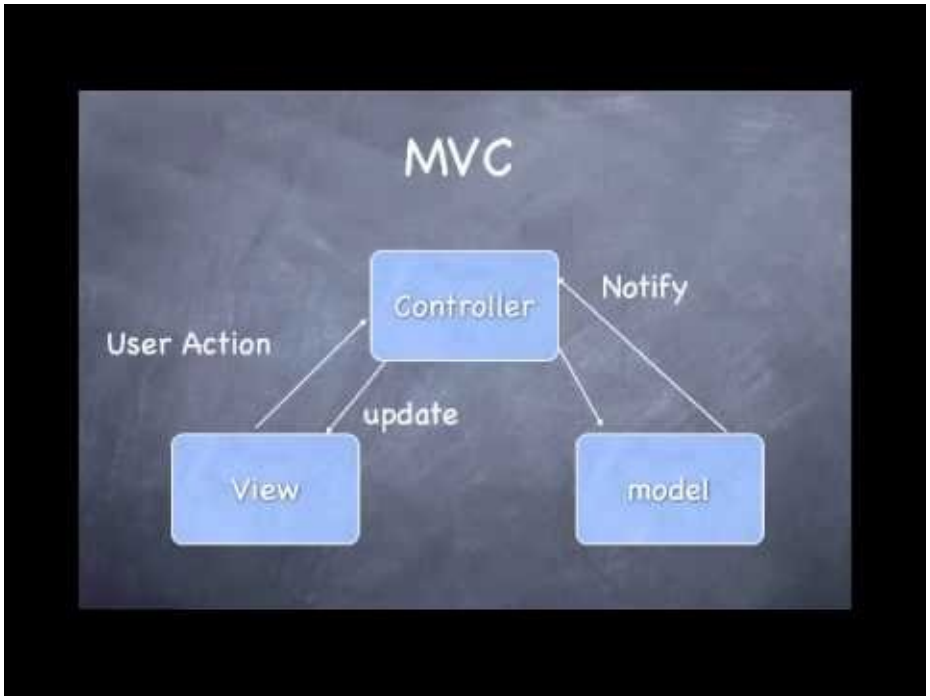
The above diagram illustrates the MVC pattern and is broken out into model, view and controller and linked between them is the action. See reference [1].

This chapter will be exploring the following topics:

1. Model - Data storage
2. View - UI as defined and created in a Storyboard
3. Controller - Controller classes which typically override a UIKit class.

[Video] Introduction to MVC

The following video serves as an introduction to the MVC pattern and provides helpful overview of the concept.



https://www.youtube.com/watch?v=Y09RvzZ1mY8

Model

A model in an iOS application can be a simple class or struct that persists in memory. It can also be a file that is written to disk, which would presist through subsequent launches of your application. CoreData is a system that Apple provides as a robust model layer in your iOS applications.

> Model objects encapsulate the data specific to an application and define the logic and computation that manipulate

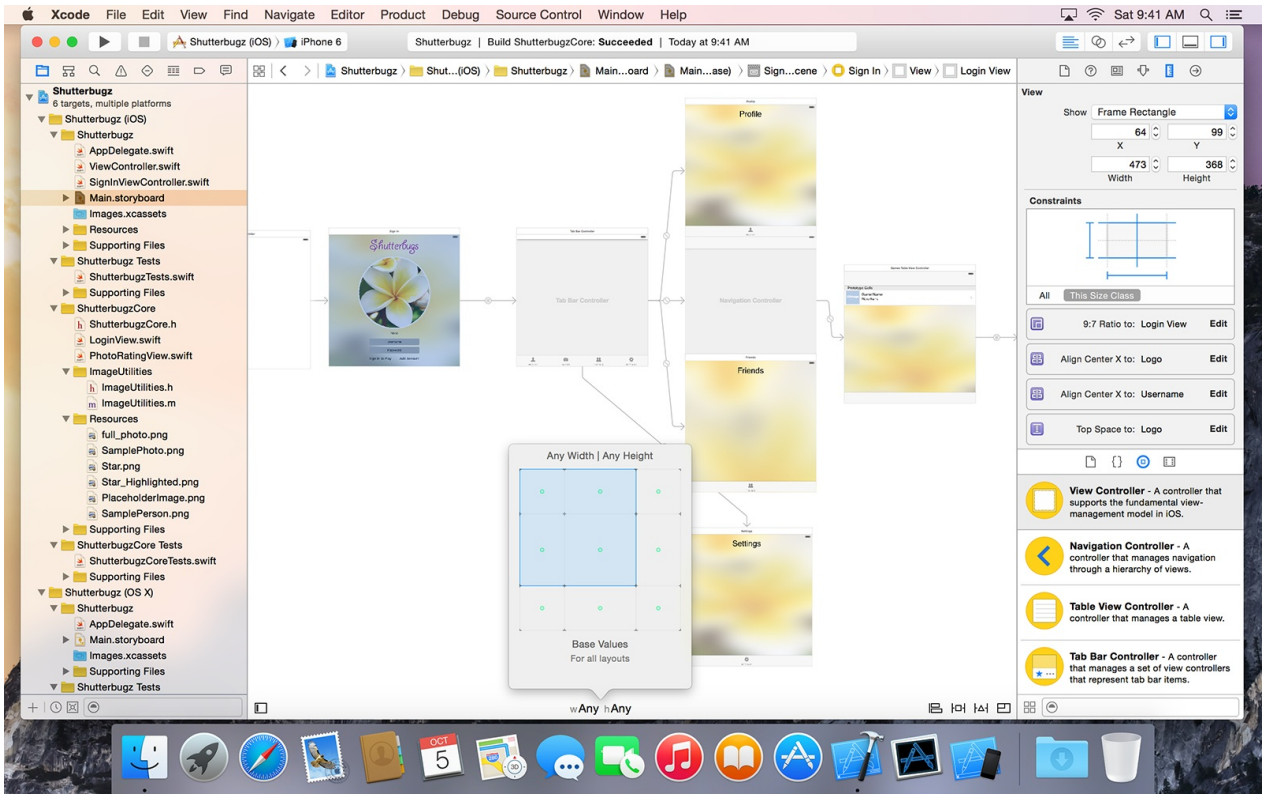
> and process that data [1]

Reference: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

View

Views are typically handled in a Storyboard, handled by Interface Builder within Xcode.

> A view object is an object in an application that users can see. A view object knows how to draw itself and can respond to user actions. [1]



Controller

Deciding what to do once your user interacts with your application is handled by the controller. In your projects, this is the subclass of UIViewControllers, UITableViewControllers, etc.

> A controller object acts as an intermediary between one or more of an application's view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. [1]

Reference: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ControllerObject.html

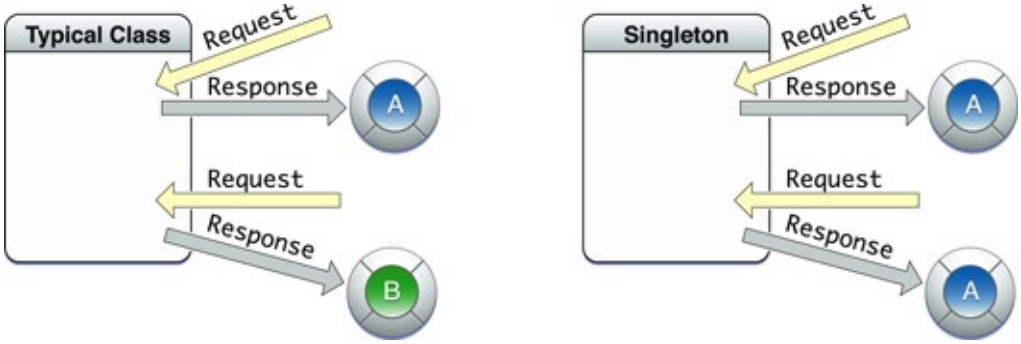
References

[1] - Cocoa Core - Model View Controller
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[2] Start Developing iOS apps today - Using Design Patterns
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

Singleton

A singleton is the same instance of a class for all that request it. Use a singleton to maintain a single point of control for your object.



> A typical class permits callers to create as many instances of the class as they want, whereas with a singleton class, there can be only one instance of the class per process [1]

An example

```
class ExampleSingleton {

    private var counter = 0

    class var sharedInstance : ExampleSingleton {
        struct Static {
            static let instance : ExampleSingleton = ExampleSingleton()
        }
        return Static.instance
    }

    func incrementCounter() -> Int {
        counter++
        return counter
    }
}

ExampleSingleton.sharedInstance.incrementCounter() // 1
ExampleSingleton.sharedInstance.incrementCounter() // 2
ExampleSingleton.sharedInstance.incrementCounter() // 3
ExampleSingleton.sharedInstance.incrementCounter() // 4
```

References

[1] Core Competency - Singleton - https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html

[2] Swift Singleton - GitHub - https://github.com/hpique/SwiftSingleton https://github.com/hpique/SwiftSingleton

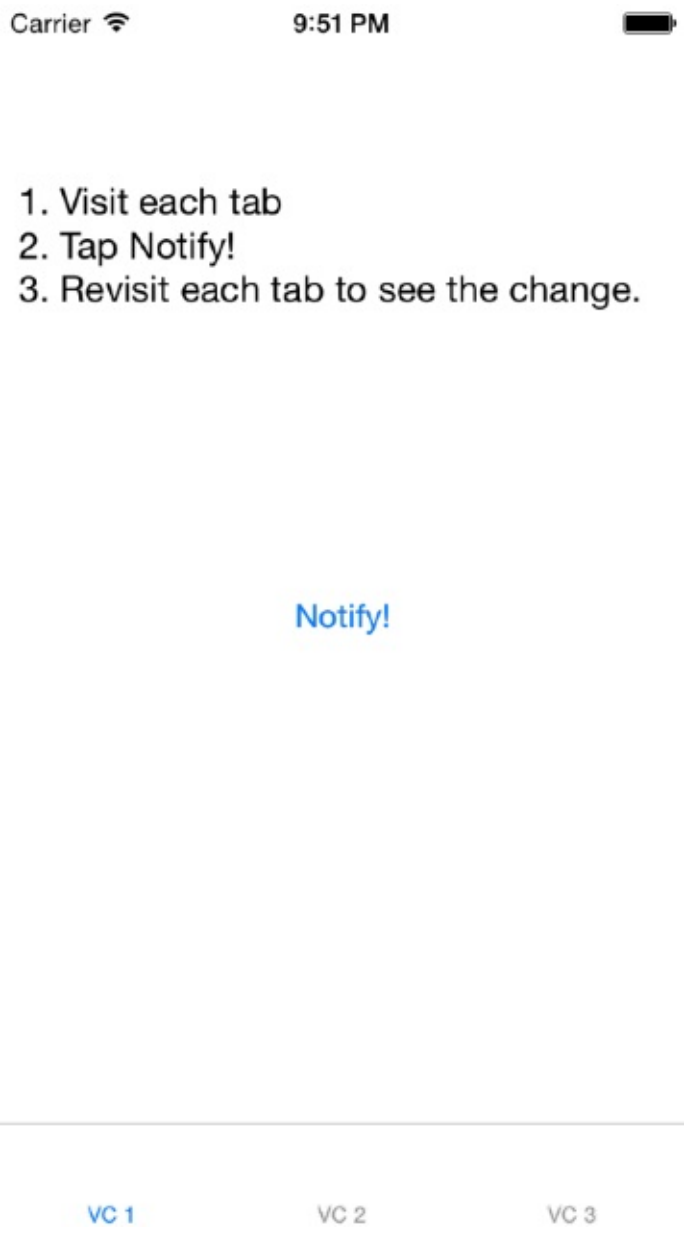
Notification

A notification is simply a broadcast of information. Similar to the way television works, a station can broadcast a television signal and you can see this broadcast by tuning your television to a particular channel. The television station doesn't know about your particular TV. The receiver, you in this example, chooses to receive that signal by tunning your TV.

Notifications in iOS can be handled by the UINotificationCenter. Notifications differ from delegates in that they are not a one-to-one connection point. Instead, they are a broadcast which objects can subscribe to.

An Example

Below is the output of some code (listed below this image) of a Notification being broadcast within an application.



Code and process an be found here: http://www.andrewcbancroft.com/2014/10/08/fundamentals-of-nsnotificationcenter-in-swift/

References

[1] UINotificationCenter -

https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter_Class/index.html

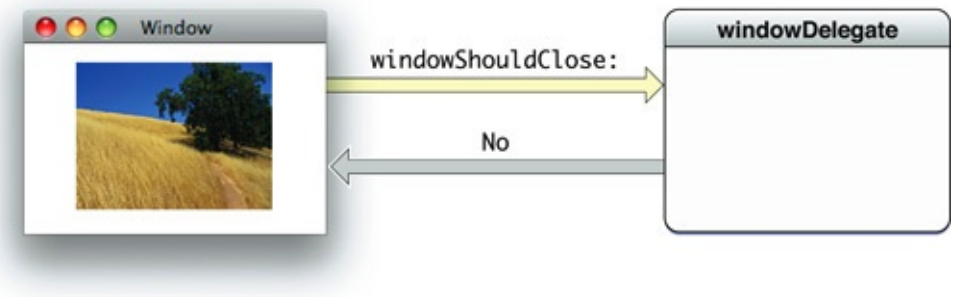
[2] Cocoa Core Competencies - Notification
https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Notification.html

[3] NSNotification - NSHipster - http://nshipster.com/nsnotification-and-nsnotificationcenter/

Delegates (Protocols)

Delegates provide a messaging system between two separate objects. We use delegates to inform controller classes of some updated information or request that an action is to take place.

> Delegation is a design pattern that enables a class or structure to hand off (or delegate) some of its responsibilities to an instance of another type [1]



The above diagram (sourced from [1] below) shows an example of a delegate call between a window and its controller. The window is asking if it's OK to close and the controller refuses. Thus, the window stay active.

Let's check out an example of a practical use case for delegates.

Example

So I have a toddler. And part of that experience is teaching my son how to count. We really want to push the limits of counting and go all the way to twenty. Now how would we represent this within a object oriented program? I want to know when my son has started counting, what number he has counted to and when he is finished. Let's see how we can do it with delegates.

```
import Foundation

protocol CountingDelegate {
    func didBeginCounting()
    func didCount(currentValue:Int)
    func didEndCounting(finalCount: Int)
}

class CountTo {

    var delegate: CountingDelegate?

    func beginCounting(countUntil: Int) {
        var internalCounter = 0

        // OK, we're about to begin counting, let's inform our delegate
        delegate?.didBeginCounting()

        while ++internalCounter < countUntil {
            delegate?.didCount(internalCounter)
        }

        // Look's like we completed our while loop, let's inform our delegate
        delegate?.didEndCounting(internalCounter)
    }
}

class LearningToddler: CountingDelegate {
    var name: String
    var age: Int
    let maximumCountingAbility = 20
    let countingActivity = CountTo()
```

```
    init(toddlerName: String, toddlerAge: Int) {
        name = toddlerName
        age = toddlerAge
    }

    func beginCounting() {
        countingActivity.delegate = self
        countingActivity.beginCounting(maximumCountingAbility)
    }

    // Define our delegate methods which will be invoked
    func didBeginCounting() {
        NSLog("We have begun counting!")
    }

    func didCount(currentValue: Int) {
        NSLog("Our current count is \(currentValue) and we're counting until \(maximumCountingAbility)")
    }

    func didEndCounting(finalCount: Int) {
        NSLog("We successfully counted all the way to \(finalCount)!")
    }
}

// Running the example
let toddler = LearningToddler(toddlerName: "Raleigh", toddlerAge: 2)
toddler.beginCounting()
```

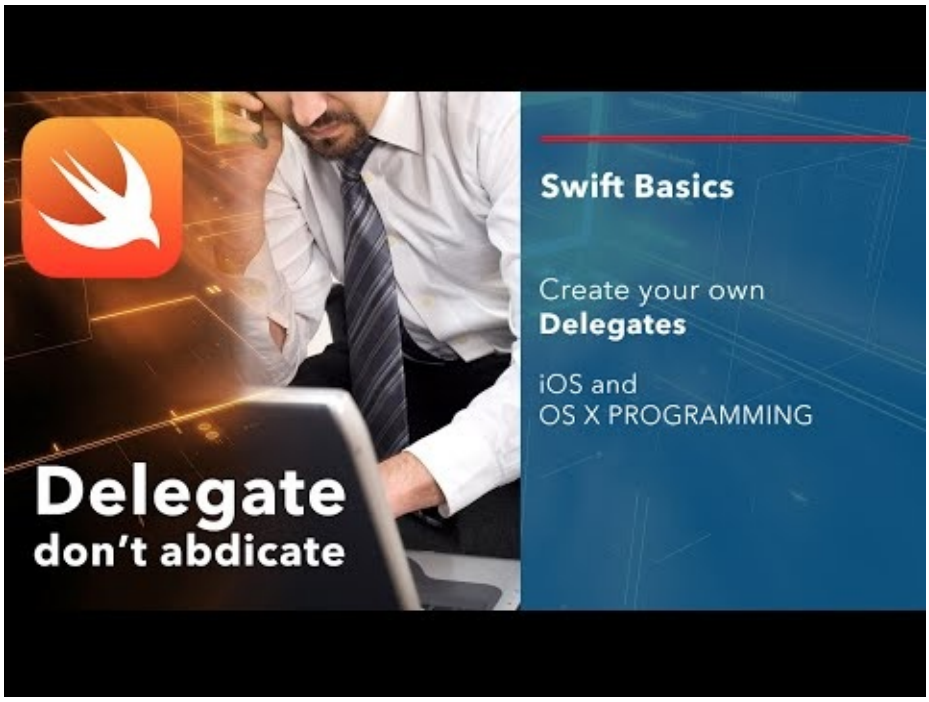
Copying the above code into Playgrounds should yield the following result:

```
2015-01-04 14:47:35.685 MyPlayground[21014:3462547] We have begun counting!
2015-01-04 14:47:35.686 MyPlayground[21014:3462547] Our current count is 1 and we're counting until 20
2015-01-04 14:47:35.687 MyPlayground[21014:3462547] Our current count is 2 and we're counting until 20
2015-01-04 14:47:35.689 MyPlayground[21014:3462547] Our current count is 3 and we're counting until 20
2015-01-04 14:47:35.690 MyPlayground[21014:3462547] Our current count is 4 and we're counting until 20
2015-01-04 14:47:35.692 MyPlayground[21014:3462547] Our current count is 5 and we're counting until 20
2015-01-04 14:47:35.694 MyPlayground[21014:3462547] Our current count is 6 and we're counting until 20
2015-01-04 14:47:35.695 MyPlayground[21014:3462547] Our current count is 7 and we're counting until 20
2015-01-04 14:47:35.697 MyPlayground[21014:3462547] Our current count is 8 and we're counting until 20
2015-01-04 14:47:35.699 MyPlayground[21014:3462547] Our current count is 9 and we're counting until 20
2015-01-04 14:47:35.701 MyPlayground[21014:3462547] Our current count is 10 and we're counting until 20
2015-01-04 14:47:35.703 MyPlayground[21014:3462547] Our current count is 11 and we're counting until 20
2015-01-04 14:47:35.704 MyPlayground[21014:3462547] Our current count is 12 and we're counting until 20
2015-01-04 14:47:35.706 MyPlayground[21014:3462547] Our current count is 13 and we're counting until 20
2015-01-04 14:47:35.708 MyPlayground[21014:3462547] Our current count is 14 and we're counting until 20
2015-01-04 14:47:35.709 MyPlayground[21014:3462547] Our current count is 15 and we're counting until 20
2015-01-04 14:47:35.711 MyPlayground[21014:3462547] Our current count is 16 and we're counting until 20
2015-01-04 14:47:35.712 MyPlayground[21014:3462547] Our current count is 17 and we're counting until 20
2015-01-04 14:47:35.714 MyPlayground[21014:3462547] Our current count is 18 and we're counting until 20
2015-01-04 14:47:35.716 MyPlayground[21014:3462547] Our current count is 19 and we're counting until 20
2015-01-04 14:47:35.717 MyPlayground[21014:3462547] We successfully counted all the way to 20!
```

Let's explore delegates conceptually next.

[Video] Protocols and Delegates

Below is a video which will provide more context on both the concepts and another example use case for delegates:



https://www.youtube.com/watch?v=9LHDsSWc680

References

[1] Swift Programming Guide - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

[2] Cocoa Core - Delegation https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.htm

Views

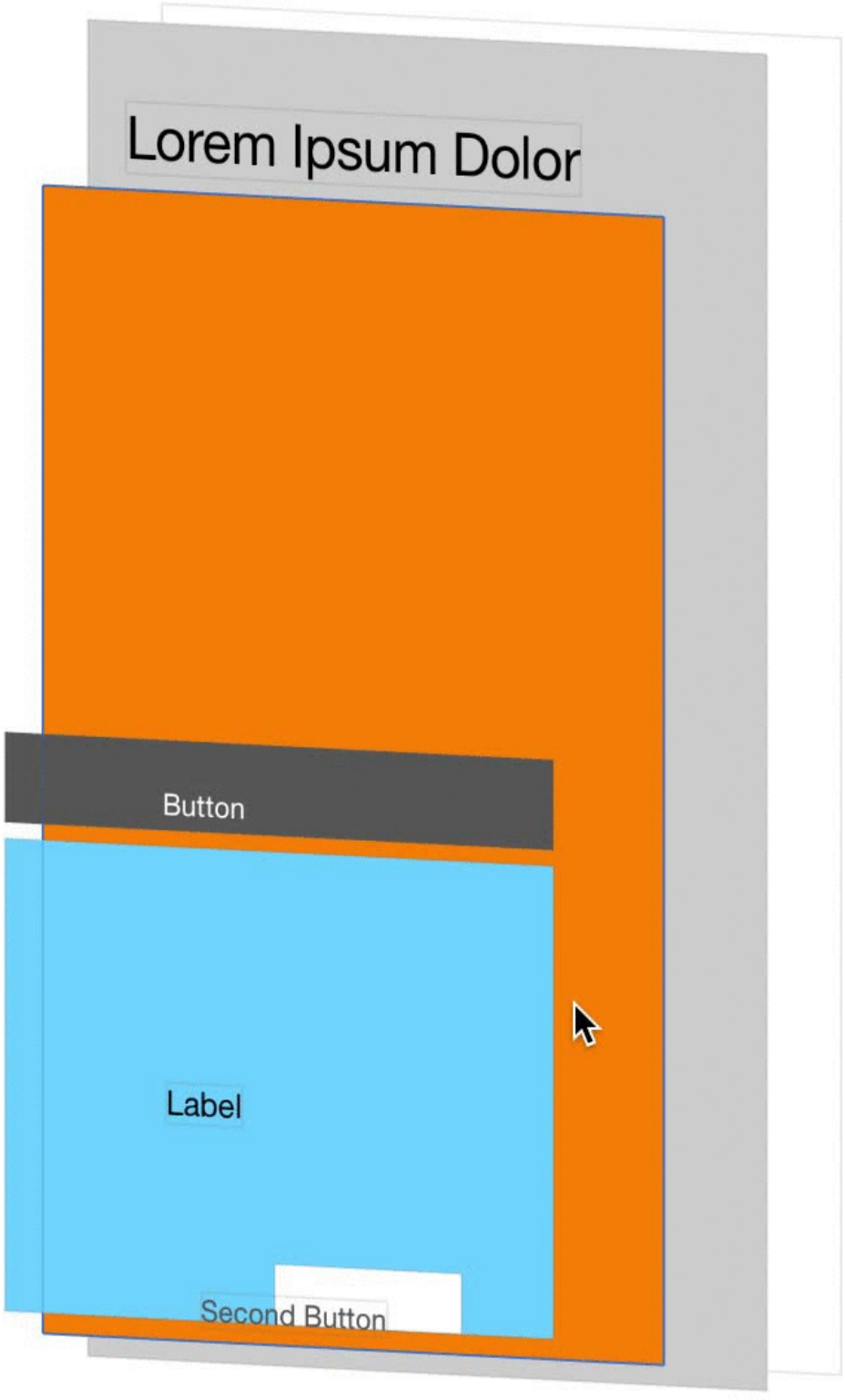
Views are the building block of iOS interfaces. Views display graphics and information to the end user and are their design is managed in a variety of ways.

