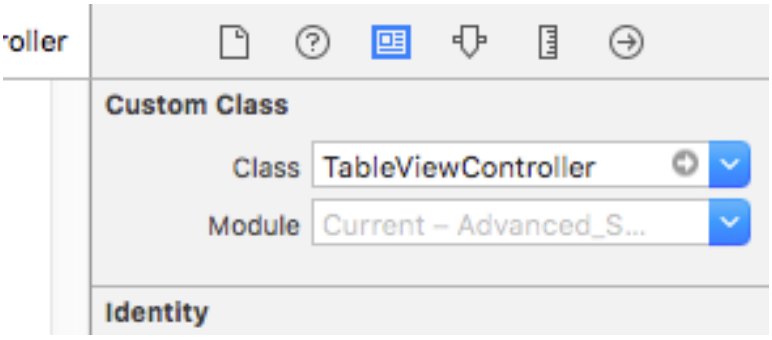
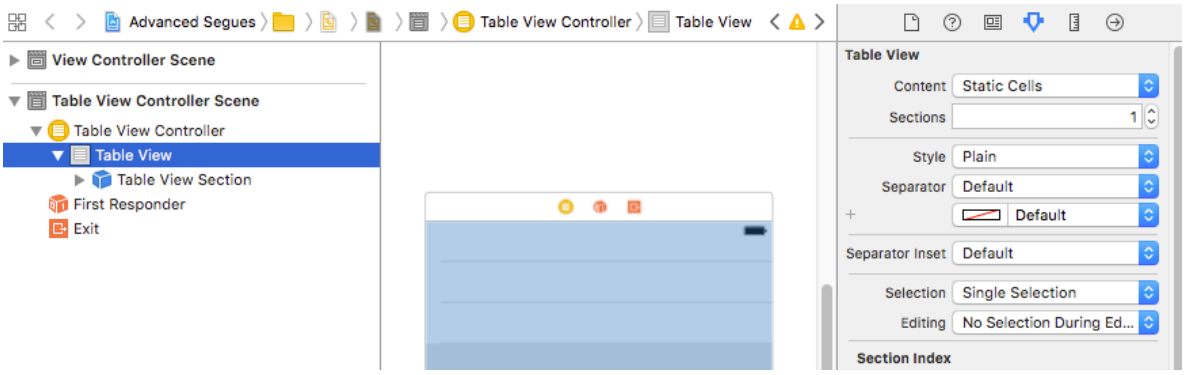
Advanced segues

1. We want to create a segue to a table with numbered cells, click on a cell and move back to the initial view reporting the index of the cell that was tapped
2. add a TableViewController (TVC) in the storyboard
3. create a basic segue by adding add a button to the initial VC, then Ctrl + Drag the button onto the TVC, select show
4. create a new Cocoa Touch Class file, TableViewController.swift, as a subclass of UITableViewController
5. From the Identity inspector of the TVC select the file just created as the controlling custom class



6. In the Document Outline pane select Table View, from the Attribute inspector select Static Cells content, 1 section



7. In the Document Outline pane select the Tabe View Section, from the Attribute inspector select 4 Rows

Converting Old Apps To Use A Storyboard

If you have an existing app which uses individual .xib files for each screen do this to convert it to use a storyboard.

Check your has projectnameAppDelegate.h and .m (if not see the apple guide above to create one)

Update main.m from

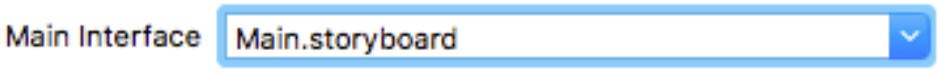
```
#import <UIKit/UIKit.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

to