> UIView provides a structure for drawing and handling events. A UIView object claims a rectangular region of its enclosing superview (its parent in the view hierarchy) and is responsible for all drawing in that region, as well as receiving events that occur in the region. [Xcode Tooltips]

Contents

This section will cover the following chapters

1. Working with Views - Describes view creation and hierarchy.
2. UIScrollView - Describes an example view subclass commonly used in iOS apps.
3. View Layout - Goes into detail on how to build adaptive layouts in IB as well as code.



The above examples shows an exploded representation of a view heirarchy which is discussed here. Working with Views

References

Although each section has its own references, a good starting point would be:

Auto Layout Concepts
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/AutoLayoutConcepts/AutoLayoutConcepts.html

Working with Views

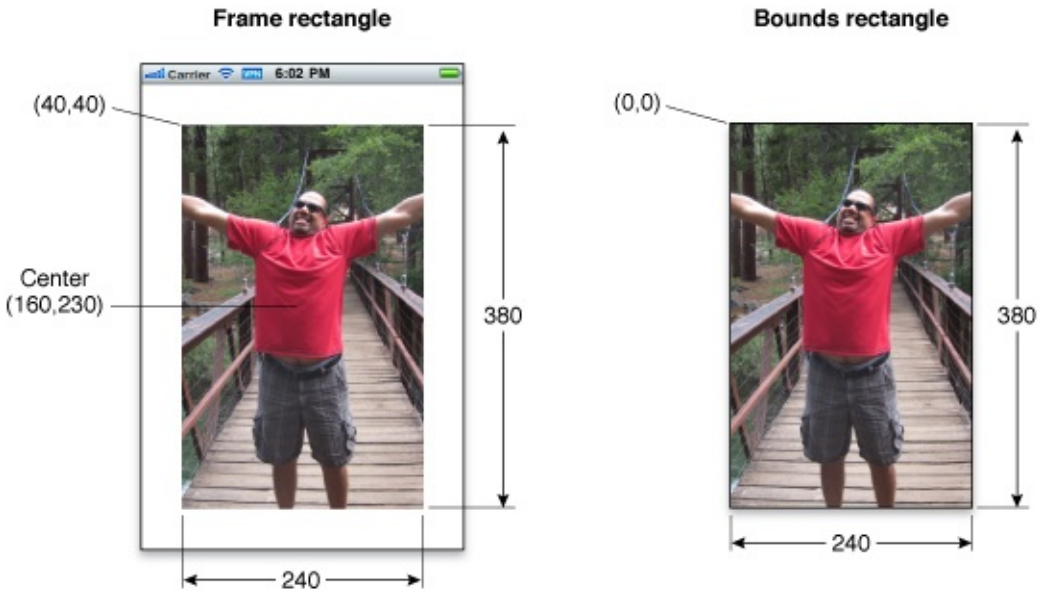
> View objects are the main way your application interacts with the user [1]

Definitions

Before we dive in, let's build our vocabulary associated with UIViews:

- **UIView** - The UIView class defines a rectangular area on the screen and the interfaces for managing the content in that area. At runtime, a view object handles the rendering of any content in its area and also handles any interactions with that content.[2]
- **Coordinate System** - a system of defining the location of objects within a 2-dimensional space.
- **Frame** - The frame rectangle for the view, measured in points. The origin of the frame is relative to the superview in which you plan to add it. [2]
- **Bounds** - The bounds rectangle, which describes the view's location and size in its own coordinate system. [2]
- **CGPoint** - Represents a point in a two-dimensional coordinate system. [3]
  > (x, y)
- **CGSize** - A data structure that represents the dimensions of width and height. [3]
  > (width, height)
- **CGRect** - A data structure that represents the location and dimensions of a rectangle. [3]
  > (x, y, width, height)

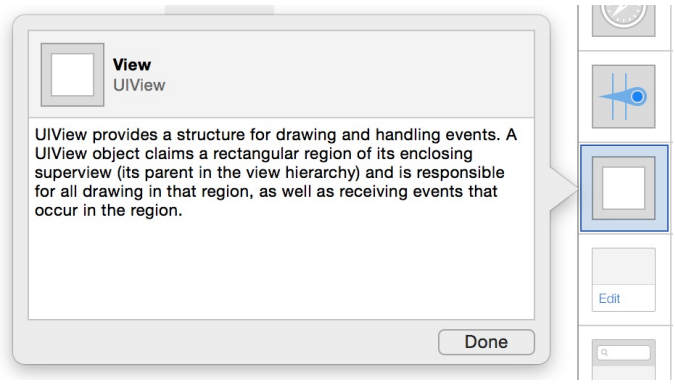
The **difference between frame and bounds** is the coordinate system the frame refers. As visualized below, frames refer to the parent view's coordinate system. Bounds refers to the view's own coordinate system:



Source: https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewPG_iPhoneOS/WindowsandViews/WindowsandViews.html

Creating Views

The majority of time, you will be creating views within Interface Builder. They are automatically added to your view controllers, they are implicitly created with UIButtons and UILabels and of course, you can add them directly through the Object Library.

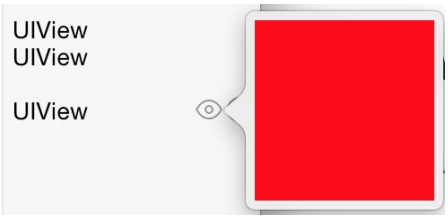


There may be times, however where it is important to create views within your code, outside of IB. The following code creates a simple view with a red background:

```
import UIKit

var view = UIView()
view.frame = CGRect(x: 0.0, y: 0.0, width: 100.0, height: 100.0)
view.backgroundColor = UIColor.redColor()
```

The above codes output, as seen in Playgrounds:

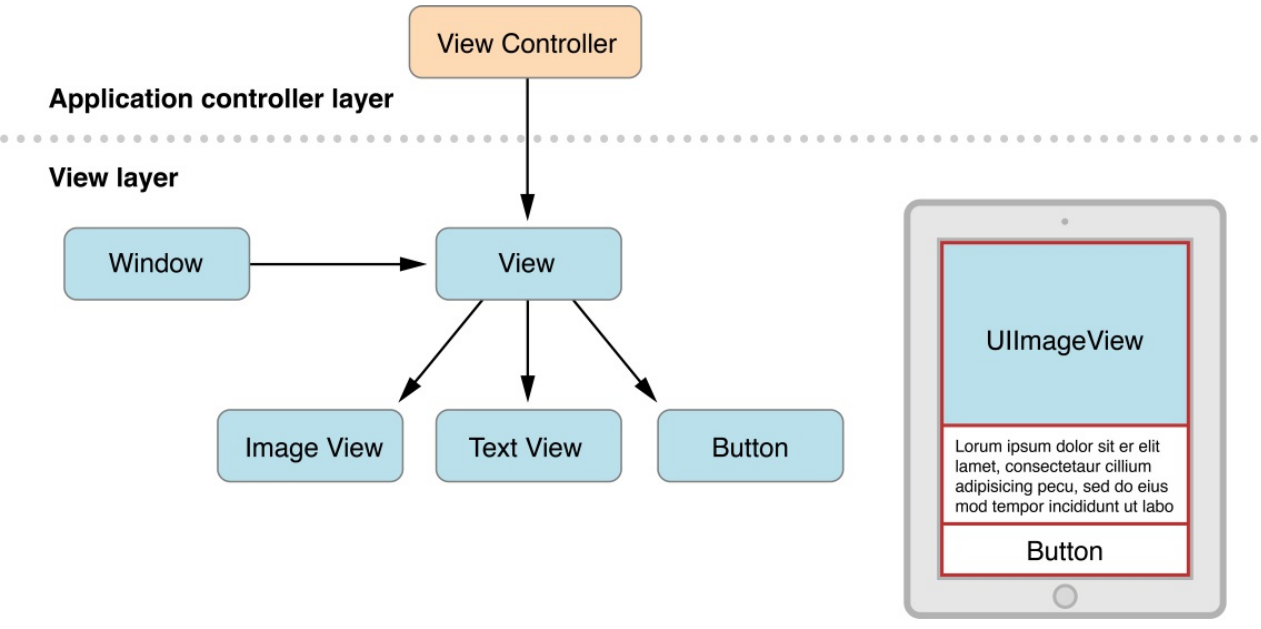
```
var view = UIView()
view.frame = CGRect(x: 0.0, y: 0.0, width: 100.0,
    height: 100.0)
view.backgroundColor = UIColor.redColor()
```



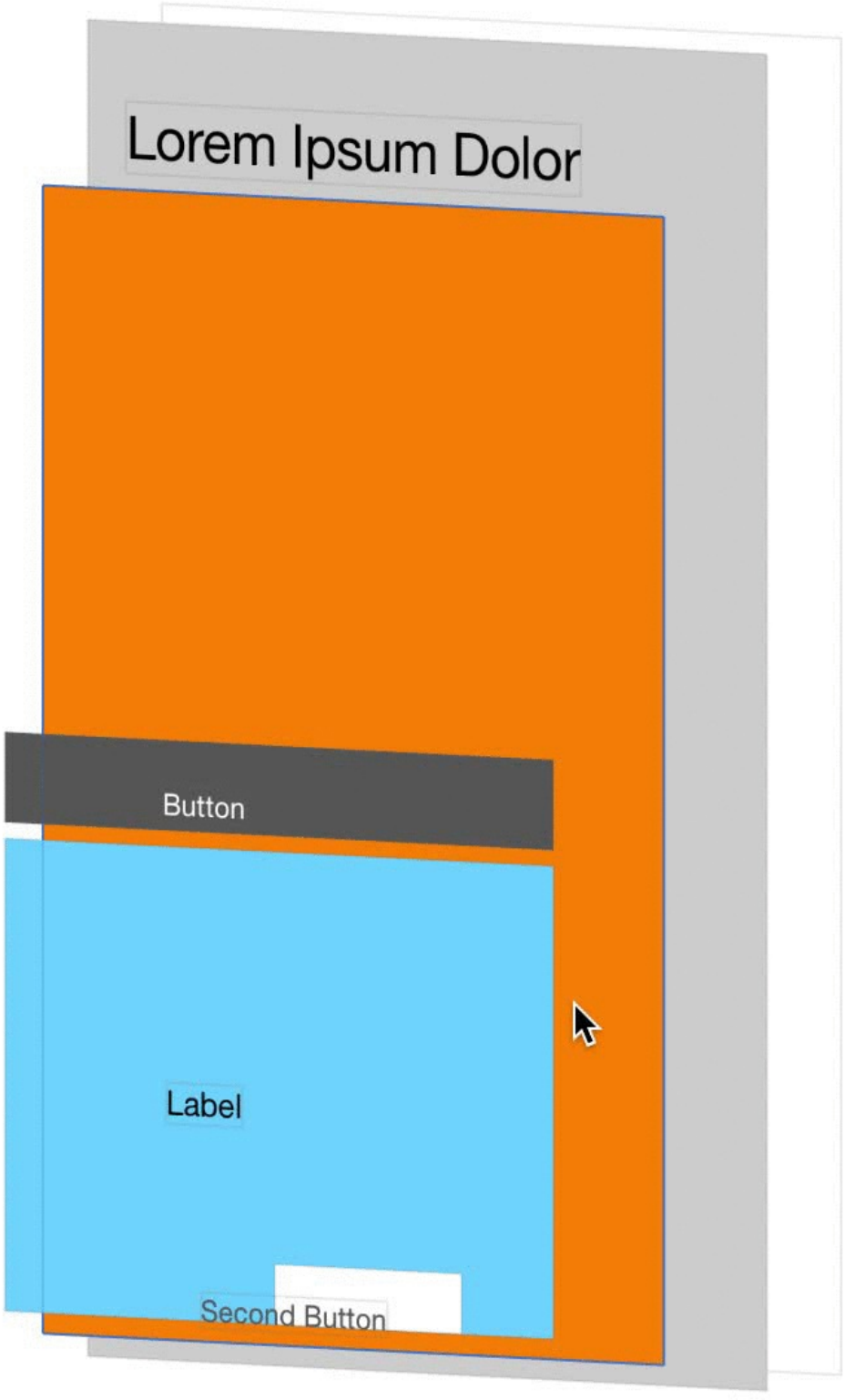
Now that we've seen one view, let's see how views are displayed within a hierarchy.

View Hierarchy

An important concept of views is hierarchy. Views are layered on top of each other. In the below example, we see an UIView that contains an UIImageView, UITextView and a UIButton.



This hierarchy can be visualized using the Xcode Debug View Hierarchy:



Notice how each layer is either below, the same or above adjacent views. Planning your view hierarchy is important is crucial for good interface design.

References

[1] View Programming Guide for iOS - https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewPG_iPhoneOS/CreatingViews/CreatingViews.html

[2] UIView Class Reference - https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html

[3] CGGeometry Reference - https://developer.apple.com/library/mac/documentation/GraphicsImaging/Reference/CGGeometry/index.html

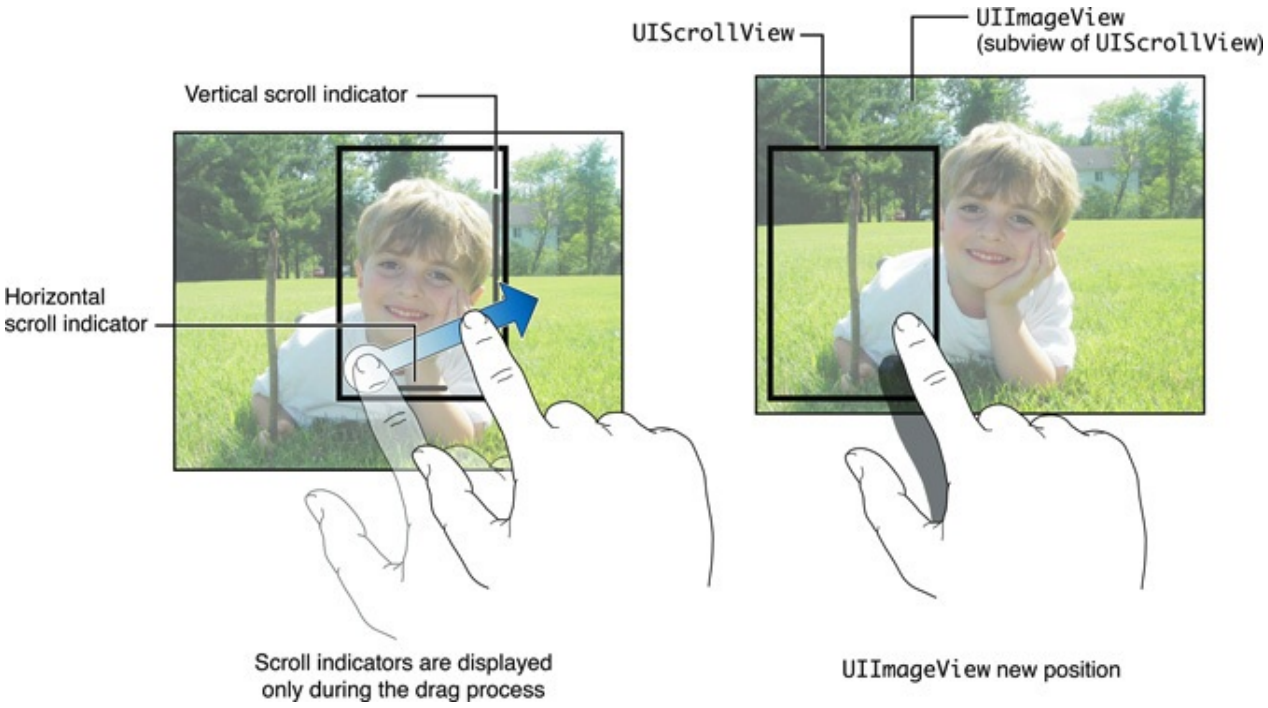
[4] View and Window Architecture - https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewPG_iPhoneOS/WindowsandViews/WindowsandViews.html

[5] Debugging Views - https://developer.apple.com/library/ios/recipes/xcode_help-debugger/using_view_debugger/using_view_debugger.html

[6] Defining the Interaction - https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DefiningtheInteraction.html

UIScrollView

The UIScrollView can be used explicitly with the control from the Object Library and the manual creation through code. UIScrollView is also included within UITableView, UIWebView and UICollectionView.



So why use a scroll view, why are scroll views included in UITableView?

> UIScrollView provides a mechanism to display content that is larger than the size of the application's window and enables users to scroll within that content by making swiping gestures. [Xcode]

UIScrollView has two sizes

1. Frame size which is the size of its presentation
2. Content size which is the size of the content within the scroll view.

As long as the content size is larger than the frame size, UIScrollView allows the user to pan around the images or other content within the control.

[Video] - Zoom and pan large images - UIScrollView

Let's dive in and watch a developer go through the process of creating a UIScrollView and watch the developer add a large

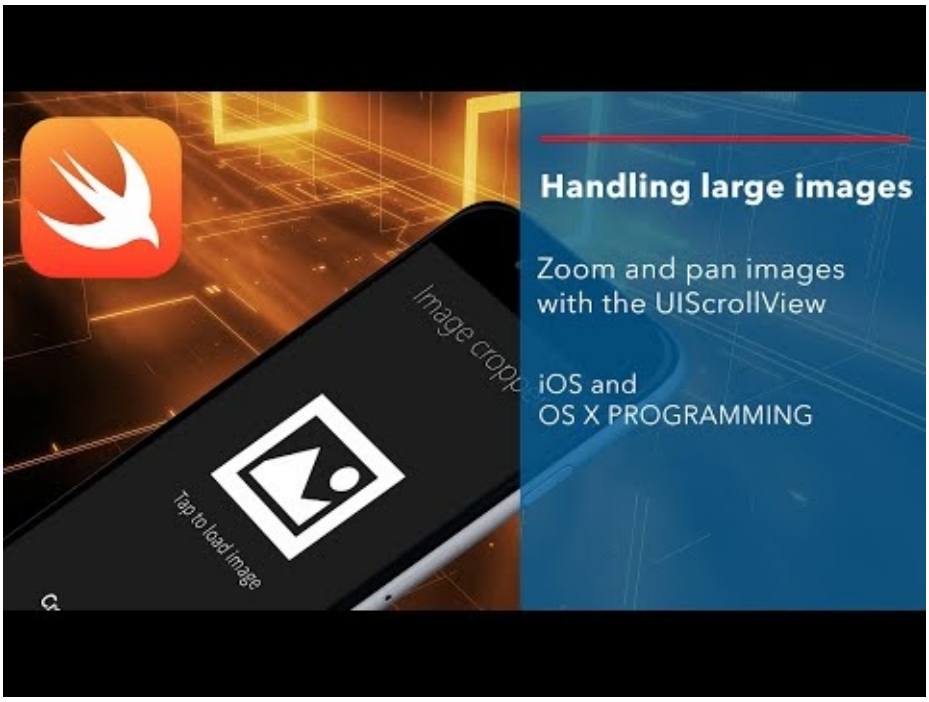


image to pan and zoom.

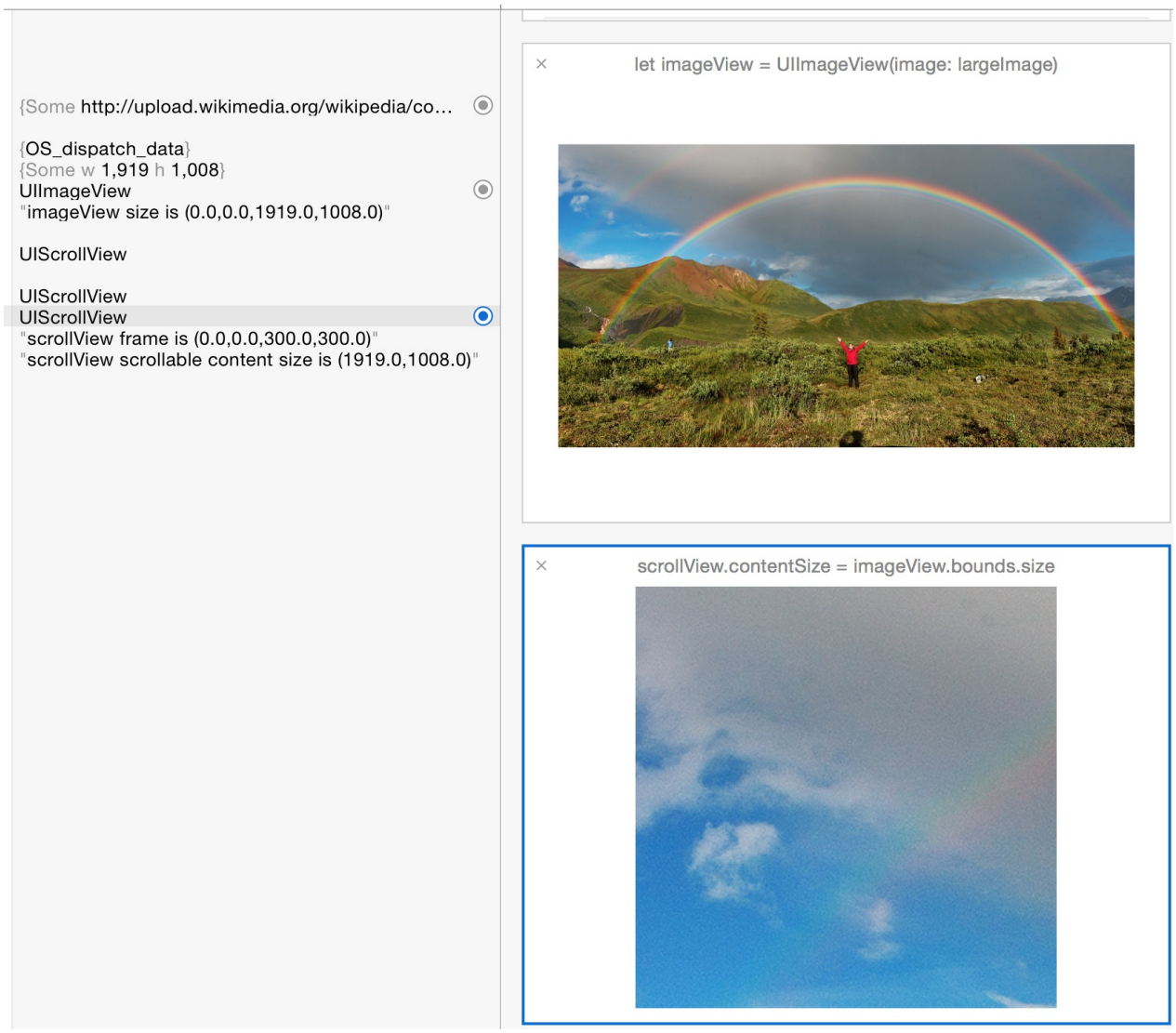
Here is another example of a very similar use case, but with a different implementation style.

Code

Below is an example of configuring a UIScrollView with a large UIImage loaded from Wikicommons.

```
import UIKit

let imageURL = NSURL(string: "http://upload.wikimedia.org/wikipedia/commons/5/5c/Double-alaskan-rainbow.jpg")
let imageData = NSData(contentsOfURL: imageURL!)
let largeImage = UIImage(data: imageData!)
let imageView = UIImageView(image: largeImage)
print("imageView size is \(imageView.bounds)")

let scrollView = UIScrollView(frame: CGRectMake(0, 0, 300.0, 300.0))
scrollView.addSubview(imageView)
scrollView.contentSize = imageView.bounds.size
print("scrollView frame is \(scrollView.frame)")
print("scrollView scrollable content size is \(scrollView.contentSize)")
```

The last two lines demonstrate that even though the view frame of the UIScrollView is 300x300, the content will scroll to the UIImageview size of 1919x1008



The UIScrollView functionality is more than just images, of course. You can create more powerful examples such as paging views and utilize delegate callbacks to observe notifications on scrolling behavior.

References

[1] UIScrollView Class
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScrollView_Class/index.html

[2] Scroll View Programming Guide for iOS
https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/UIScrollView_pg/Introduction/Introduction.html

View Layout

iOS defines where views will be placed on screen with a system called Auto Layout.

> Auto Layout is a system that lets you lay out your app's user interface by creating a mathematical description of the relationships between the elements. You define these relationships in terms of constraints either on individual elements, or between sets of elements. [1]

Auto Layout is a term used to describe how iOS will dynamically adjust the various views of your app. View layout can be configured through a variety different ways, all of which will be described in detail below.

This chapter will be covering:

1. **Definitions**
2. **Springs and Struts**
3. **Interface Builder**
4. **Auto Layout From Code**

Definitions

Let's frame our material with definitions for terms encountered when talking about Auto Layout.

- **Constraints** - A constraint is a mathematical representation of a human-expressable statement. [2]
  - **Constant** - The physical size or offset, in points, of the constraint. [2]
  - **Relation** - You can use relations and inequalities such as greater-than-or-equal to define constraints. [2]
  - **Priority** - Constraints with higher priority levels are satisfied before constraints with lower priority levels. [2]
  - **Multiplier**
- **Content** - Content refers to the data being displayed in your label, button or view.
  - **Hugging** - Defines how a view will hug its contents when resized.
  - **Content Compression** - Defines how a view will compress the content of its view when resized.
  - **Intrinsic Size** - Content knows its size better than its containing view. A UILabel that has dynamic text will know its size better than its containing view.

Each term will be expanded upon in detail below.

Springs and Struts

Before we continue, let's quickly talk about springs and structs. Springs and struts where the common way of laying out views until iOS 7. Springs and struts are also known as **autoresizing masks**.

> A flexible width the view will become proportionally wider if the superview also becomes wider. And with a fixed right margin, the view's right edge will always stick to the superview's right edge.

iOS development has moved beyond Springs and Struts since iOS7.

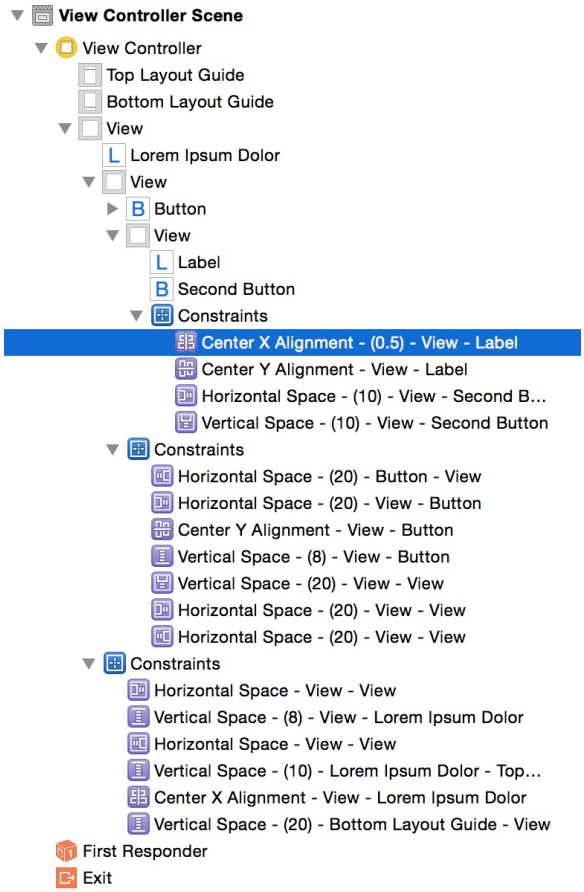
> The [springs and struts] system works well for simple cases, but it quickly breaks down when your layouts become more intricate.

When iPhone moved to multiple sized devices, Auto Layout became indispensable.

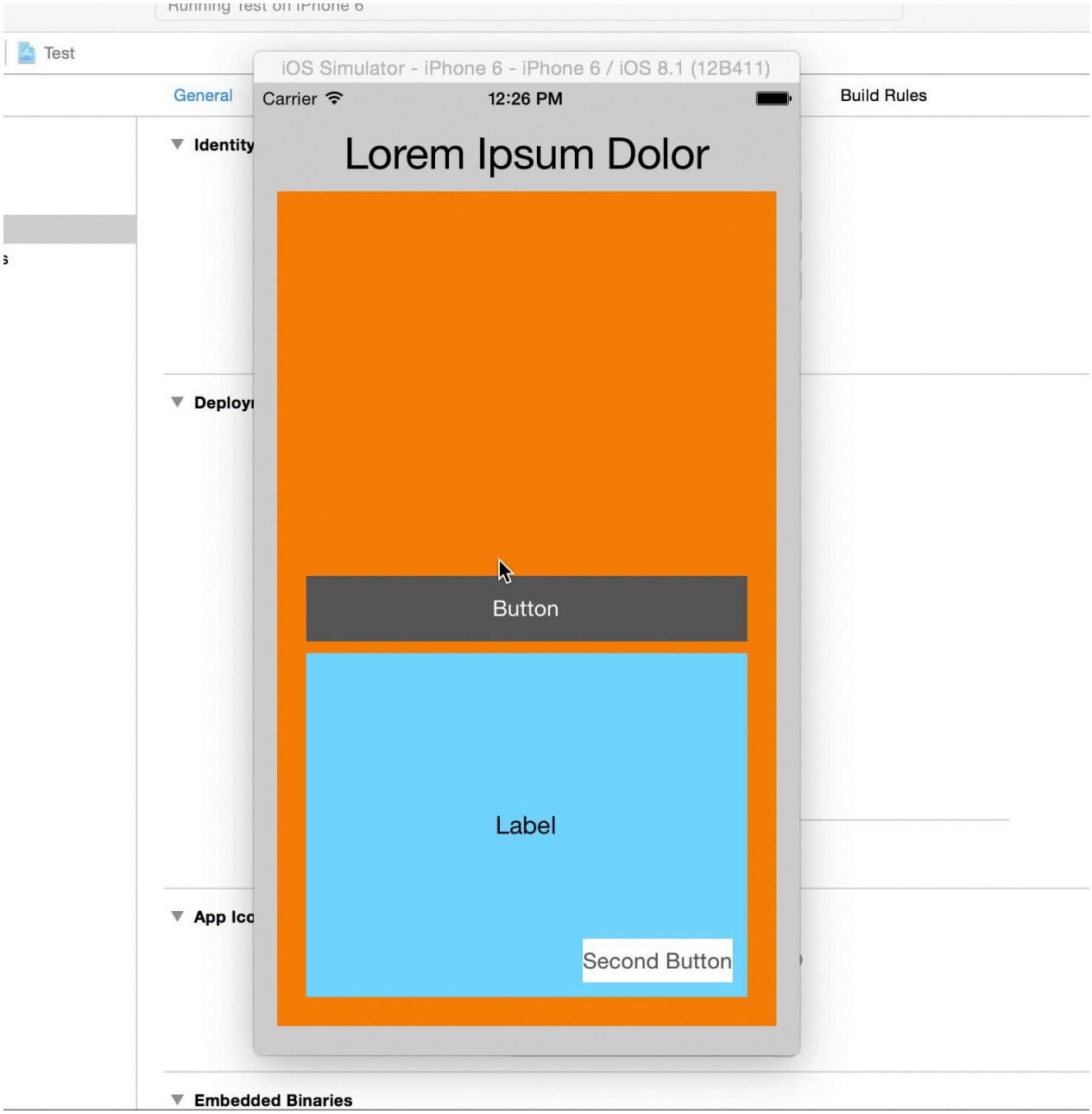
Next step, getting started with Auto Layout.

Overview of Auto Layout in IB

It is easiest to begin in Interface Builder when configuring Auto Layout in your mobile app. Below shows a configured view as seen in Document Outline:



The output of these constraints can be seen in both iPhone with both orientations:



Next, we will watch a video demonstrating the process of setting up constraints in a mobile app.

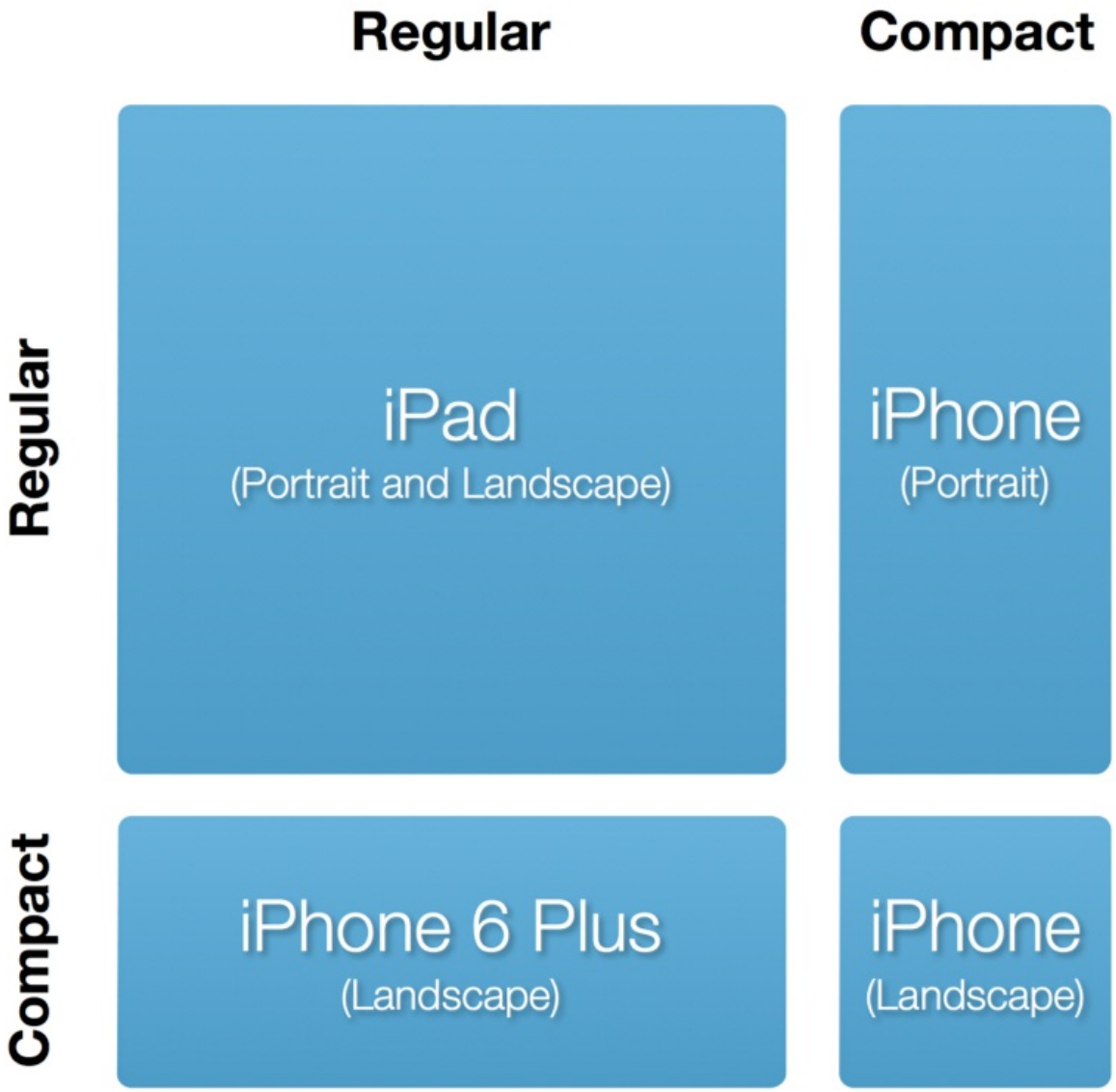
[Video] Building Adaptive Layouts with UIKit

Apple gave an excellent talk at WWDC 2014 on adaptive Layouts. In this video, Apple engineers talk about how and why to make your views adaptive using Auto Layout:

http://devstreaming.apple.com/videos/wwdc/2014/216xxcnxc6wnkf3/216/216_hd_building_adaptive_apps_with_uikit.mov

Advanced: Size Classes

The above video briefly covered size classes. Size classes are used to define layouts for particular devices. Best practices are to keep one layout across all devices and move to more specific size classes only when necessary.



Source [4]

Further information on Size Classes can be found here: https://developer.apple.com/library/ios/recipes/xcode_help-IB_adaptive_sizes/chapters/AboutAdaptiveSizeDesign.html

Overview of Auto Layout in Code

There are two ways to control Auto Layout through code. Those are

1. **NSLayoutConstraints**
2. **Visual Format Language**

We will go into each in detail now.

NSLayoutConstraints

NSLayoutConstraints can be have associated IBOutlets within your implementation file. They can also be generated and attached within code.

First, an example of controlling a constraint by qualifying it with an IBOutlet, connecting it in Xcode and controlling the constant in code:

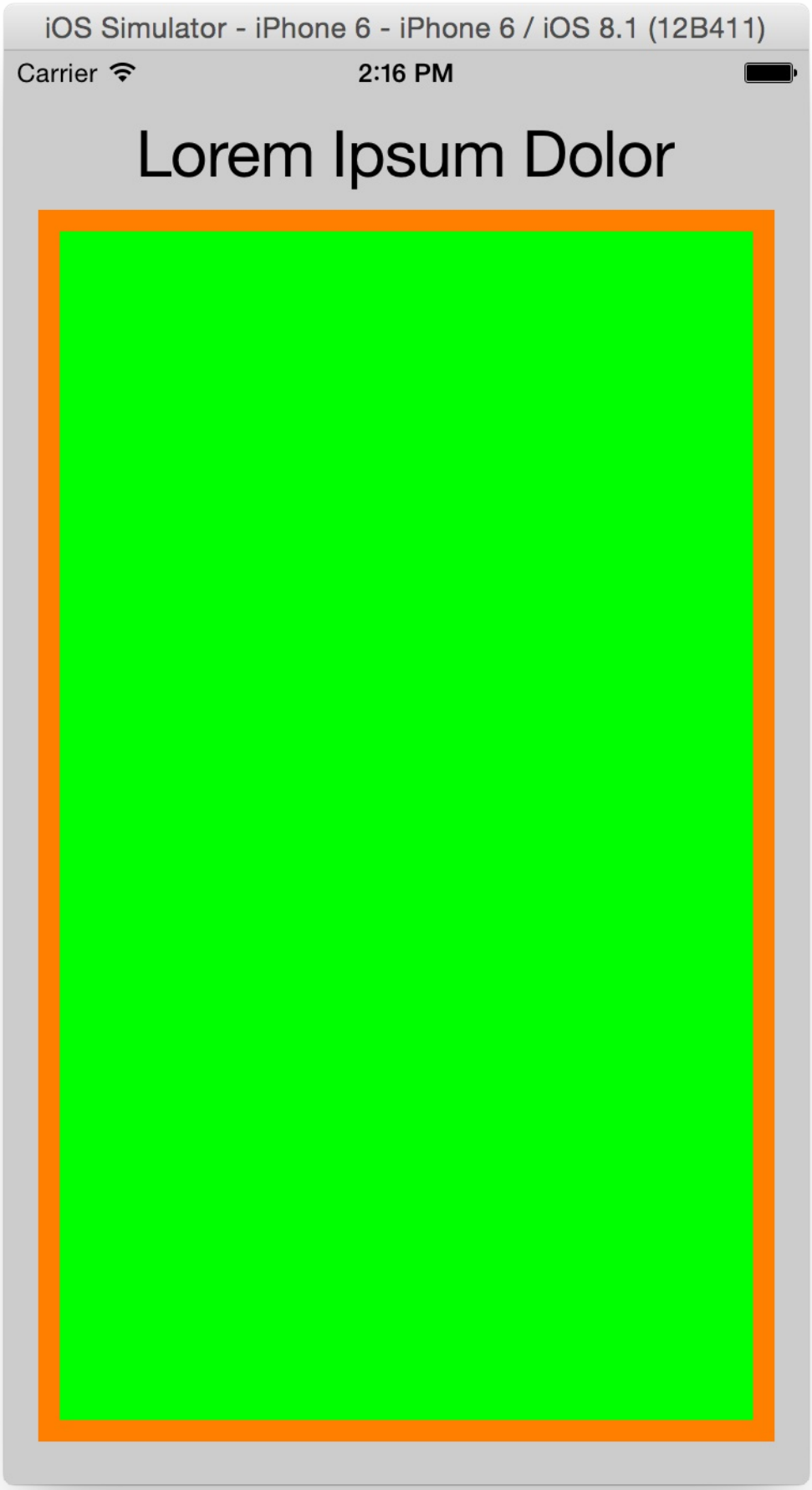
```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak private var bottomConstraint: NSLayoutConstraint!

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)
        bottomConstraint.constant = -8
    }
}
```

This is pretty powerful in that you can animate and dynamically resize views based on constraints instead of geometry.

Next, you can also add constraints directly to views within code, without the need for Interface Builder. The following code creates an UIView, adds it as a subview and attaches constraints in order to create a 10 pixel border.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak private var containerView: UIView!

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)

        var innerView = UIView()
        innerView.backgroundColor = UIColor.greenColor()
        containerView.addSubview(innerView)

        innerView.setTranslatesAutoresizingMaskIntoConstraints(false)
        let constraintTop = NSLayoutConstraint(item: innerView, attribute: .Top, relatedBy: .Equal, toItem: containerVi
        let constraintRight = NSLayoutConstraint(item: innerView, attribute: .Trailing, relatedBy: .Equal, toItem: cont
        let constraintBottom = NSLayoutConstraint(item: innerView, attribute: .Bottom, relatedBy: .Equal, toItem: conta
        let constraintLeft = NSLayoutConstraint(item: innerView, attribute: .Leading, relatedBy: .Equal, toItem: contai
        containerView.addConstraints([constraintTop, constraintRight, constraintBottom, constraintLeft])
        containerView.layoutIfNeeded()
    }
}
```

The output of the above code will then be:


iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.1 (12B411)
Carrier
2:16 PM
Lorem Ipsum Dolor

The following tutorial expands on the concept of adding constriants in Swift with several examples:

http://www.ioscreator.com/tutorials/auto-layout-in-ios-6-adding-constraints-through-code

Visual Format Language

VFL is an advanced topic that we will mention here and encourage you to explore deeper once the other methods of Auto Layout have been tackled.

The next code example translates the previous NSLayoutConstraint code into VFL, which produces the same result as above:

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak private var containerView: UIView!

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)

        var innerView = UIView()
        innerView.backgroundColor = UIColor.greenColor()
        containerView.addSubview(innerView)

        innerView.setTranslatesAutoresizingMaskIntoConstraints(false)
        let views = Dictionary(dictionaryLiteral: ("inner", innerView))
        let horiz = NSLayoutConstraint.constraintsWithVisualFormat("H:|-10.0-[inner]-10.0-|", options: nil, metrics: ni
        let vert = NSLayoutConstraint.constraintsWithVisualFormat("V:|-10.0-[inner]-10.0-|", options: nil, metrics: nil
        containerView.addConstraints(horiz)
        containerView.addConstraints(vert)
    }
}
```

For further step-by-step tutorials and instruction in VFL, please check out the following two resouces:

1. http://nsscreencast.com/episodes/134-visual-format-language
2. http://commandshift.co.uk/blog/2013/01/31/visual-format-language-for-autolayout/

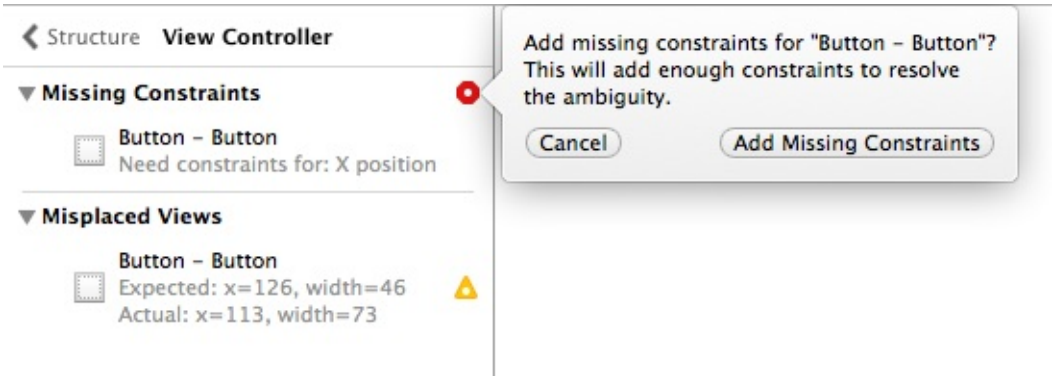
Handling Warnings & Errors

Auto layout is very specific. Constraints must be understood by iOS in order to function.

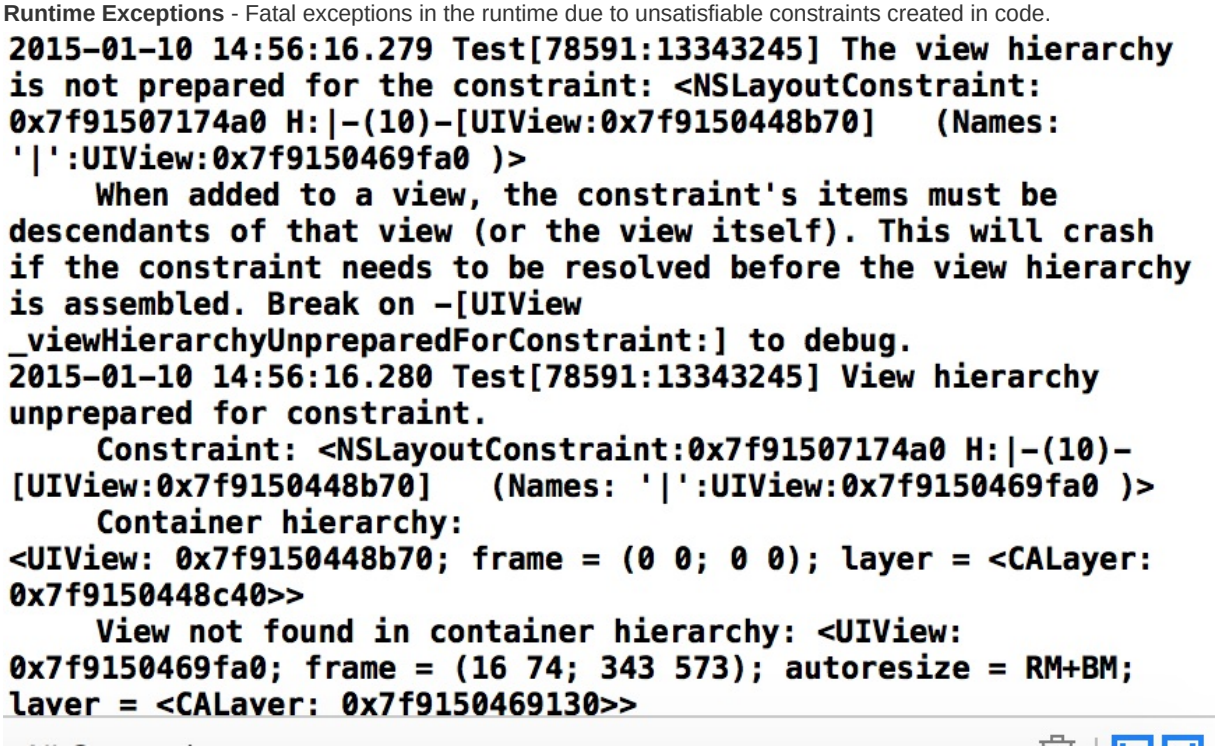
> Auto Layout issues occur when you create conflicting constraints, when you don't provide enough constraints, or when the final layout contains a set of constraints that are ambiguous.