```
#import <UIKit/UIKit.h>
#import "projectnameAppDelegate.h"
int main(int argc, char *argv[]) {
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil,
NSStringFromClass([projectnameAppDelegate class]));
    }
}
```

Add A Storyboard

Right click the project folder, New File > iOS > User Interface > Storyboard
Name it "Main.storyboard"
Now add your first View Controller to the storyboard from the Object library.
If the first view controller is embedded in a container such as a navigation controller or tab bar controller see the apple guide above.
Select the project in the left view and then select it in the Targets section shown. In 'Main Interface' select your MainStoryboard.



Click the top bar of the first scene in the storyboard created and then in the attributes panel select the 'Is Initial View Controller' checkbox.

Creating a static library CSP in Xcode

We want create a static library in iOS for the CSP (Cosmed Streaming Protocol)
Start a library project in Xcode called CSP.

link the binary in build phases to the **libLibname.a**
put the **Libname.h** header file in the project folder
set the project to always search the user paths and added **$(BUILT_PRODUCTS_DIR)** recursive to the user header search paths.

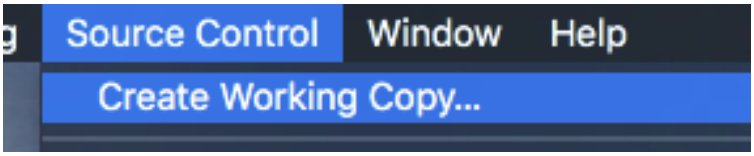
Create a Appname swift project
Add new file in the project folder named **Appname-Bridging-Header.h**. Inside that file is where the #import "Libname.h" line belonged
I open the app build settings and under **Objective-C Bridging Header** I add the path **Appname/Appname-Bridging-Header**. I also added the **Header Search Path $(BUILT_PRODUCTS_DIR)**
After I did all this I am able to use the Libname classes in my ViewController.swift file.

Compiler flags

Activate the appropriate warning flags to the compiler. For instance
-Wall -Wextra -Weverything

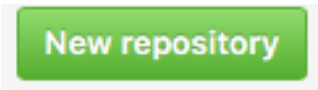
Add the project to a GitHub git repo

First of all, create a local git repo on your mac.