The two most common AL problems are

1. **Build Errors** - Conflicting constraints in IB, which will show a red warning indicator and



2. **Runtime Exceptions** - Fatal exceptions in the runtime due to unsatisfiable constraints created in code.



To resolve Auto Layout issues, walk through the code and constraints individually. You can always clear all existing constraints and rebuild them. Each added constraint should be purposed, if you can leave out a constraint, do so.

Apple provides two guides for resolving Auto Layout issues:

1. https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/ResolvingIssues/ResolvingIssues.html
2. https://developer.apple.com/library/ios/recipes/xcode_help-IB_auto_layout/chapters/resolve_view_layout_issues_quickly.html

Xcode provides the ability to attempt and automatically resolve Auto Layout issues for you, when working with Interface Builder. Generally speaking, spend the extra time and resolve the issues yourself. The more experience you have with Auto Layout, the easier it will be to create accurate constraints in the future.

Reference

[1] Auto Layout Guide
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/Introduction/Introduction.html

[2] Auto Layout Concepts
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/AutoLayoutConcepts/AutoLayoutConcepts.html

[3] Beginning Auto Layout http://www.raywenderlich.com/20881/beginning-auto-layout-part-1-of-2

[4] iOS 8 Development Tips https://medium.com/@getaaron/ios-8-development-tips-for-iphone-6-and-iwatch-1c772554ffe0

NSLayoutConstraints
https://developer.apple.com/library/prerelease/ios/documentation/AppKit/Reference/NSLayoutConstraint_Class/index.html

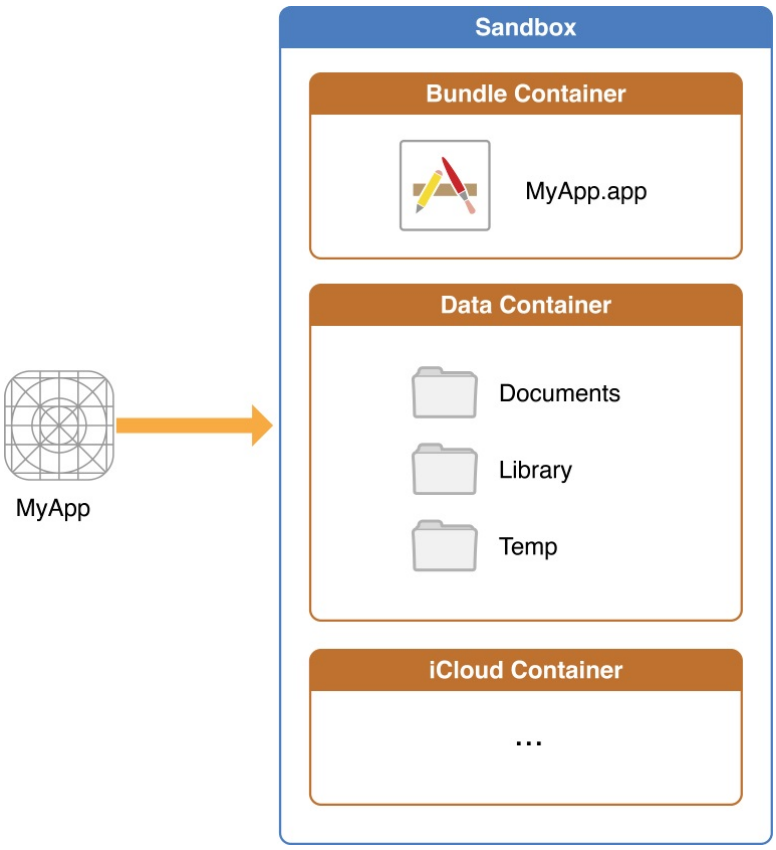
Visual Format Language
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/VisualFormatLanguage/VisualFormatLanguage.html

Persistence

In most applications, storing and retrieving data is crucial. Since an iOS application can be terminated at anytime, it is important to consider how your app will perform data storage and retrieval.

This section will provide detail on how to:

1. Working with local storage
2. Working with disk storage
3. Working with Core Data



Source [1]

References

[1] File System Programming Guide
https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html

Local Storage

iOS provides two convenient and easy ways to store data that will persist across app launches. Those two ways are:

1. NSUserDefaults
2. plist files

These are paradigms typically used for configuration-level settings.

NSUserDefaults

NSUserDefaults persist between app loads and are useful for non-secure configuration settings.

> The NSUserDefaults class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences

Reading from NSUserDefaults

```
import Foundation

let name = "Jeff"
let key = "Name"

var defaults = NSUserDefaults.standardUserDefaults()
defaults.setObject(name, forKey: key)
defaults.synchronize() // save our defaults
```

Note, to save NSUserDefaults to disk, the function **synchronize()** must be called.

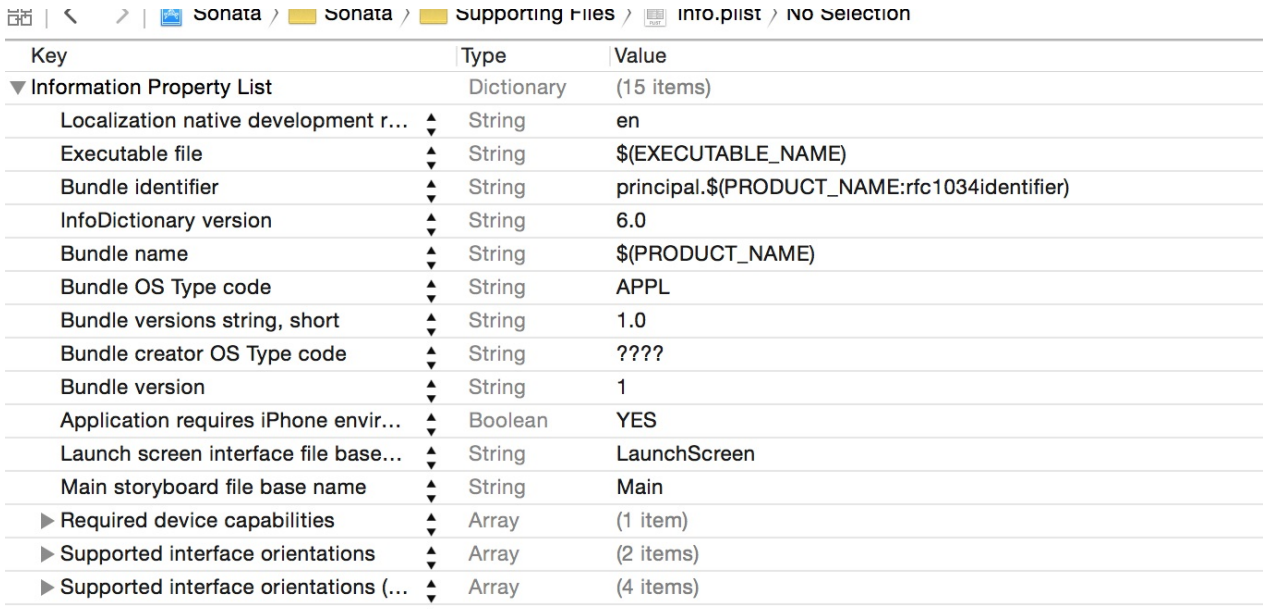
Writing to NSUserDefaults

```
import Foundation

let name = "Jeff"

if let usersName : AnyObject? = defaults.objectForKey(key) {
    NSLog("Hey \(usersName)!")
}
```

You can store a variety of information to NSUserDefaults including arrays, dictionaries, custom objects and scalar types such as integers and floats.

plists

Property Lists (or plists) are XML-based files that are usually contained within your application bundle. They are called plists because of their file extension, which is *.plist. The application bundle are a set of files which comprise imagery, other plists and your application executable.



Reading from a plist file

To read from a plist file, we first get a reference into our main bundle object. Then we copy the contents of that file into a dictionary object. Once we have the dictionary object, we can retrieve a particular value:

```
import Foundation

var textSize = 12.0

let path = NSBundle.mainBundle().pathForResource("AppSettings", ofType: "plist")

if let dict = NSDictionary(contentsOfFile: path!) {
    textSize = dict.objectForKey("Preferred Text Size") as Double
    NSLog("Preferred text size is \(textSize) points")
}
else {
    NSLog("Preferred text is not available, defaulting to \(textSize) points")
}
```

Writing to a plist file

Writing to a plist follows the same conventions as above. The only difference is once we have access to the dictionary, we simply need to write to the dictionary as opposed to read from it. Additionally, we also need to save that file back to disk.

```
import Foundation

var newTextSize = 14.0

let path = NSBundle.mainBundle().pathForResource("AppSettings", ofType: "plist")

if let dict = NSDictionary(contentsOfFile: path!) {
    dict.setValue(newTextSize, forKey: "Preferred Text Size")
    NSLog("Preferred text size is now \(textSize) points")
    dict.writeToFile(path!, atomically: false)
}
else {
    NSLog("Preferred text is not writable")
}
```

For more robust data storage options, read on as we walk through storing data in files and managing a Core Data database.

References

[1] https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSUserDefaults_Class/index.html

[2] https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/AboutInformationPropertyListFiles.html

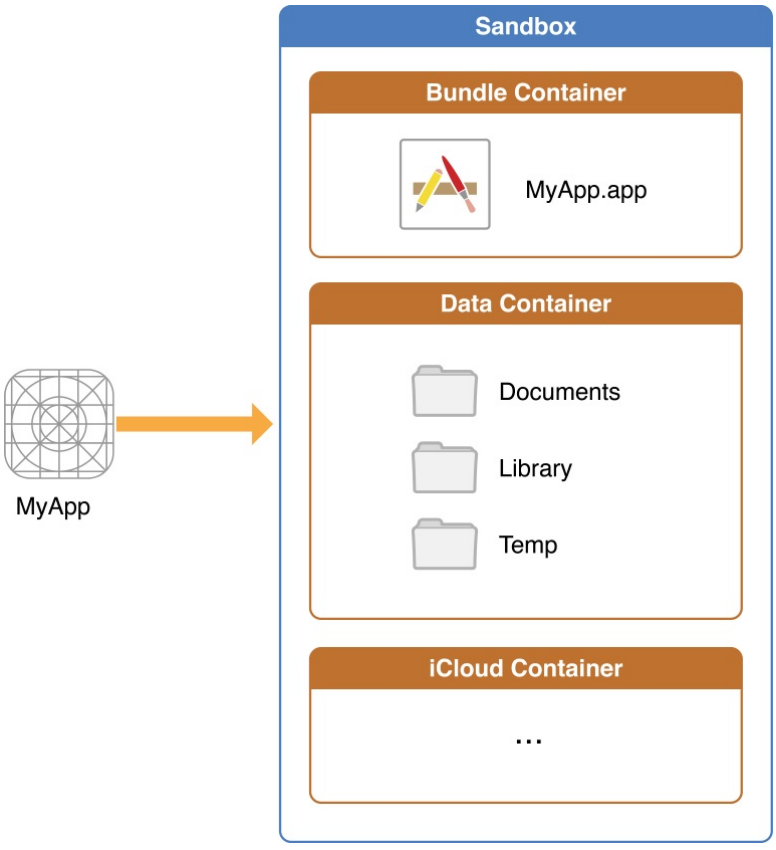
Working with Files

There are instances where you need a more robust file store capability that NSUrlDefaults and plist files cannot help you with. This includes binary data like images and serialized custom class objects. For those instances, writing your own files to disk makes the most sense.

Once we begin working with our own files, more responsibility needs to be assumed. For instance, ensuring that a directory is created, deleting unused files and selecting an appropriate directory.

The Sandbox

iOS **sandboxes** each application, segregating application and user-generated data into its own directory. This is an important security feature of the iOS operating system. Each application is assigned a directory and within that directory a common structure applies:



Where to put files

iOS has a fixed amount of hard drive space. Your phone cannot be upgraded. People's phones accumulate a lot of data including imagery and videos, games and apps, documents and OS-specific files. Because of these constraints, Apple limits the amount of data your app can take. They do this in two ways.

1. Any files placed within the tmp directory will be cleaned out by the operating system at an unknown time. Files placed within this directory should be able to be recreated easily.
2. Any files that are not automatically cleaned up by iOS, i.e. files within the Documents directory, need to have as much of a small footprint as possible. Apple has rejected apps from the iTunes App Store that take up too much space on the user's drive.

Reading Files

Reading a file is straight forward. Check that the file exists, then read it into a variable. Make sure you use the correct

encoding. In our example, we are reading a JSON file which is stored in plain text.

```
import Foundation

let path = NSTemporaryDirectory() + "advanced.json"
var jsonOutput:String?

if NSFileManager().fileExistsAtPath(path) {
    var err:NSError?
    jsonOutput = String(contentsOfFile: path, encoding: NSUTF8StringEncoding, error: &err)
    if err != nil {
        NSLog("We read the following JSON from disk \(jsonOutput)")
    }
}
```

Saving Files and Creating Directories

Saving (or writing) files is also straight forward. Here we want to create the directory if it does not yet exist. Once we have completed that process, we can write to that directory. Here we define the encoding that we used in reading (above). Again, since our example is using a JSON file, we are wanting to write s string.

```
import Foundation

let path = NSTemporaryDirectory() + "advanced/advanced.json"

if !NSFileManager().fileExistsAtPath(path) {
    var err:NSError?
    NSFileManager().createDirectoryAtPath(path, withIntermediateDirectories: true, attributes: nil, error: &err)
}

let jsonOutput = "{'states': {'california': {'population_millions' ': 38.3,'square_miles' : 163696,'percent_water' : 4.

var err:NSError?
let result = jsonOutput.writeToFile(path, atomically: true, encoding: NSUTF8StringEncoding, error: &err)

NSLog("We attempted to remove a file and our result was \(result)")
```

Deleting Files

To delete a file, simply ensure that it exists in a similar fashion as before and call the removeItemAtPath method. There is also a URL equivalent should you not have a string directory reference.

```
import Foundation

let path = NSTemporaryDirectory() + "advanced.json"

if NSFileManager().fileExistsAtPath(path) {
    var err:NSError?
    let success = NSFileManager().removeItemAtPath(path, error: &err)
    NSLog("We attempted to remove a file and our result was \(success)")
}
```

Core data is next. Instead of managing files directly, we can use a central database stored on disk and replicated through iCloud.

References

NSCoding Tutorial for iOS: How To Save Your App Data http://www.raywenderlich.com/1914/nscoding-tutorial-for-ios-how-to-save-your-app-data

File System Programming Guide
https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/Introduction/Introduction.html

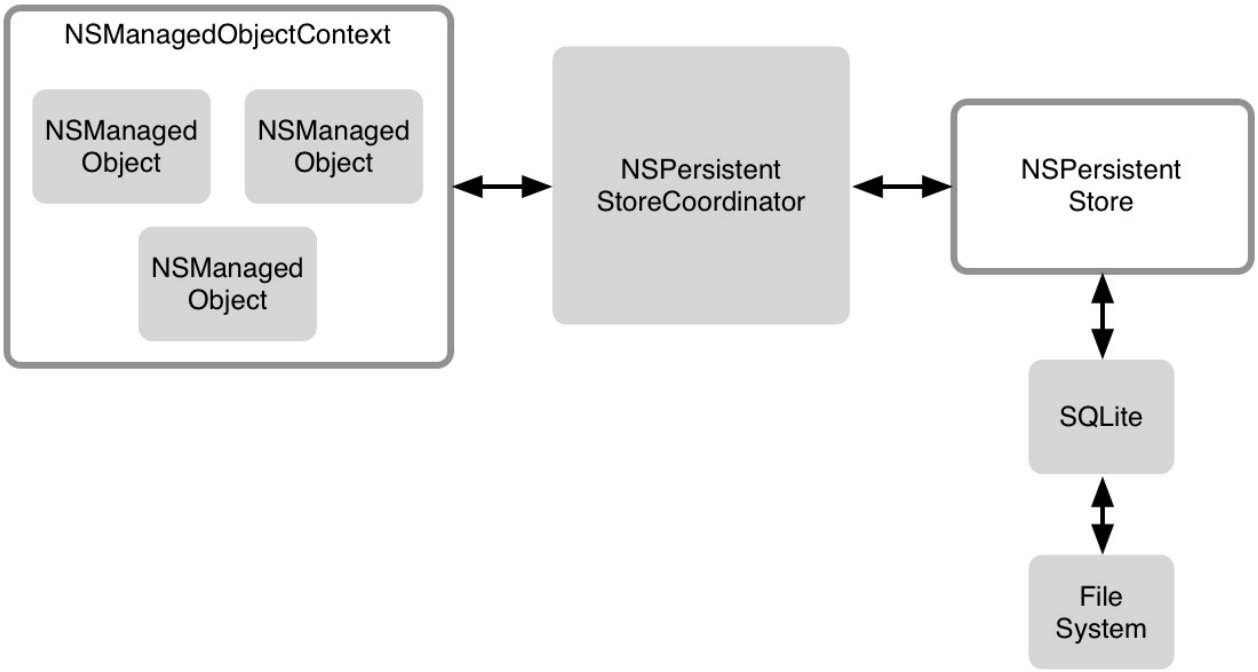
Core Data

In addition to configuration settings found in Local Storage and dynamic file saving, iOS provides a powerful database engine to use within your mobile apps. Welcome to Core Data.

> [Core Data] allows data organised by the relational entity–attribute model to be serialised into XML, binary, or SQLite stores. ... Core Data interfaces directly with SQLite, insulating the developer from the underlying SQL [1]

In a basic form, iOS will be creating a file on disk, called the database. iOS will manage creating, opening, writing to and reading to that file. In order to do so efficiently, quickly and without risking loss of data, we have an abstracted layer on top of this file management layer. Core Data is that abstract layer.

The ObjC blog [12] created graph of a simple Core Data setup:



Definitions

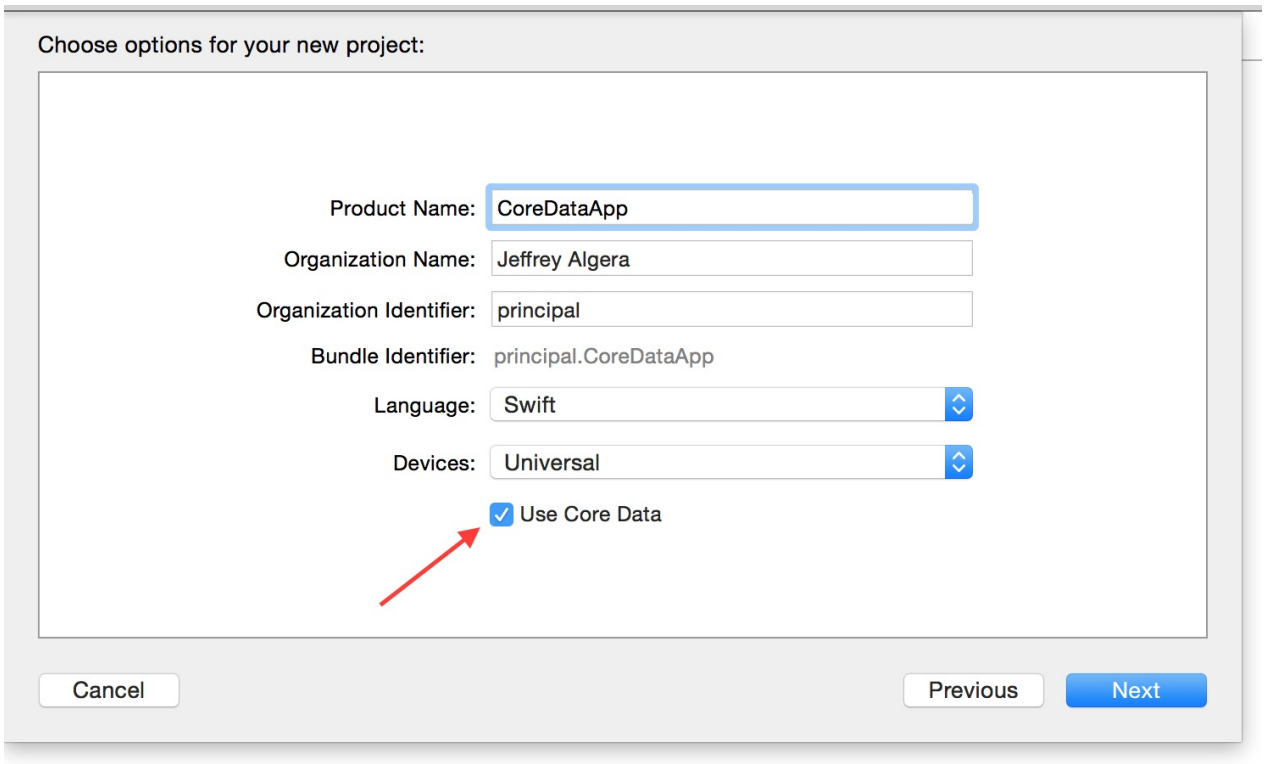
To frame our discussion, let's discuss terms used when implementing Core Data:

- **NSEntityDescription** - Describes an entity (also konwn as table) in Core Data. Main container for a set of information.
- **NSFetchedResultsController** - Manages the results returned from a Core Data fetch request to provide data for a UITableView object. [5]
- **NSFetchRequest** - Describes search criteria used to retrieve data from a persistent store. [6]
- **NSManagedObject** - A class that implements all the basic behavior required of a Core Data model object [7]
- **NSManagedObjectContext** - Represents a single “object space” or scratch pad in an application. Its primary responsibility is to manage a collection of managed objects. [8]
- **NSPersistentStore** - An abstract base class for all Core Data persistent stores, will typically represent a SQLite database for our discussions. [9]
- **NSPersistentStoreCoordinator** - Associate persistent stores with a model and serve to mediate between the persistent store and the managed object context. [10]
- **NSSortDescriptor** - Defines the sorting mechanism for a set of objects.
  > Describes a basis for ordering objects by specifying the property to use to compare the objects, the method to use to compare the properties, and whether the comparison should be ascending or descending. [11]

Let's move on by working with Xcode.

Project templates

Grasping Core Data takes time. Luckily, Apple makes it easy to begin by providing project templates that include all necessary pieces you need to begin working with Core Data. Just make sure to have the "Use Core Data" checkbox selected. Xcode will provide boiler plate to get you setup.



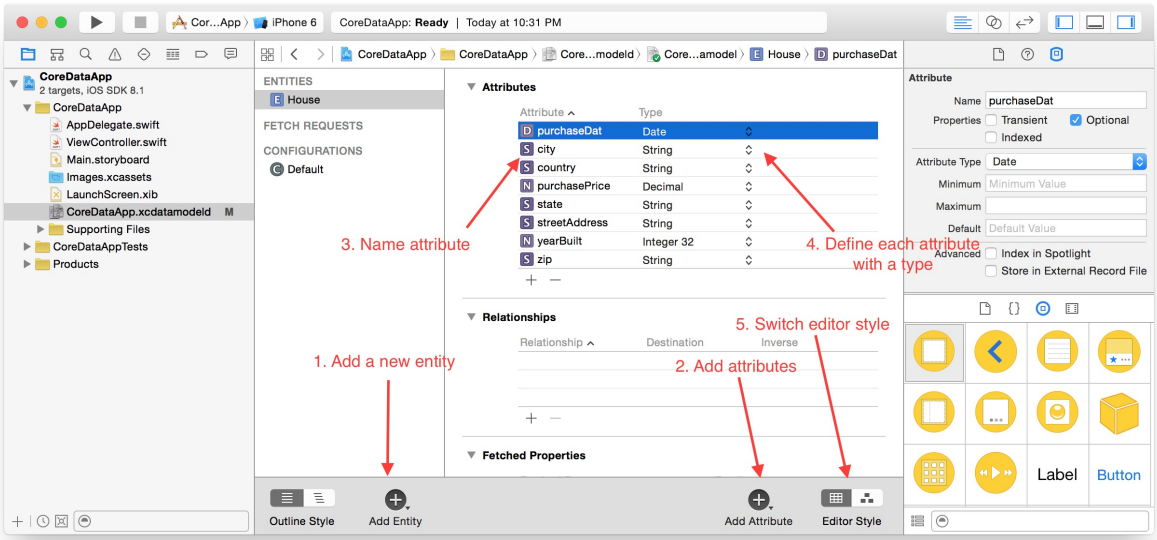
Before we dive to deep, let's reference a video introducing us to Core Data

[Video] Swift Tutorial - Core Data

http://www.youtube.com/watch?v=3IDfgATVqHw

Tools

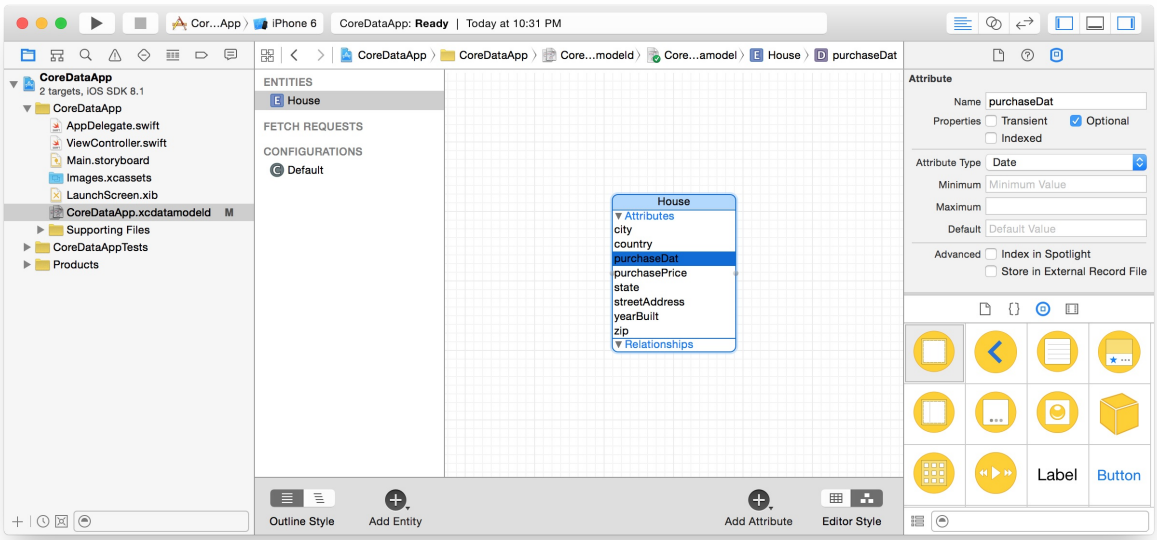
Xcode provides a Data Model view to manage core data objects. Selecting on the provided *.xcdatamodeld file opens the Data Model view in Xcode.



The steps listed in the above graphic are generally the steps approached on a new Core Data app. Note, we are using the Table View (as pictured).

1. Define a model
2. Add a set of attributes to that model
3. Name each attribute accordingly
4. Define each attribute with an appropriate type
5. Switch the editor style over to Graph to view the result

The result of the above steps is then:

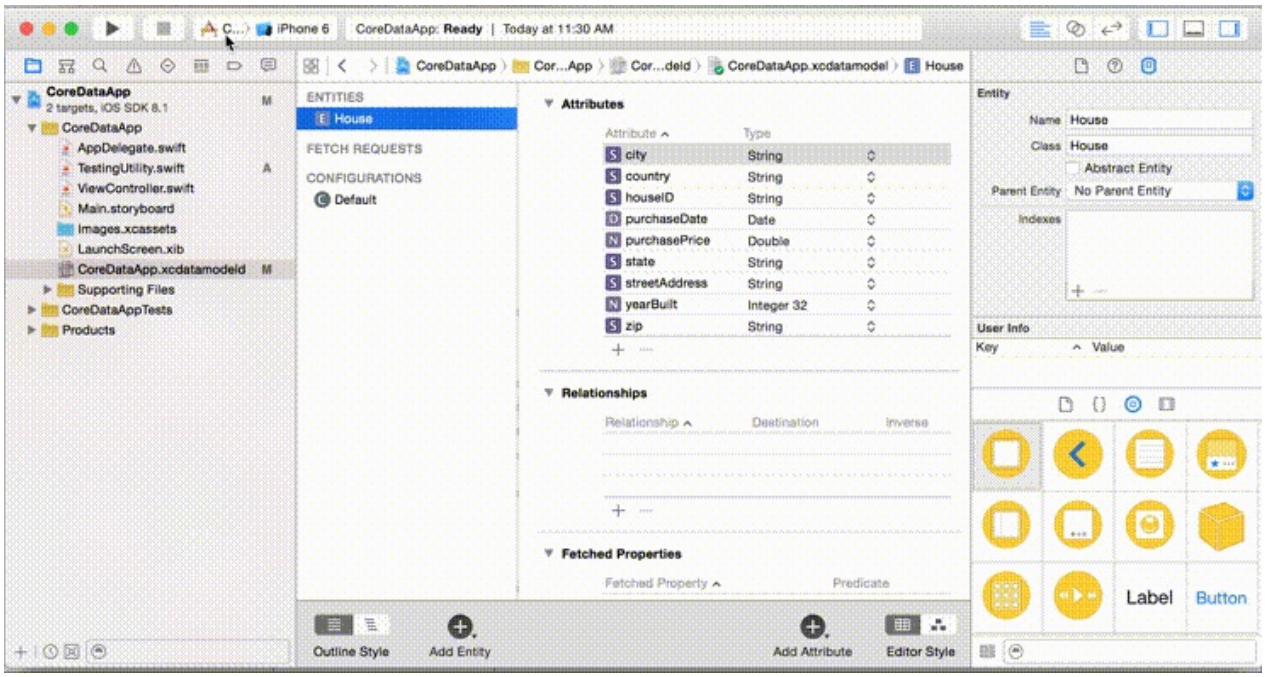


We use this Graph view as a visual representation of the Table View, pictured on the previous step.

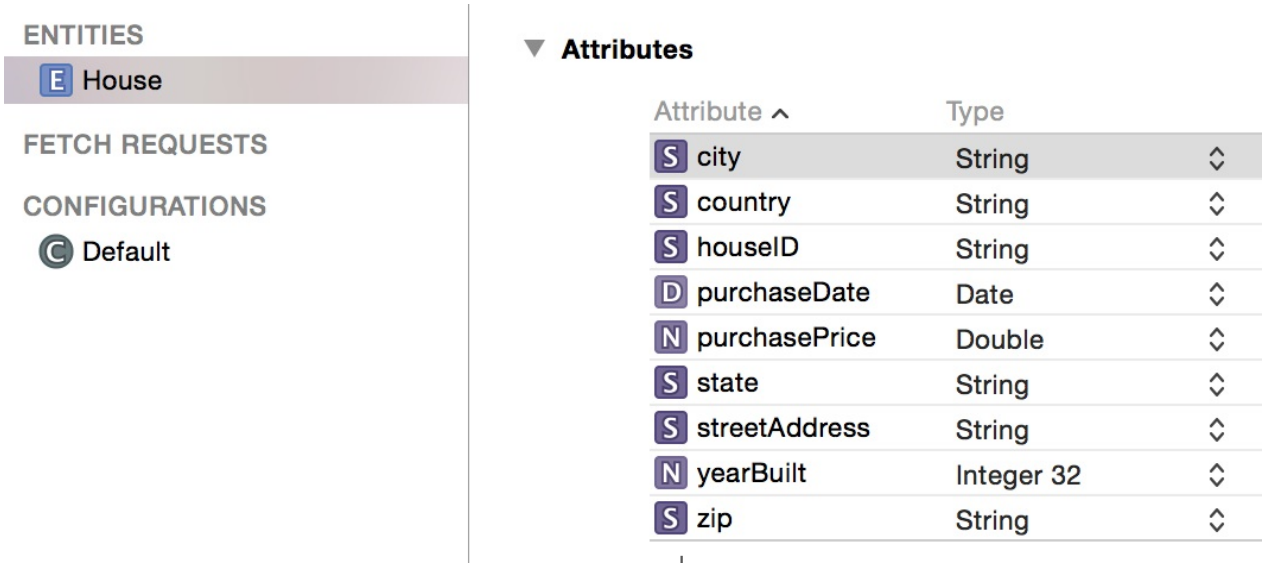
Code

iOS provides quite a bit of boiler plate code for you to use in your application. We will go over some additional code snippets to perform specific operations on the model object we created above.

The entity we will be using is the following:



Net we will generate a model object file from the Data Model view. We did this by selecting on the model, then clicking New File -> NSManagedObject subclass.



This is the House model object we've generated:

```
import Foundation
import CoreData

class House: NSManagedObject {

    @NSManaged var yearBuilt: NSNumber
    @NSManaged var streetAddress: String
    @NSManaged var city: String
    @NSManaged var state: String
    @NSManaged var zip: String
    @NSManaged var country: String
    @NSManaged var purchasePrice: NSNumber
    @NSManaged var purchaseDate: NSDate
    @NSManaged var houseID: String
}
```

We will show our methods to manage the data in the next few code snippets.

Utility Class

```
import UIKit
import CoreData

class TestingUtility: NSObject {

    private var houseModels = Array<House>()
    private var appDelegate = UIApplication.sharedApplication().delegate as AppDelegate

    // We will be adding methods starting here
}
```

Reading rows from the entity

The first function we will add is to retrieve all model objects into an array. We wil use this array as a property on the class.

```
    func retreveAllHouseObjects() {
        let fetchRequest = NSFetchRequest(entityName: "House")
        let sortDescriptor = NSSortDescriptor(key: "PurchasePrice", ascending: true)
        fetchRequest.sortDescriptors = [sortDescriptor]

        if let fetchResults = appDelegate.managedObjectContext!.executeFetchRequest(fetchRequest, error: nil) as? [Hous
            houseModels = fetchResults
        }
    }
```

Adding a entry to a entity

Our next function adds a entry into our House entity.

```
    func addHouseObject(purchasePrice: Double, street: String, city: String, state: String, zip: String, purchaseDate:

        let newHouse = NSEntityDescription.insertNewObjectForEntityForName("House", inManagedObjectContext: appDelegate
        newHouse.purchasePrice = NSNumber(double: purchasePrice)
        newHouse.streetAddress = street
        newHouse.city = city
        newHouse.state = state
        newHouse.zip = zip
        newHouse.purchaseDate = purchaseDate
        newHouse.houseID = NSUUID().UUIDString

        appDelegate.saveContext()
    }
```

Updating an entity

Our next function edits the purchase price of a particular house. Notice that we are using a houseID string to reference the house we will be wanting to update. Our database is configured to guarantee that only one entry exists per house ID.

```
    func updateHouseObject(houseID: String, purchasePrice: Double) {
        let housesToUpdate = houseModels.filter({$0.houseID == houseID})
        if housesToUpdate.count > 0 {
            let house = housesToUpdate[0]
            house.purchasePrice = purchasePrice
            appDelegate.saveContext()
        }
    }
```

Deleting a entry from an entity

Last, let's show the code necessary to delete a house object. This is similar to how we setup edit, in that we will be using a unique houseID to find our house object, and then delete it.

```
func deleteHouseObject(houseID: String) {
    let housesToDelete = houseModels.filter({$0.houseID == houseID})
    if housesToDelete.count > 0 {
        let house = housesToDelete[0]
        appDelegate.managedObjectContext!.deleteObject(house)
    }
}
```

With practice, Core Data can be mastered. Check out the following tutorials which will help expand our above examples with completed code projects.

Tutorials

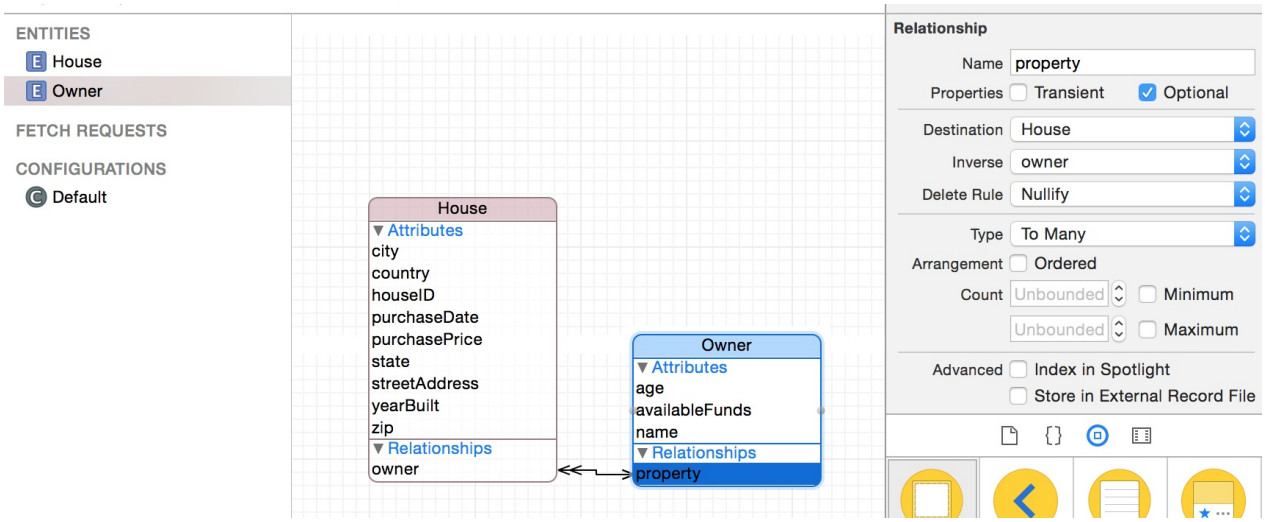
1. Tutorial 1 - Core Data in Swift http://jamesonquave.com/blog/core-data-in-swift-tutorial-part-1/
2. Tutorial 2 - Your First Core Data App with Swift http://www.raywenderlich.com/85578/first-core-data-app-using-swift

Advanced

Once you can read and write to Core Data from within your app, go ahead and explore the following topics.

Relationships

One of the features of a database is the ability to relate objects to each other. For example, houses have owners. A particular owner may have multiple properties.



For more on relationships between entities, check out the Core Data Programming Guide: Relationships and Fetched Properties
https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/Articles/cdRelationships.html

Model Revisions

After deploying your app a new version may require an update to your Core Data model. If that is the case, iOS requires that you generate a model revision. This will provide an automatic path for current users of your app to upgrade into the new model.

For more information on creating model revisions and an upgrade path for your users, check out Core Data Model Versioning and Data Migration Programming Guide here:
https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreDataVersioning/Articles/Introduction.html

Core Data Conceptual Overview

ObjC provides a very good conceptual overivew of Core Data an its internal workings here: http://www.objc.io/issue-4/core-data-overview.html

References

[1] Core Data - http://en.wikipedia.org/wiki/Core_Data

[2] Core Data Programming Guide - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html

[3] Using a Managed Object Model - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/Articles/cdUsingMOM.html

[4] Creating and Deleting Managed Objects - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/Articles/cdCreateMOs.html

[5] NSFetchedResultsController - https://developer.apple.com/library/ios/documentation/CoreData/Reference/NSFetchedResultsController_Class/

[6] NSFetchRequest - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSFetchRequest_Class/index.html

[7] NSManagedObject - https://developer.apple.com/library/ios/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSManagedObject_Class/index.html

[8] NSManagedObjectContext - https://developer.apple.com/library/ios/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSManagedObjectContext_Class/index.html

[9] NSPersistentStore - https://developer.apple.com/library/ios/documentation/Cocoa/Reference/NSPersistentStore_Class/index.html

[10] NSPersistentStoreCoordinator - https://developer.apple.com/library/mac/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSPersistentStoreCoordinator_Class/index.html

[11] NSSortDescriptor - https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSSortDescriptor_Class/index.html

[12] ObjC - Core Data Overview http://www.objc.io/issue-4/core-data-overview.html

Networking

Communicating to external services is essential to the everyday function of many apps. These communications enable us to login, connect and share across social newtorks like Twitter, purchase items and message users. Additionally, retrieving and pushing content to web services enables apps like Instagram and Facebook.

In this section, we're going to cover

1. Asynchronous Programming
2. Networking with iOS
3. AFNetworking Library

References

Learn NSURLSession with Swift - https://medium.com/swift-programming/learn-nsurlsession-using-swift-ebd80205f87c

AFNetworking - https://github.com/AFNetworking/AFNetworking

Concepts and Closures

In this chapter, we will talk about the concept of asynchronous programming, an overview of **REST** and then move onto a new Swift programming idiom, **closures**.

Asynchronous Behavior

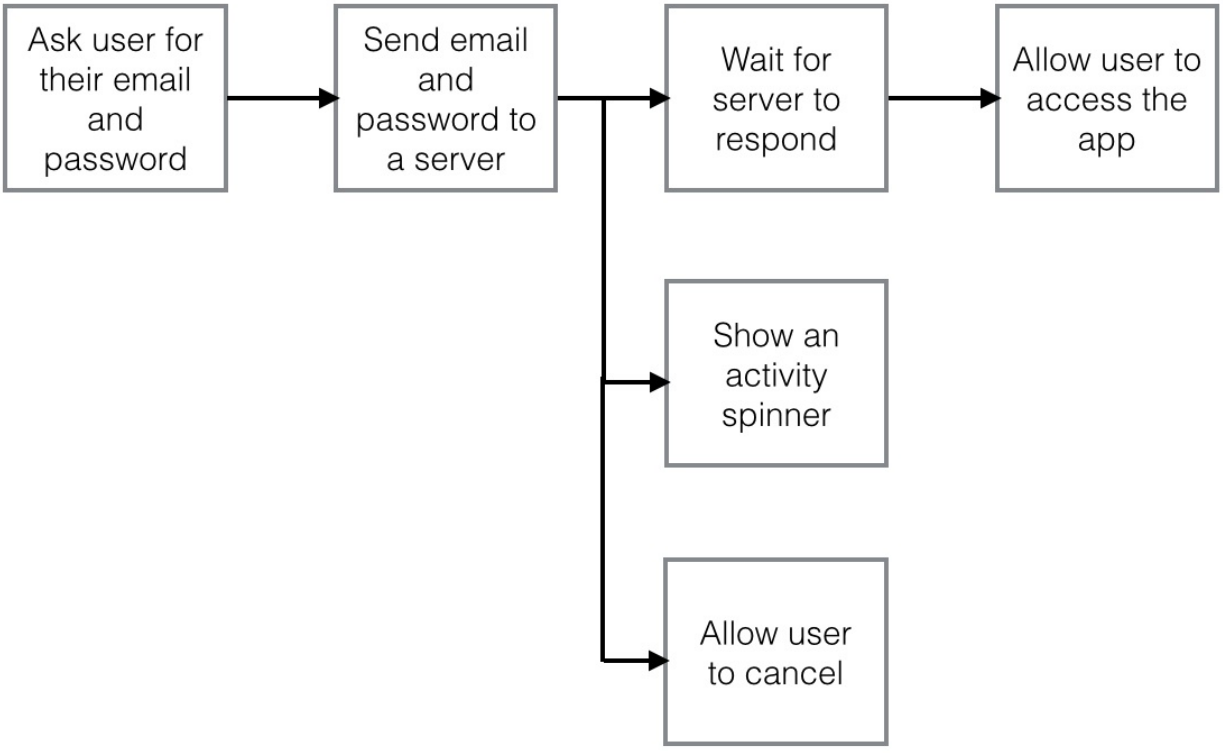
Communicating to a network is essential to perform operations such as logging in, account registration and content sharing.

Let's discuss the concept of asynchornous behavior with an example.

Example - Logging into a server

Let's say you are making an app that asks a user to login to the app with an email address and a password before continuing.

What if you wanted to show an activity spinner so your app still looks responsive? And what if you wanted the user to cancel the login operation if it took too long?



That's asynchronous behavior. We can wait for a web server to return our authentication check, while still allowing the app to appear responsive. We can start jumping in later chapters that can illustrate this behavior

REST

> REST is an architecture style used for creating web services which allow anything connected to a network to communicate with one another via HTTP [2]

REST communicates to the server using the standard HTTP verbs, which are:

- GET - Retrieves data
- POST - Creates new data
- PUT - Updates existing data
- DELETE - Deletes data on the server

Status codes are also defined by HTTP. The most common are

- 200 - OK
- 401 - Unauthorized
- 404 - Not found
- 500 - Internal server error

REST endpoints are typically nouns which describe the action taking place. For example, this is a theoretical server which allows for customer login and customer retrieval based on name:

#	Endpoint	Verb	Parameters	Posted data
1	https://example.org/customer/	GET	?name=donald	None
2	https://example.org/customer/	POST	None	{'user':'jeff', 'password':'1234'}
3	https://example.org/customer/	PUT	?name=jeff	{'password':'9876'}
4	https://example.org/customer/	DELETE	?name=jeff	None

In order, the previous commands

1. Retrieve a customer object named donald
2. Create a new customer object named jeff
3. Update the customer object named jeff
4. Delete the customer object named jeff

Most APIs that you will write against will contain their own detailed API which will describe what the server expects and what the server will return.

In order to implement REST APIs, we will be using closures.

Closures

Programming with the expectation of asynchronous execution requires certain blocks of functionality will be executed at different times. Swift includes **closures** for just this purpose.

> Closures are self-contained blocks of functionality that can be passed around and used in your code [1]

Code

Let's check out an example where we are expecting a block of code to execute after some asynchronous behavior completes.

```
func helloIsAnyoneThere(completion:(response: String) -> Void) {
    sleep(10) // wait for 10 seconds
```

```
    completion(response: "Yes, I am here")
}

NSLog("Let's see if anyone is there...")
helloIsAnyoneThere { (response) -> Void in
    NSLog("We received a response!  And that response is \(response)")
}
```

The above code will be calling a function with a closure as a parameter. The function itself is going to sleep for 10 seconds, this is to simulate a long running process or event. Once those 10 seconds are complete, the closure will be called.

```
16:35:50 MyPlayground: Let's see if anyone is there...
16:36:00 MyPlayground: We received a response!  And that response is Yes, I am here
```

Note that the timestamps are 10 seconds apart. If you were to cut and paste this code example into Playgrounds, you'll notice that delay upon execution.

The syntax to define a closure is shown on our first code block and can be written in this way:

```
{ (parameters) -> return type in
    statements
}
```

Closures are used in other ways as well. Inline functions, sorting blocks and others are discussed in Apple's Closure documentation [see 1 below].

References

[1] Closures - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html#//apple_ref/doc/uid/TP40014097-CH11-XID_151

[2] REST - http://en.wikipedia.org/wiki/Representational_state_transfer

Networking APIs

Communicating to remote APIs is a common occurance for mobile applications. You can rely on centralized servers for user authentication and content generation into your app. Facebook uses an API to authenticate you, retrieve your news feed, pictures, comments and perform actions such as generating a new friend request.

Apple provides a set of classes for connecting to remote URLs including APIs. That collection is centered around **NSURLSession**.

Definitions

As in previous chapters, let's frame our discussion with terms and classes used in this chapter:

- **JSON** - JavaScript Object Notation - A readable text format used to communicate simple objects such as strings and integers.
- **NSURLConnection** - A class that lets you load the contents of a URL by providing a URL request object. [2]
- **NSURLRequest** - Represents a URL load request in a manner independent of protocol and URL scheme. [3]
- **NSURLSession** - Provides an API for downloading content via HTTP. This API provides a rich set of delegate methods for supporting authentication and gives your app the ability to perform background downloads. [1]
- **SSL** - Secure Sockets Layer, a protocol for securing a connection between two endpoints which allows for encrypted transmissions.

Code Examples

The two most common examples of talking to an external server are downloading a file as well as GET and POST commands.

Downloading a file

The following code will download an image file from wikipedia and once downloaded, will return that file within the response closure:

```
import UIKit
import XCPlayground // Specific for Playgrounds

XCPSetExecutionShouldContinueIndefinitely() // Allows us to run asynchronous code in Playgrounds

let url = NSURL(string: "http://en.wikipedia.org/wiki/File:Cuernos_del_Paine_from_Lake_Peho%C3%A9.jpg")!

var session = NSURLSession.sharedSession()
let requestTask = session.dataTaskWithRequest(NSURLRequest(URL: url), completionHandler: { (data, response, error) -> \
    NSLog("response from our request was \(response)")
})

NSLog("Going to begin downloading the file")
requestTask.resume()
```

And the above console output is:

```
2015-01-11 17:24:04.005 MyPlayground2[82717:16247271] Going to begin downloading the file
2015-01-11 17:24:04.214 MyPlayground2[82717:16247321] response from our request was <NSHTTPURLResponse: 0x7fae92007760>
```

Please note that when downloading larger files, we can start looking at NSURLSession delegates to manage resuming

downloads should a failure occur.

Primarily, we will be working with GET and POST to web services. The following code demonstrates these two commands

GET

The following code will submit a GET to Google for querying the term Swift. The code will then output the result to the console.

Note that query parameters are added inato the HTTPAdditionalHeaders.

```
import Foundation
import XCPlayground // Specific for Playgrounds

XCPSetExecutionShouldContinueIndefinitely(continueIndefinitely: true) // Allows us to run asynchronous code in Playgrou

let parameters = Dictionary(dictionaryLiteral: ("q", "swift"))
var config = NSURLSessionConfiguration.defaultSessionConfiguration()
config.HTTPAdditionalHeaders = parameters

var session:NSURLSession = NSURLSession(configuration: config)
let url = NSURL(string: "http://google.com")!

let task = session.dataTaskWithURL(url, completionHandler: { (data, response, error) -> Void in
    if error != nil {
        print("Error trying to GET from Google \(error)")
    }
    else {
        var result = NSString(data: data, encoding:
            NSASCIIStringEncoding)!
        print("We retrieved the data as \(result)")
    }
})
task.resume()
```

POST

POST is different than GET in that the data not sent in the URL but is instead sent as a separate payload, and is typically JSON.

In the below example, we are going to be posting a JSON string and outputting the result to the console.

```
import XCPlayground // Specific for Playgrounds

XCPSetExecutionShouldContinueIndefinitely() // Allows us to run asynchronous code in Playgrounds

let url = NSURL(string: "http://your-post-url-here")! // make sure to put a valid URL!
let jsonString = "{'test':12345}"
var urlRequest = NSMutableURLRequest(URL: url)
urlRequest.HTTPMethod = "POST"
urlRequest.HTTPBody = jsonString.dataUsingEncoding(NSUTF8StringEncoding, allowLossyConversion: true)

var session = NSURLSession.sharedSession()

let requestTask = session.dataTaskWithRequest(urlRequest, completionHandler: { (data, response, error) -> Void in
    if error != nil {
        print("we had an error \(error)")
    }
    else {
        var result = NSString(data: data, encoding:
            NSASCIIStringEncoding)!
        print("response from our request was \(result)")
    }
})

print("Going to begin POSTing")
requestTask.resume()
```

The APIs that you'll be connecting to will provide direction on proper endpoints and expected data payloads as well should provide expected response data for both success and failure conditions.

[Advanced] HTTPS and SSL

The majority of API connections should be requiring an SSL connection. SSL encrypts the data between the phone and the server which makes it difficult for hackers to eavesdrop on the data being exchanged.

NSURLSession has delegate methods, namely didReceiveChallenge that can be implemented to handle SSL connections correctly. Please see the following blog for more details:

https://medium.com/swift-programming/learn-nsurlsession-using-swift-ebd80205f87c

Reference

Learn NSURLSession with Swift - https://medium.com/swift-programming/learn-nsurlsession-using-swift-ebd80205f87c

iOS NSURLSession with Swift - http://hayageek.com/ios-nsurlsession-example/

[1] NSURLSession Class Reference
https://developer.apple.com/library/ios/documentation/Foundation/Reference/NSURLSession_class/

[2] NSURLConnection Class Reference
https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/index.html

[3] NSURLRequest Class Reference
https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSURLRequest_Class/

AFNetworking

http://afnetworking.com

AFNetworking is a third-party framework which is used to communicate to remote web services. We use third-party here to distinguish from frameworks created by Apple.

Since AFNetworking is an Ojbective-C framework, some additional work is needed to bring it into our Swift projects.

Installation

Installing AFNetworking can happen with CocoaPods. Described in more detail here (https://github.com/AFNetworking/AFNetworking). We will be using CocoaPods to install AFNetworking. The podfile should include:

```
pod "AFNetworking"
```

Then run a `pod install`. This is described in the CocoaPods chapter, so take a look and jump back here when you're ready.

Example AFNetworking calls

The basic premise of any **REST API** is a data transfer using standard HTTP verbs of GET, POST, PUT and DELETE. If you need a refresher, we covered REST here.

Code snippets of example GET and POST commands are found on the following blog: http://www.sroze.io/2014/07/25/ios-swift-and-afnetworking-get-response-data-and-status-code-in-case-of-failure/

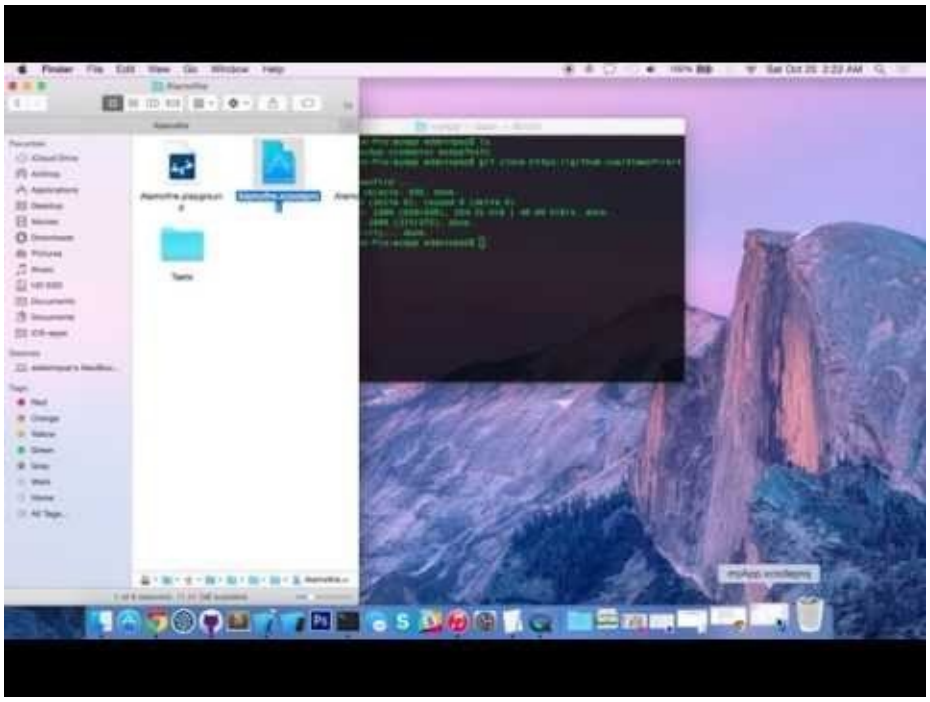
AFNetworking was recently convereted to a Swift library. That library is called Alamofire. You may choose to use this as an alternative to AFNetworking to find more Swift-specific examples available on the web.

Alamofire

Since moving to Swift, the new hotness is to use Alamofire, created by the same Matt Thomspon of AFNetworking.

Installation

For installation, we will be using CocoaPods (we have a chapter on CocoaPods coming up) but CocoaPods isn't availbe yet with Alamofire. Instead, there is a video available which goes through the process: We can start with an intro video which will go through the process of installing the third-party library that we mentioned previously.



https://www.youtube.com/watch?v=HJu0aIdJawk

Why use Alamofire over NSURLSession?

The reason a developer would choose to use a third party library instead of iOS libraries is the reduction of code and simplicity of use.

Let's check out the following examples:

GET

Notice how this GET command uses significantly less code than our previous example found in Networking APIs

```
Alamofire.request(.GET, "http://httpbin.org/get", parameters: ["foo": "bar"])
         .response { (request, response, data, error) in
                     println(request)
                     println(response)
                     println(error)
                   }
```

Using this third party library enables faster development time and a more refined focus on finding solutions.

POST

The POST retains the same code reduction as the above GET.

```
let parameters = [
    "foo": "bar",
    "baz": ["a", 1],
    "qux": [
        "x": 1,
        "y": 2,
        "z": 3
    ]
]
```

```
Alamofire.request(.POST, "http://httpbin.org/post", parameters: parameters)
```

The Alamofire framework is very powerful and robust. We covered just the basics which should cover the majority of initial API integration that you may encounter. Feel free to keep exploring their README.md file, which contains many more examples for a variety of situations.

References

Alamofire on GitHub https://github.com/Alamofire/Alamofire

AFNetworking on GitHub https://github.com/AFNetworking/AFNetworking

Alamofire - NSHipster http://nshipster.com/alamofire/

Further Resources

The following section will outline resources available to you for finding useful code, projects and other resources.

1. CocoaPods - A third-party dependency manager that integrates with Xcode
2. Existing Code - A list of blogs, articles and source code resources that will help you on your journey.
3. Finding Help - How to get unstuck.

All resources are linked within each individual chapter.

CocoaPods

http://cocoapods.org

CocoaPods is a third-party dependancy manager for Xcode. It is a free utility that was created using Ruby. You install it via Terminal and configure individual Pods using a text editor. Pods refer to third-party libraries that you will be wanting to install. After the first installation of a Pod, CocoaPods will convert your Xcode project into a workspace, which should be used for all subsequent development.

Definitions

Let's frame our discussion with the terms you will encounter when using CocoaPods:

- CocoaPods - Depedency manager for Xcode
- Pods - A collection of instructions consumed by CocoaPods to download and install third-party libraries.
- Podfile - Instructions contained within a text file, used by CocoaPods to install, update and remove pods
- Ruby - A OO language that was used to develop CocoaPods.
- Gem - RubyGems is a package manager for the Ruby programming language that provides a standard format for distributing Ruby programs and libraries [2]

Installation of CocoaPods

Installation of CocoaPods involves opening Terminal and entering a single command.

1. Open Terminal
2. Type

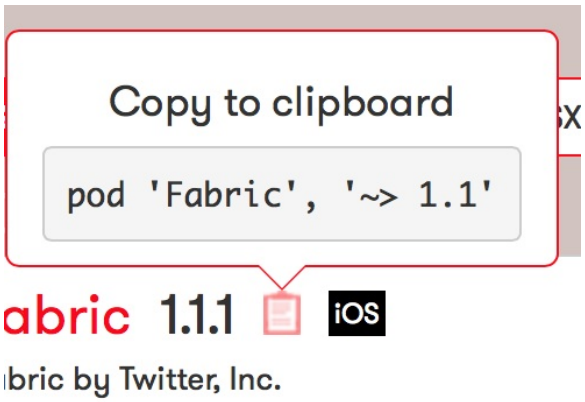
```
sudo gem install cocoapods
```

3. That's it!

Finding and installing Pods

Let's say you want to install Fabric by Twitter.

1. Go to http://cocoapods.org
2. Type "Fabric" in the search bar



3. Copy the pod string by selecting the clipboard
4. Go to your project directory in a Terminal window
5. Type

```
pod init
```

6. Open your favorite text editing utility

```
nano Podfile
```

7. Enter the Fabric pod string

```
pod 'Fabric', '~> 1.1'
```

8. Close your text editor and type

```
pod install
```

That's it! CocoaPods will handle downloading all necessary pods, install them and configure your project to handle this an any required frameworks.

Maintaining Pods

Occasionally, pods will be updated with new versions. To update to the latest, you simple need to run the installation script again.

```
pod install
```

Removing Pods

To remove a Pod, edit your Podfile with the text editor of your choice. Delete the line that represents the pod you'd like to uninstall. When complete, type

```
pod install
```

Notice a pattern? :) pod install is a one-stop shop for isntalling and updating your pods.

References

CocoaPods http://cocoapods.org

[2] RubyGems http://en.wikipedia.org/wiki/RubyGems

Existing Code

A lot can be learned from using existing code templates. Below is a list of source code resources to consider when starting a new project or updating an existing one.

Blogs

These are a list of technical blogs maintained by some of the best in the industry.

ObjC.io

http://www.objc.io/issue-16/

A periodical about best practices and advanced techniques for iOS and OS X development.

NSHipster

http://nshipster.com

NSHipster is a journal of the overlooked bits in Objective-C, Swift, and Cocoa. Updated weekly.

RAY WENDERLICH

http://www.raywenderlich.com

RAY WENDERLICH - Tutorials for developers & Gamers

GitHub

Although we've used GitHub as an interface to our Git repositories, the website is more purposed for the sharing of public and open source repositories. Here are few favorites.

Trending Swift repositories

https://github.com/trending?l=swift Find what repositories the GitHub community is most excited about today.

Alamofire

https://github.com/Alamofire/Alamofire Elegant HTTP Networking in Swift

SwiftyJSON

https://github.com/SwiftyJSON/SwiftyJSON The better way to deal with JSON data in Swift

Apple Development Videos

Apple has WWDC (World wide developer conferences) each year. In addition, they recently have begun Tech Talks which are smaller conferences aimed specifically at iOS.

Here are some direct links to those resources:

Tech Talks - iOS 7

https://developer.apple.com/tech-talks/videos/

WWDC 2014 - Videos and PDFs

https://developer.apple.com/videos/wwdc/2013/

WWDC 2013 - Videos and PDFs

https://developer.apple.com/videos/wwdc/2013/

Finding Help

Programming is sometimes referred to as a continual process of becoming stuck and trying to become unstuck. The former will come naturally. Here are the resources and processes that will help you on the latter.

Conceptual Process

Derived from the blog at Harvard, Getting Unstuck: http://www.gse.harvard.edu/news/uk/14/10/getting-unstuck

1. **Read through your code**. “It sounds obvious, but a surprising number of kids would just throw out their projects if they didn’t work. Over time, kids realized that they needed to be more analytical and critical about their work,” Brennan says.
2. **Experiment with your code.** If you can’t find the source of the problem, tinker.
3. **Look for examples**. Kids can use the rich online community in the Scratch site — home to more than 6.3 million projects — to find examples of what they want to do and build something new.
4. **Work with someone else.** Kids learned that they could build more together than they could individually, Brennan says.
5. **Be persistent**. All of the kids she surveyed talked about the challenges and the great satisfaction that comes with programming, Brennan says. They also learned when it was time to take a break.

Here are list of resources that will help you find answers to programmings questions:

Stack Overflow

http://stackoverflow.com

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Recommendation is to use Google as your search into Stack Overflow.

ASCIIwwdc

http://asciiwwdc.com

Find the content you're looking for, without scrubbing through videos

Apple Developer Forums

https://developer.apple.com/devforums/

The Apple Developer Forums are a great place to post questions and share comments with fellow developers **and Apple engineers**. Discuss a variety of development topics, from getting started to working with the latest pre-release software.

The App Store

Congratulations!

First of all, congratulations on getting to this point with your new iOS app! You've successfully:

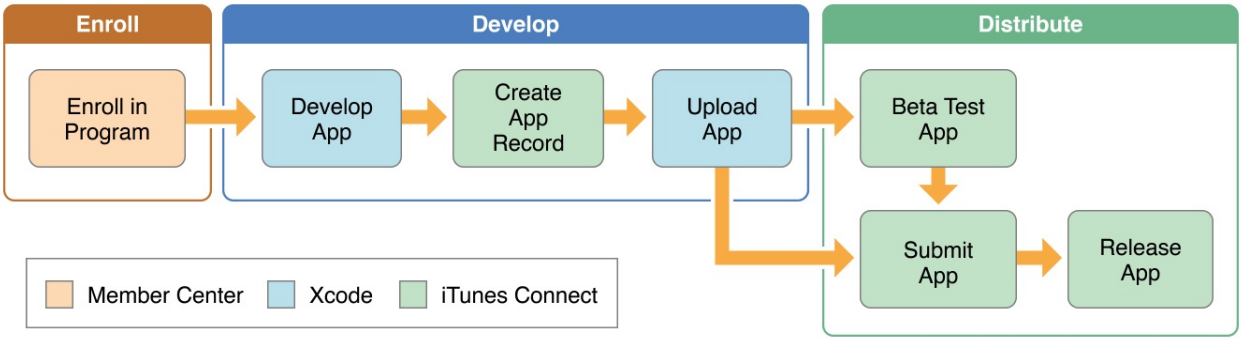
- Learned the Xcode development environment
- Mastered managing code using GitHub
- Learned Swift programming language and associated iOS SDKs
- Built, tested and debugged an iOS mobile app

This was no easy feat, so take a moment to reflect on your accomplishments!

Overview

The following chapters will provide a reference for the following concepts:

1. Provisioning Profiles
2. iTunes Connect
3. Build and Submit your IPA
4. The Review Process



Reference

Each sub-section will refer to its own references. For a detailed overview from Apple's documentation, please reference:

https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html

Provisioning

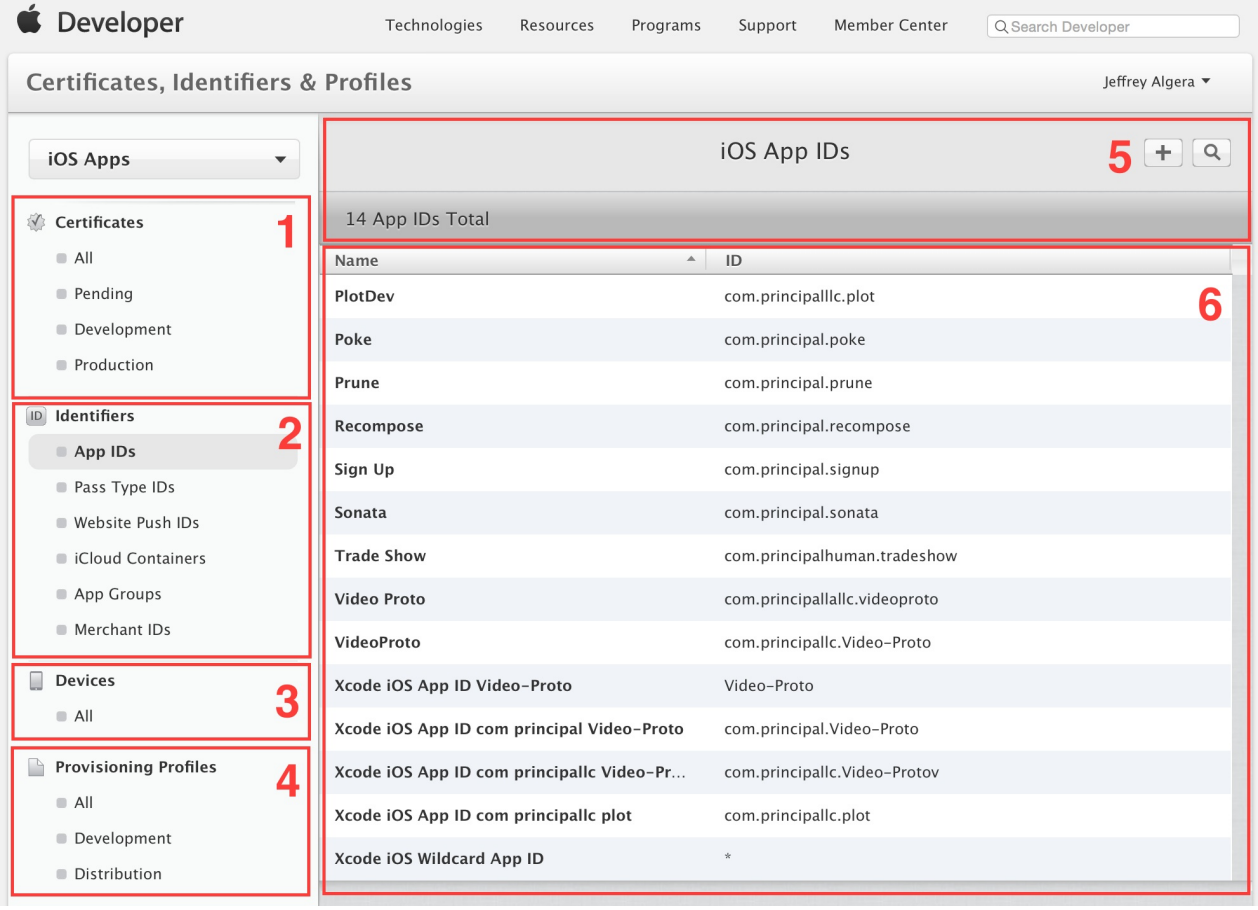
Apple requires an app to be provisioned before allowing the app to be released on the iTunes App Store. They enforce this by linking your computer to your developer profile. They then use that link to digitally sign your app. This proves to Apple that you are a human, that you have a developer profile and that you are the author of your app.

To provision your app, take the following three steps

Each of the steps will be explained in detail below, but to summarize, the three steps are:
Step 1 - Create a production certificate
Step 2 - Register your app ID and
Step 3 - Create your distribution provisioning profile

The Apple Developer Interface

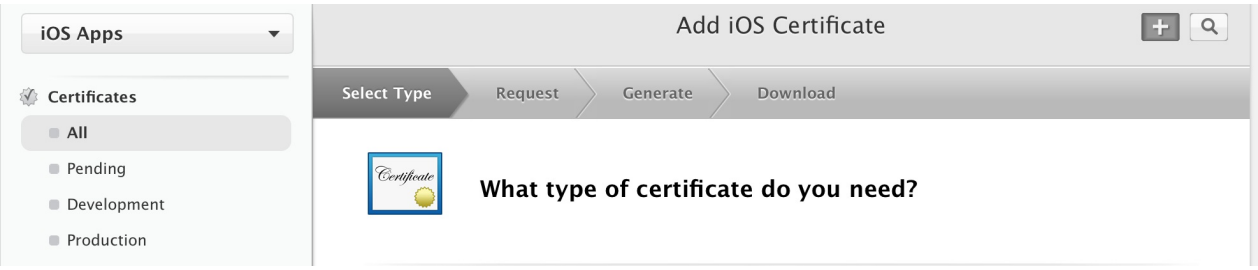
To accomplish the above tasks, we will be using the Apple Developer portal as found here: http://developer.apple.com
Below, we have the certificates interface outlined and labeled with numbers. Those numbers correspond to the definitions below:



1. Certificates - A list of certificates directly created by you, your team, or automatically created via Xcode.
2. Identifiers - A list of App IDs. Each released app has its own App ID.
3. Devices - A list of devices that are added to your developer account. Each device is allowed to install a debug or AdHoc build of your app.
4. Provisioning Profiles - A list of active and expired provisioning profiles for your developer account.
5. Control bar - titles the active selection and provides a "+" button for creating more.
6. Detail list pane - A detail list of the content corresponding to the currently selected section

---

Step 1: Creating a production certificate



1. Select "Certificates" -> "+" Button (as pictured above)
2. Under Production, select "App Store and Ad Hoc"
3. Apple will walk you through created a CSR through the application Keychain Access on your Mac

```
Within the Keychain Access drop down menu, select Keychain Access > Certificate Assistant >
Request a Certificate from a Certificate Authority.

In the Certificate Information window, enter the following information:
1.  In the User Email Address field, enter your email address.
2.  In the Common Name field, create a name for your private key (e.g., John Doe Dev Key).
3.  The CA Email Address field should be left empty.
4.  In the "Request is" group, select the "Saved to disk" option.
5.  Click Continue within Keychain Access to complete the CSR generating process.
```

4. Upload your CSR file to Apple
5. Apple will create a Production certificate for you which you can now download and install this certificate by double clicking it from the Finder.

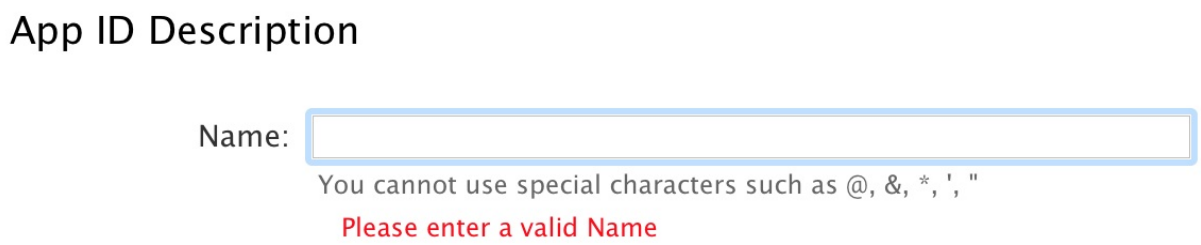
Step 2: Creating a new App ID

From within Identifiers, you can select App IDs. Then hit the "+" sign to create a new app ID

There main sections of creating an App ID are

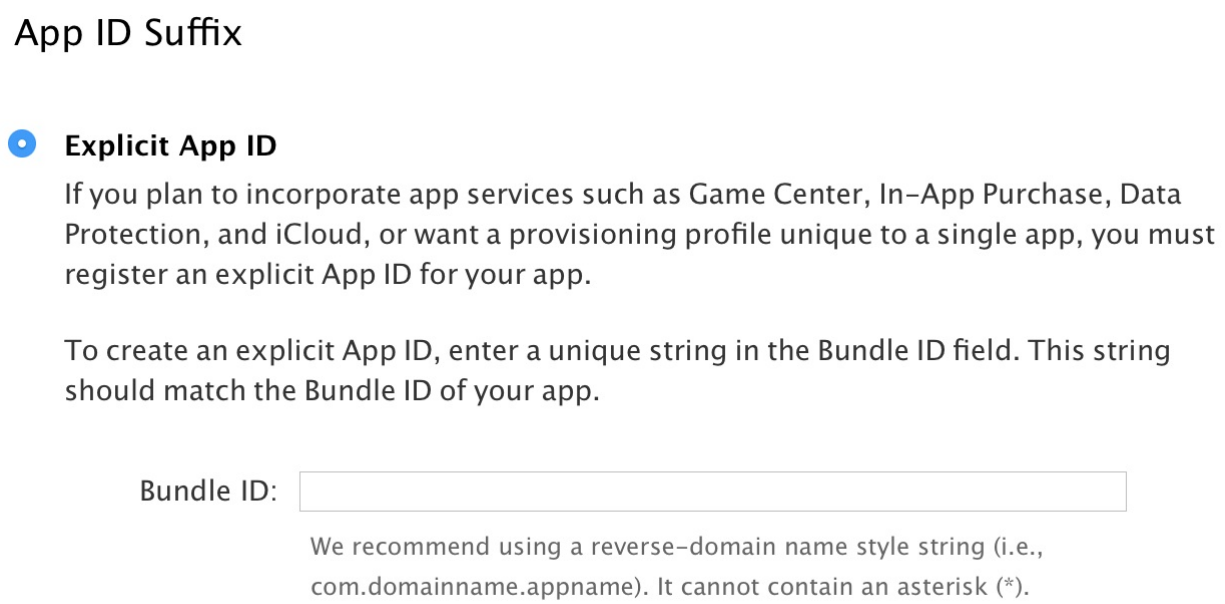
App ID Description

This is user-facing string used to reference this particular App ID. Examples include your app name, e.g. "Awesome App"



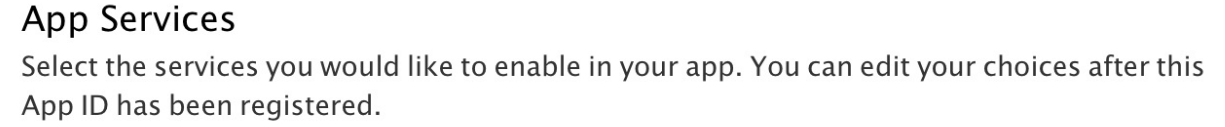
App ID Suffix

This is also known as your bundle ID. The bundle ID will be how you reference your app in iTunes Connect as well as within the rest of this Apple Developer Portal. We typically ally use a reverse domain style, e.g. com.generalassembly.awesome-app



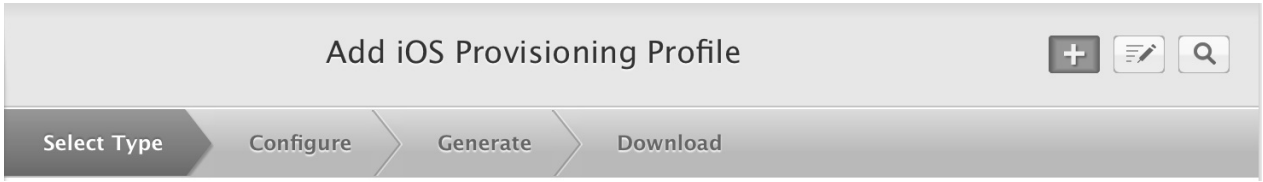
App Services

These represent a list of services that are available to your app. They must be chosen before the app is built and released. Think through each option and decide if you'd like to utilize these services.



Step 3: Creating a Distribution Provisioning Profile

To create a provisioning profile for use on the iTunes App Store, select Provisioning Profiles -> Distribution -> "+" button.



Apple will walk you through creating the profile. You will be selecting your Distribution Certificate from step 1 (above) and your bundle ID from step 2 (above) in order to complete this process.

Phew

We made it! Now onto our next step, iTunes Connect

References

Apple Developer Portal - http://developer.apple.com

Demystifying iOS certificates and provisioning files - http://escoz.com/blog/demystifying-ios-certificates-and-provisioning-files/

iTunes Connect

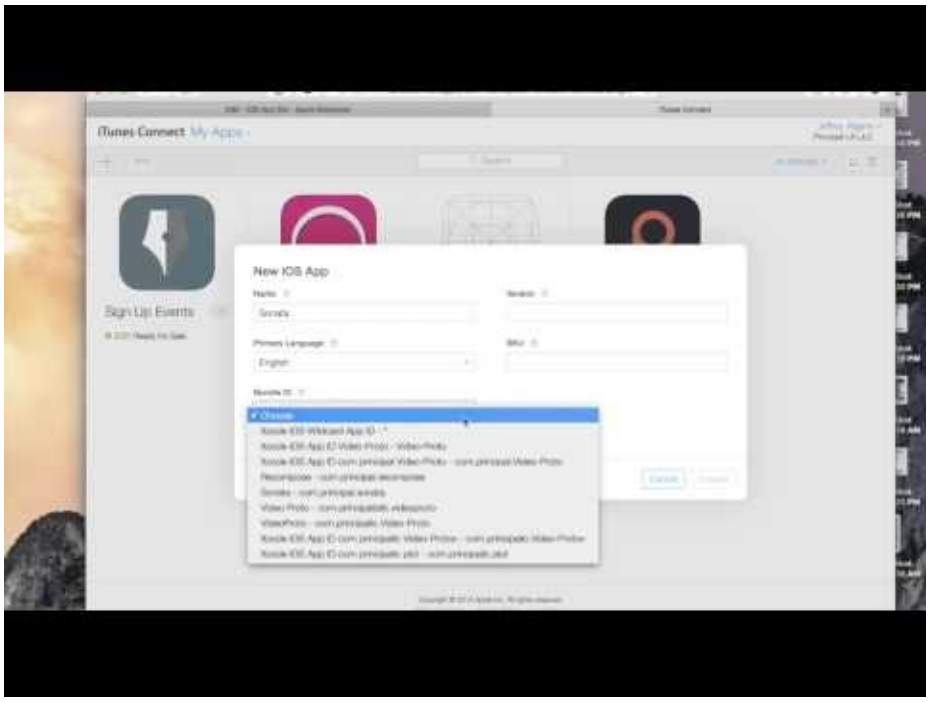
iTunes Connect provides a portal for managing your apps and items related therein.

1. **My Apps** - Provides a list of apps that you have configured through ITC, as well as allows you to create new builds
2. **Sales and Trends** - Provides an interface for viewing sales and downloads across various regions throughout the world
3. **Payments and Financial Reports** - Provides a configurable list of payments and reports that have been disperesed into your banking account.
4. **iAd** - used to configure iAd within your apps
5. **Users and Groups** - Used for creation of ITC test users (for testing IAP) and for granting additional developers access on a team.
6. **Agreements**, Tax and Banking - Used for formal contracts, banking information and tax forms.
7. **Resources and Help** - provides a FAQ as well as a contact form for getting in touch with the iTunes Connect team.

Preparing your app for upload

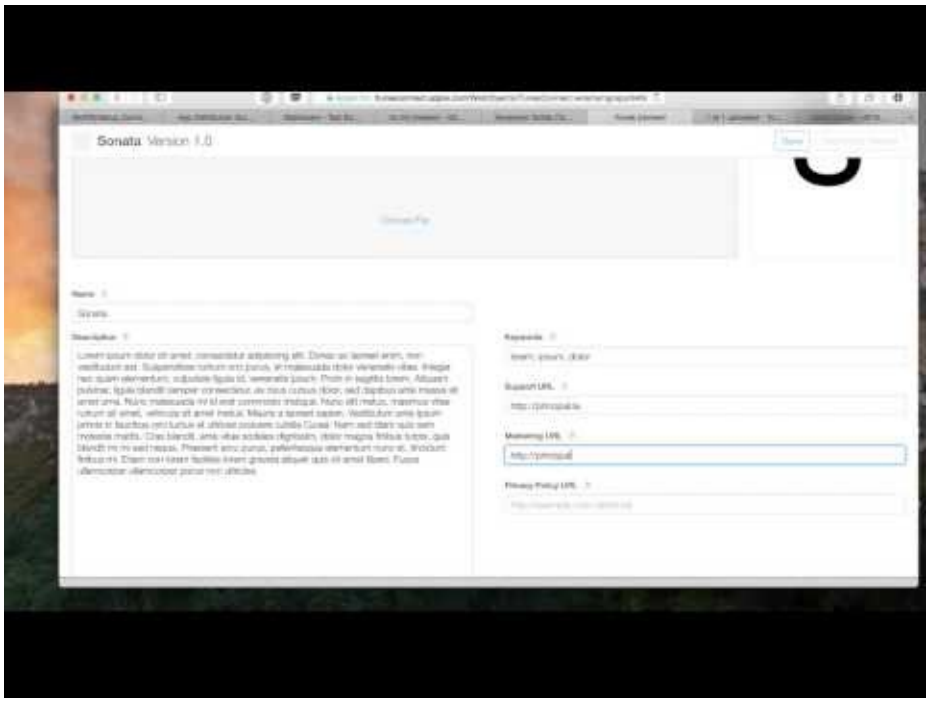
The following video will guide you through creating a new app on iTunes Connect and prepare you for uploading that app.

Step 1: App Setup



http://www.youtube.com/watch?v=JABIKx3UaQQ

Step 2: Adding metadata and images



http://www.youtube.com/watch?v=UPx4spZ7bE8

Next steps

Build and submit your app to iTunes Connect via Xcode

Reference Materials

Screenshot sizes for portait iPhone app previews

iPhone Type	Size	Pixel Width	Pixel Height
iPhone 4/4s	3.5 inch	640	960
iPhone 5/5s	4 inch	640	1136

iPhone 6	4.7 inch	750	1334
iPhone 6 Plus	5.5 inch	1242	2208

iTunes Connect App Properties - https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Appendices/Properties.html

Managing your app with iTunes Connect https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/UsingiTunesConnect/UsingiTunesConnect.html

Build and Submit

Process

At this point, we have successfully created the necessary provisioning profiles and certificates. We've also configured out build in iTunes Connect and have added the necessary metadata.

[Video] How To Submit An App To Apple's App Store

via Paul Flahan on YouTube

http://www.youtube.com/watch?v=rRlOdp4uZoo#t=6m26s

Step by Step

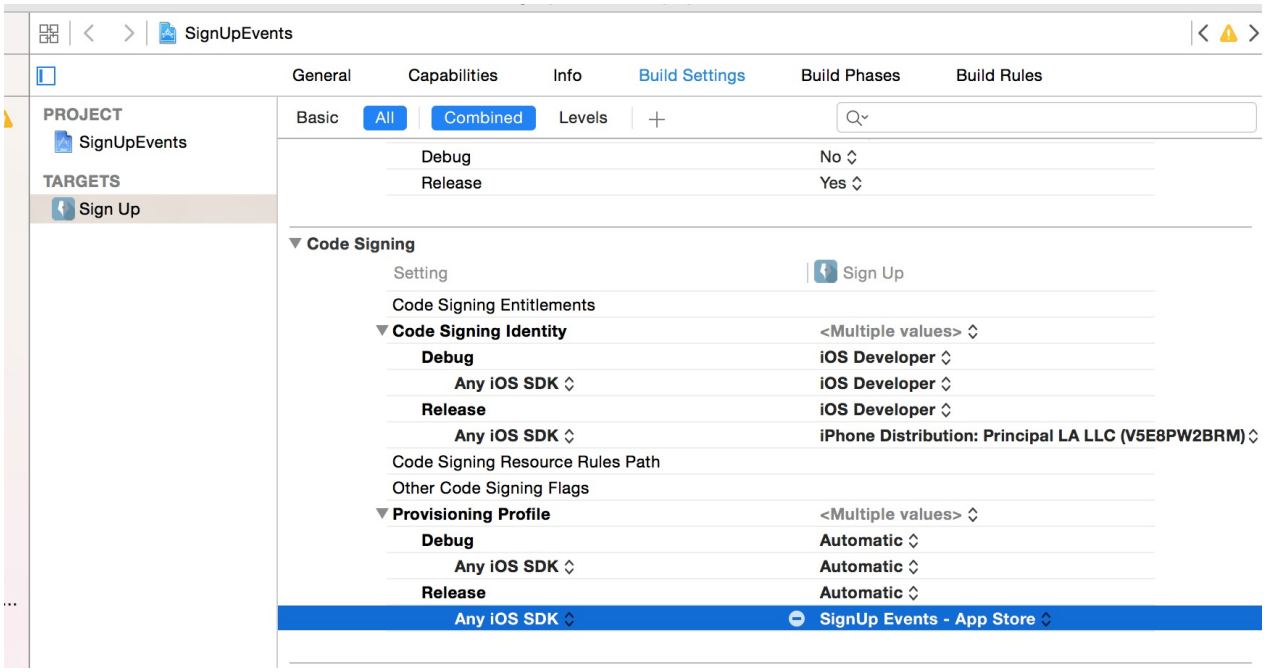
We've listed 9 steps, below, that go through the submission process. Each one references images. Each step is also covered in the above video.

Step 1

Ensure the Bundle Identifier in your project settings matches the ID we setup in Provisioning and iTunes Connect.

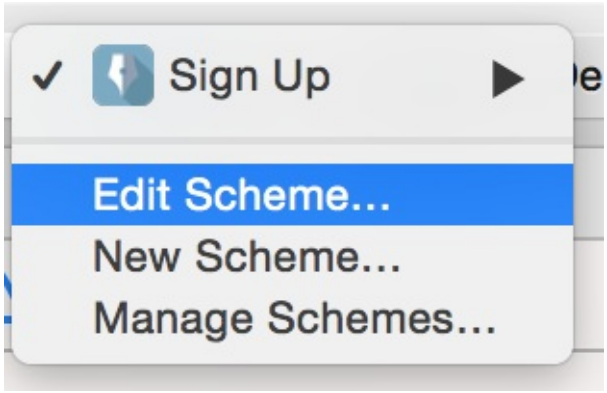
Step 2

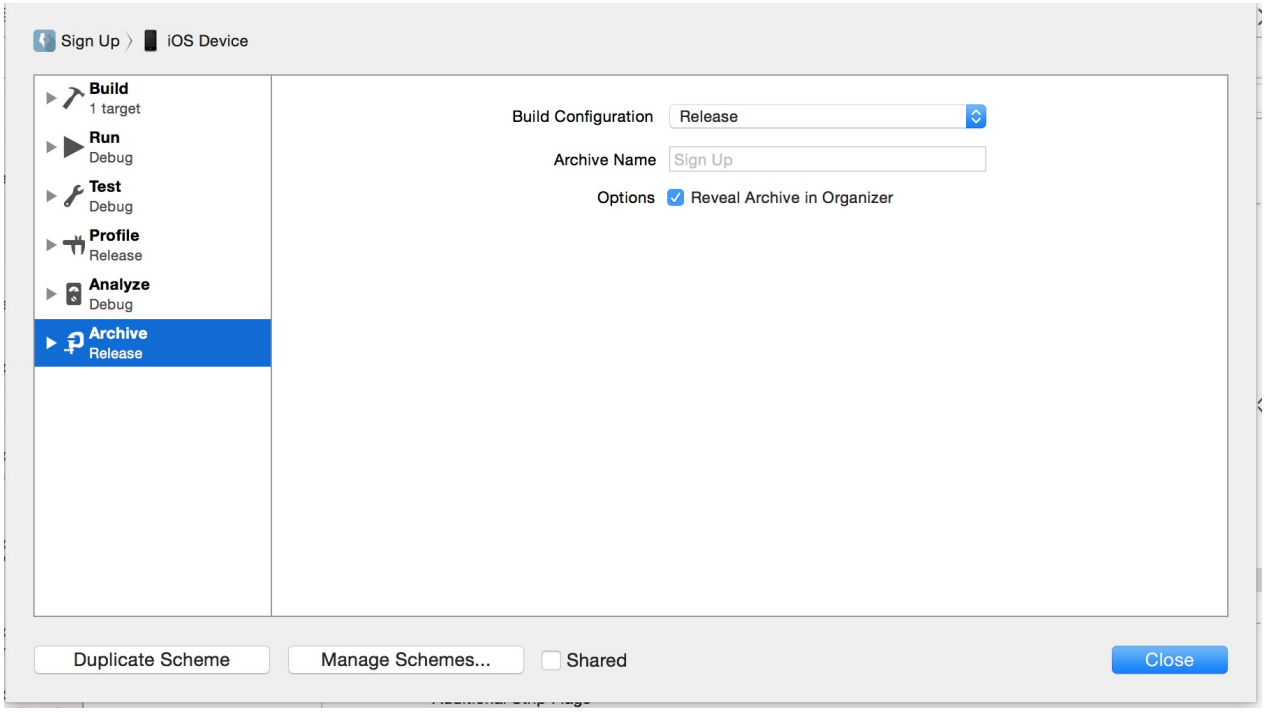
Select the appropriate Code Signing Identity and the appropriate Provisioning Profile



Step 3

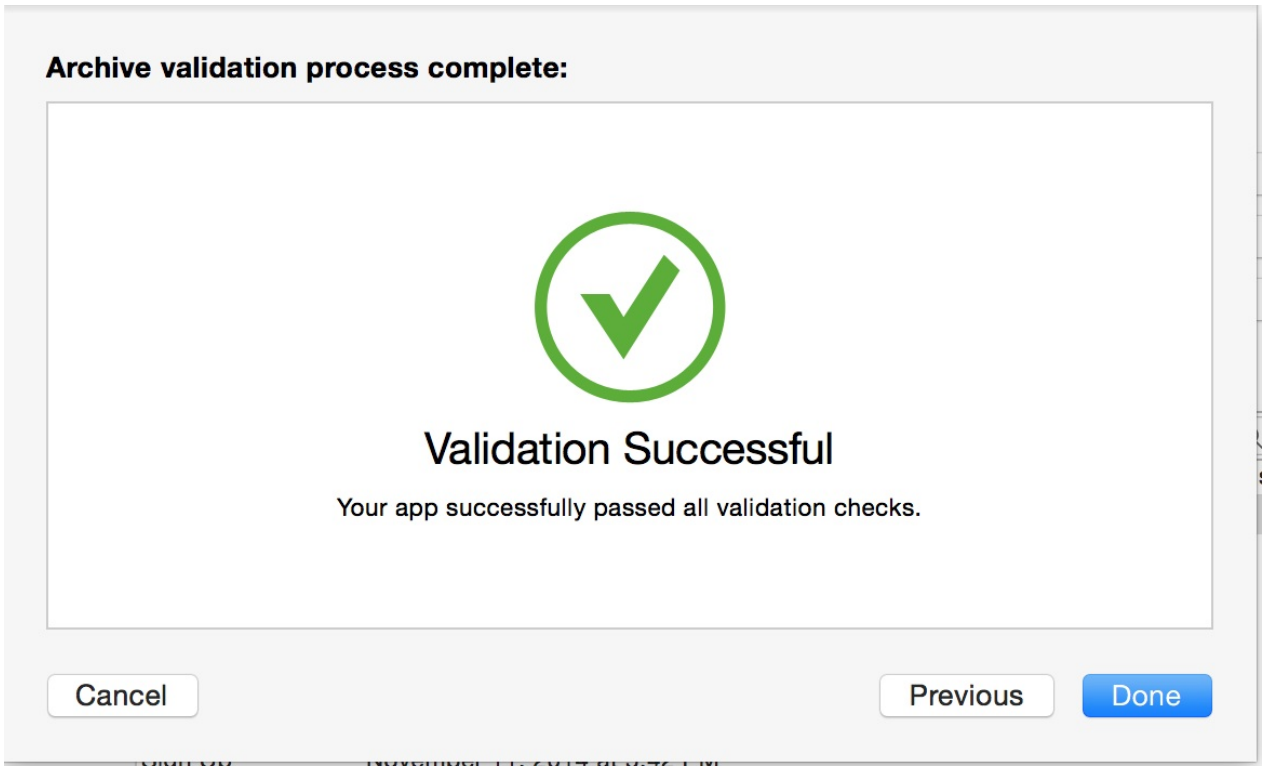
Ensure scheme is set to Archive.





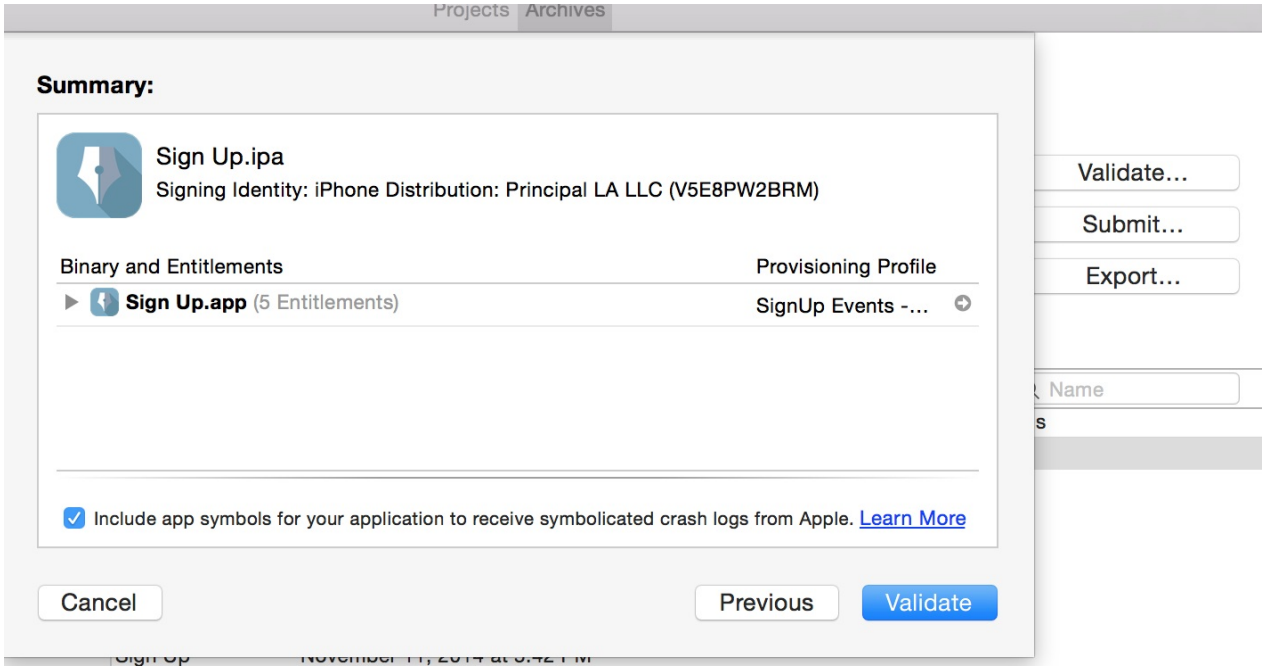
Step 4

Step Select Product -> Archive



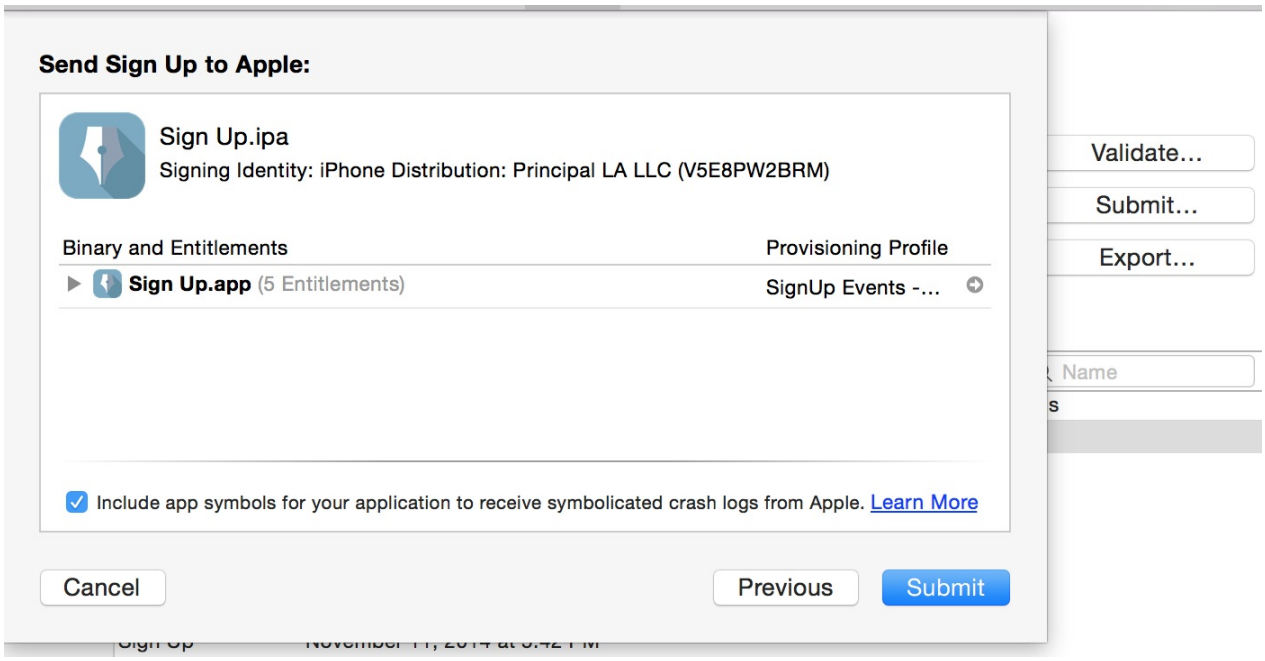
Step 5

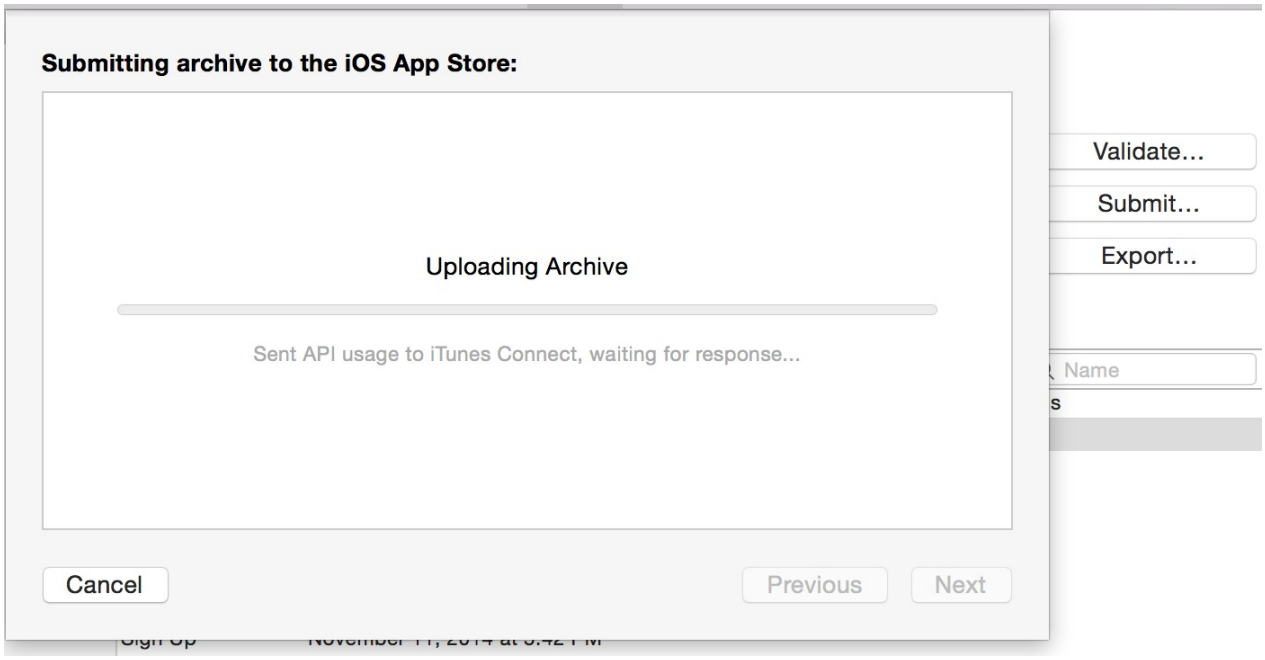
Validate build with Organizer by selecting the Validate button.

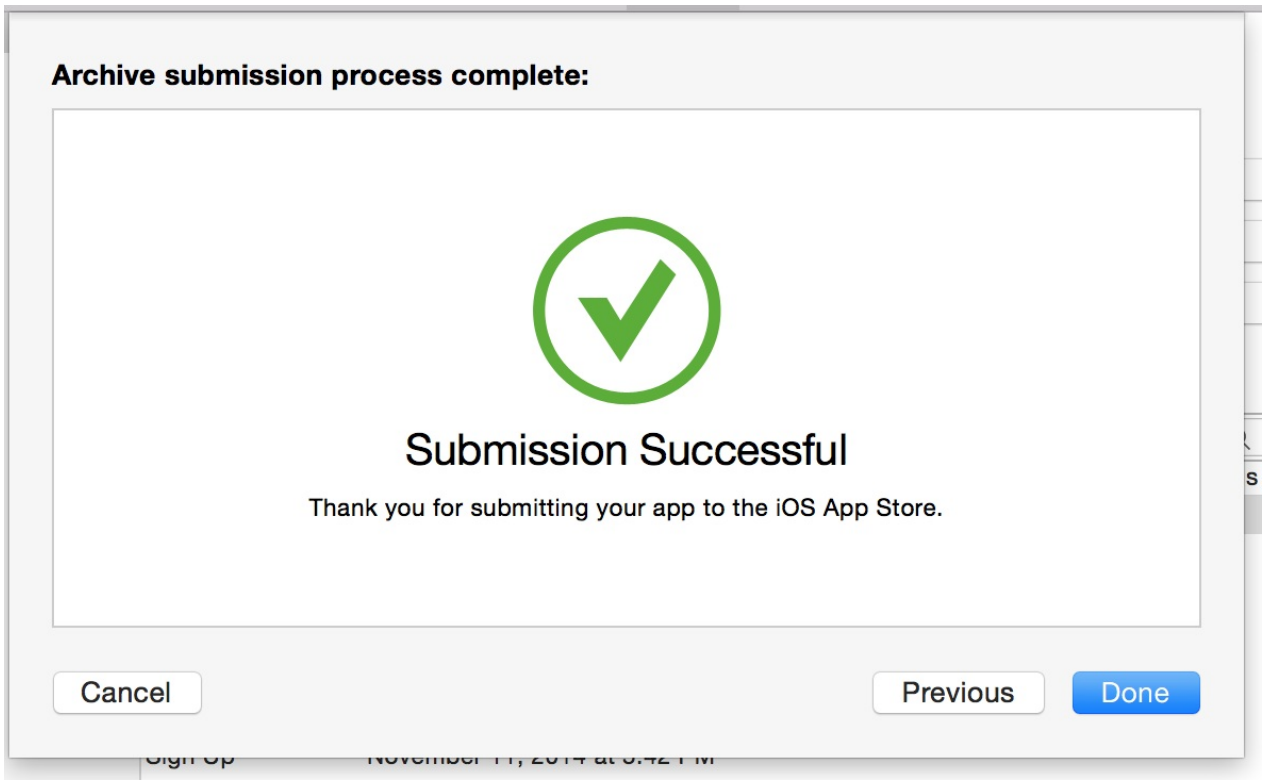


Step 6

Submit build to iTunes App Store with Organizer


Send Sign Up to Apple:
Sign Up.ipa
Signing Identity: iPhone Distribution: Principal LA LLC (V5E8PW2BRM)
Binary and Entitlements
Provisioning Profile
Sign Up.app (5 Entitlements)
SignUp Events -...
Include app symbols for your application to receive symbolicated crash logs from Apple. Learn More
Cancel
Previous
Submit
Validate...
Submit...
Export...
Name


Submitting archive to the iOS App Store:
Uploading Archive
Sent API usage to iTunes Connect, waiting for response...
Cancel
Previous
Next
Validate...
Submit...
Export...
Name

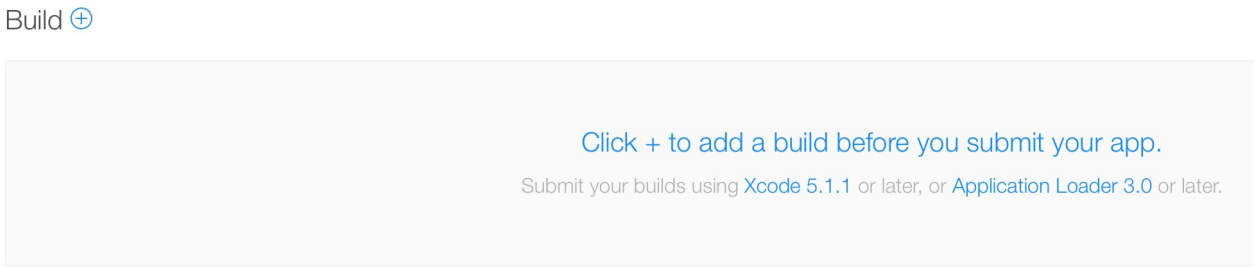


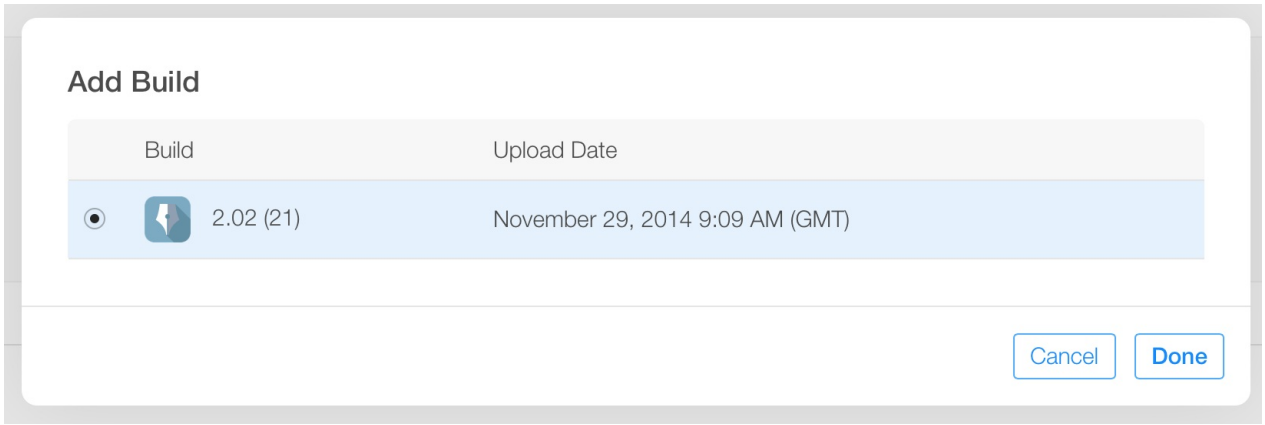
Step 7

Return to iTunes Connect (http://itunesconnect.apple.com)

Step 8

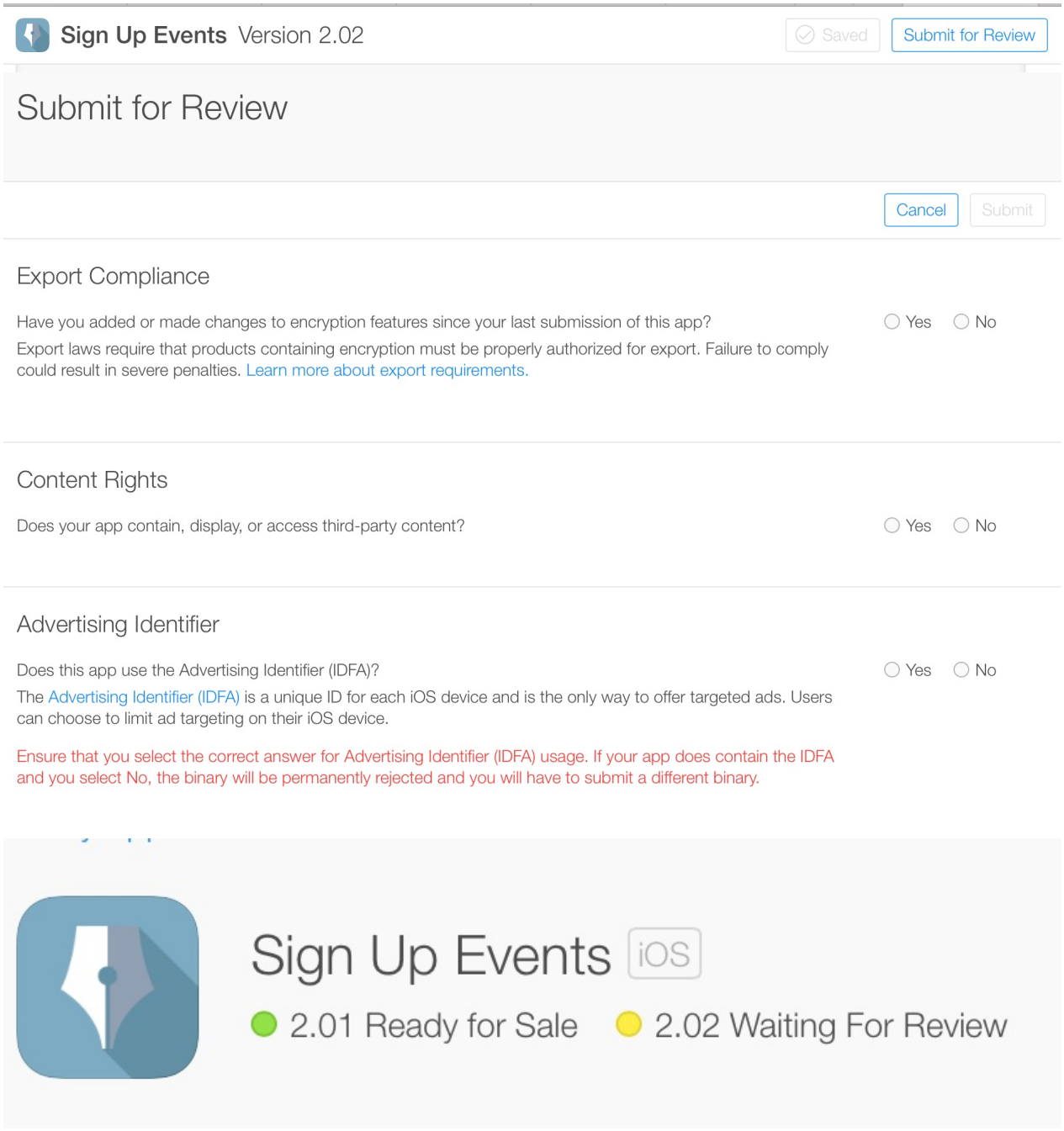
Attach your build to your app by selecting the "+" sign. There may be a delay of up to 10 minutes before your uploaded build will be available.





Step 9

Hit Save and Submit. Apple requires the you to answer three questions regarding export compliance, content rights and advertising identifier. Each provides a description and should correspond to your particular app. In this example, we answered "No" to each.



Reference

Submitting Your App to the Store
https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html

Review Process

Apple reviews each build that they receive. This is a measure of quality control that needs to be considered carefully when submitting new apps and updates to existing apps.

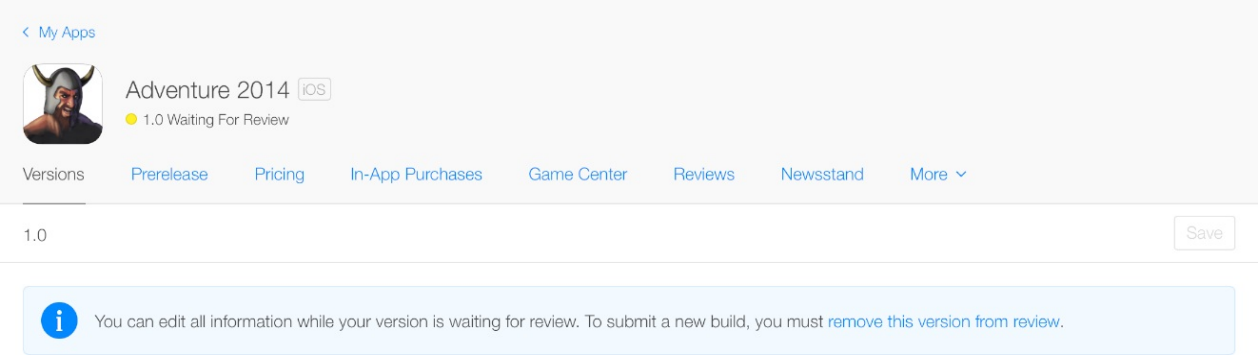
Waiting for Review

Apple will not specify a specific duration, but the general timeframe for an app review will be 10-14 days for a new app and 5-7 days for a review.

http://appreviewtimes.com

Removing your build from the review queue

There are times when a developer may wish to remove their binary from the the review queue. This could happen if the binary you submitted has a critical bug which is discovered after submission. Click on the version of the app within ITC. Then you'll see a message in the app's metadata, which has a link to remove your build from the queue (see reference 2).



Expedited Review

From the Apple documentation (see reference 1):

If you face extenuating circumstances, you can request the review of your app to be expedited. These circumstances include fixing a critical bug in your app on the App Store or releasing your app to coincide with an event you are directly associated with.

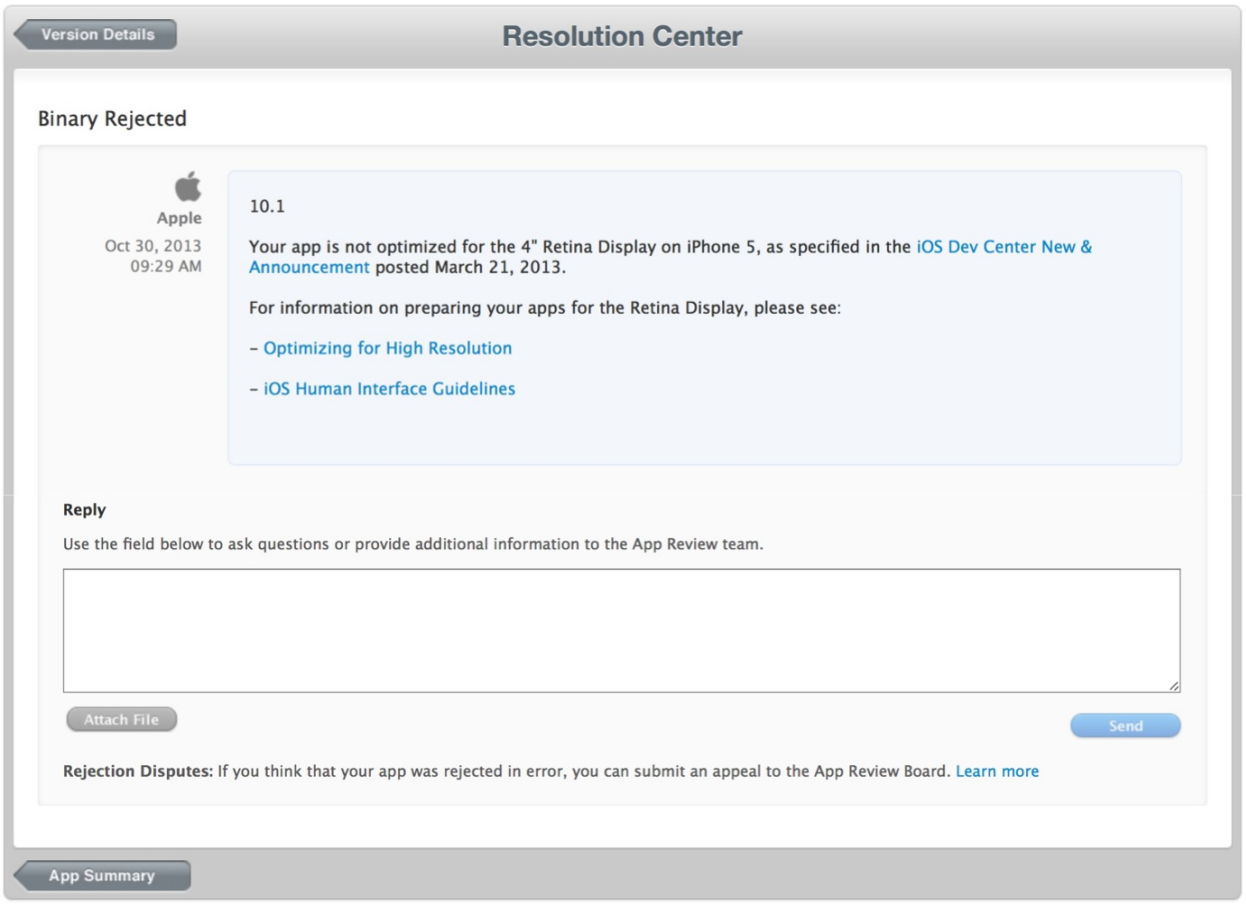
Urgent Bug Fix If you've submitted an update to fix a critical bug in your app on the App Store and you are requesting an expedited review, be sure to include the steps to reproduce the bug on the current version of your app.

Time-Sensitive Event For apps associated with an event, we recommend you plan and schedule the release of your app in iTunes Connect. However, if your app is stil in review and the launch of your event is quickly approaching, you can request to have your app review expedited. When submitting your request, it's important to include the event, date of the event, and your app's association with the event.

Please Note: Expedited reviews are granted on a limited basis and we cannot guarantee that every request will be expedited. Sign in to request an expedited review.

Responding to Rejection

If Apple decides to reject your binary upload, they will provide a very specific reason from the following document: https://developer.apple.com/app-store/review/guidelines/



You can then respond to their feedback within a form that they will send to you. Please remember to be courteous and polite to your reviewer. Once responded, you can upload a new binary, if necessary. Sometimes, Apple will reject for missing metadata. If this is the case, you can edit the metadata and submit your existing build for review, without uploading a new binary. Apple looks at every build and every comment.

References

(1) Apple's App Review Documentation https://developer.apple.com/support/appstore/app-review/

(2) Viewing and Changing Your App's Status and Availability
https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/Chan

gingAppStatus.html

Conclusion

You've done it. A big congratulations to you. Whether this was your first language or fifth, getting to this point is challenging.

From all of us at General Assembly, thank you for taking the time and making this course possible. We look forward to seeing your next app on the iTunes App Store.