Go to **Source Control > Create Working Copy...**

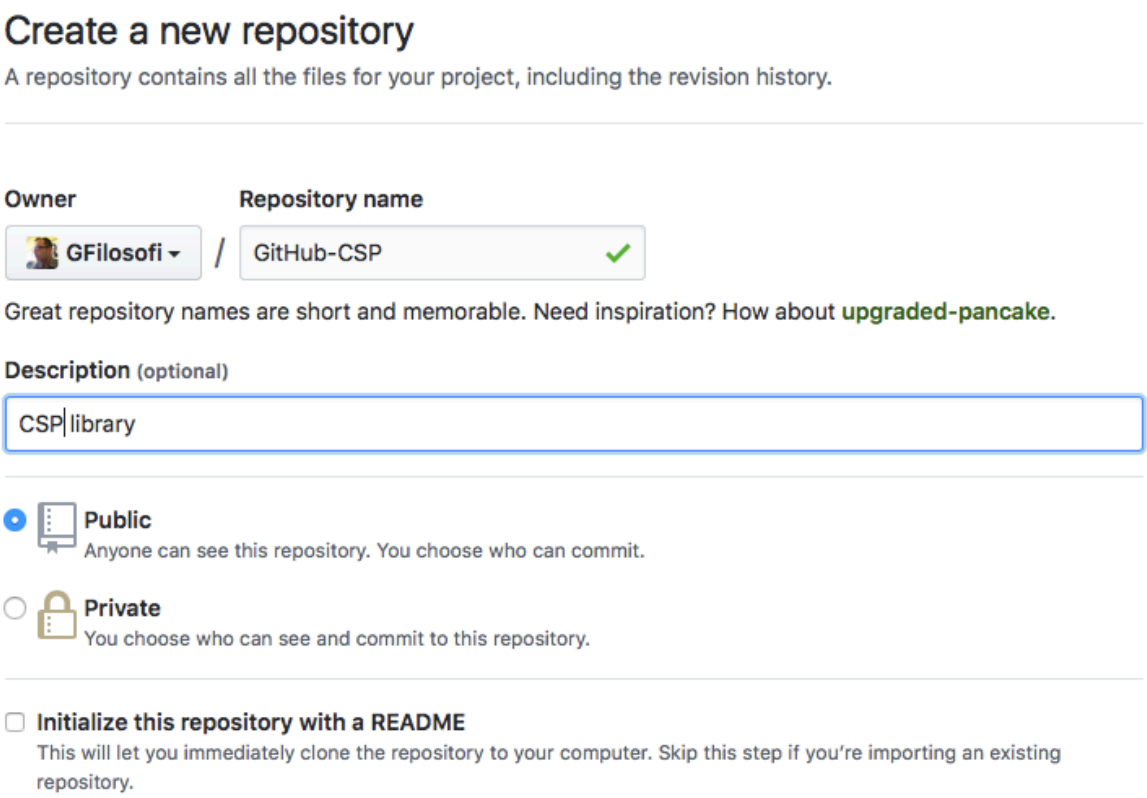


Select the project and hit **Create**, Xcode will automatically perform an initial commit for you.

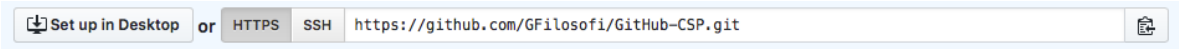
To add a remote repo,
Sign in to GitHub site
Click on New Repository button next to your account name



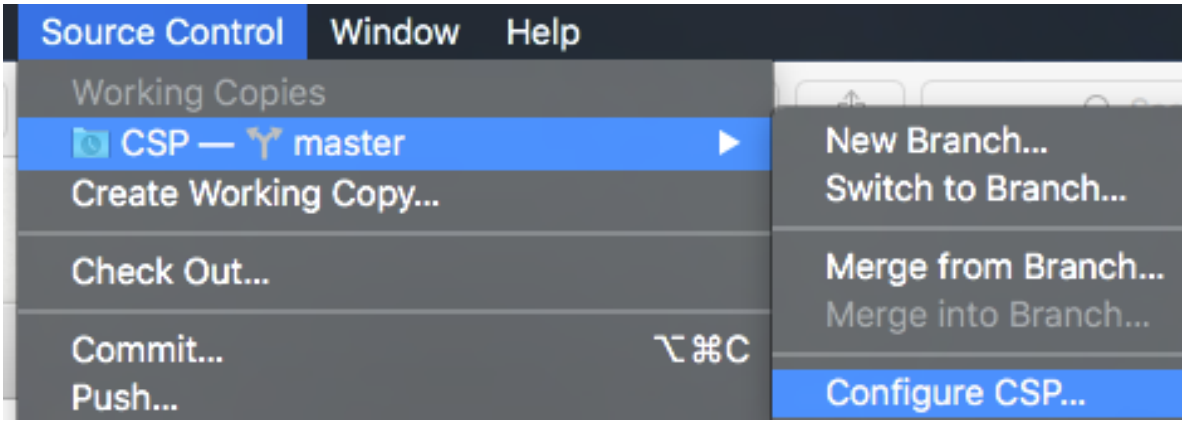
Type the repository name and description (use a name different from Xcode project name).
Do not initialize your repository with a README.



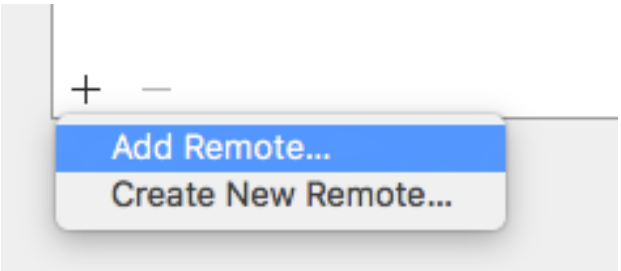
You get this



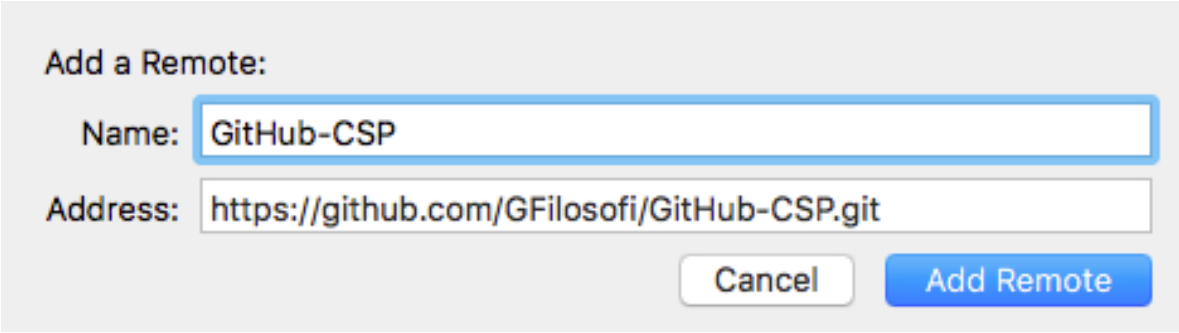
Copy the repo https/ssh link and go back in Xcode and select **Source Control > Configure > Remotes**



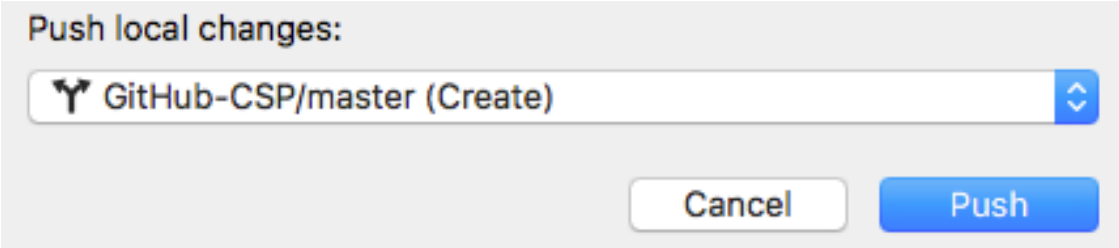
Click on the "+" sign in the bottom left corner and select Add Remote



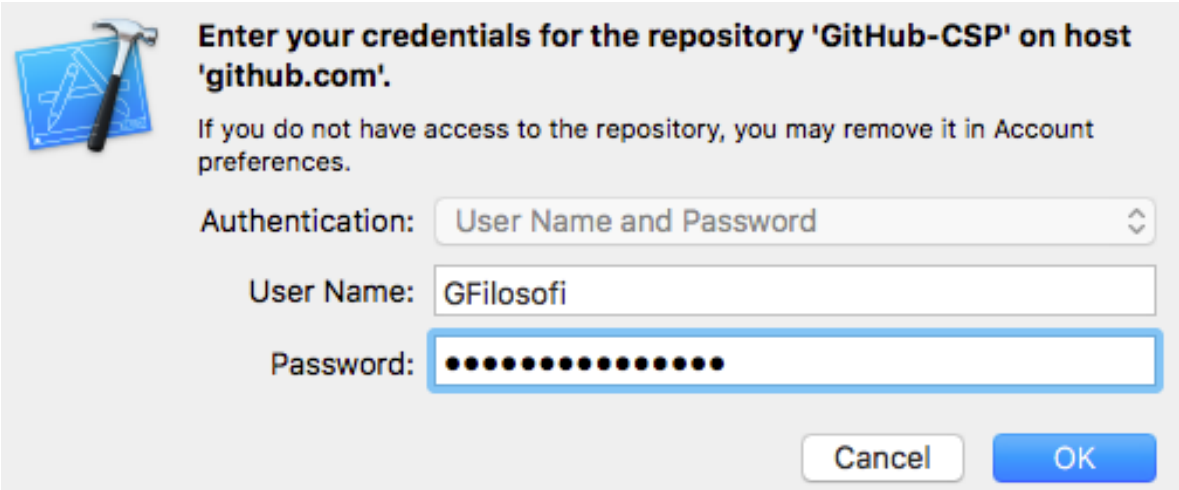
In the Address text field paste the repo link
In the Name field copy the same name you used in GitHub



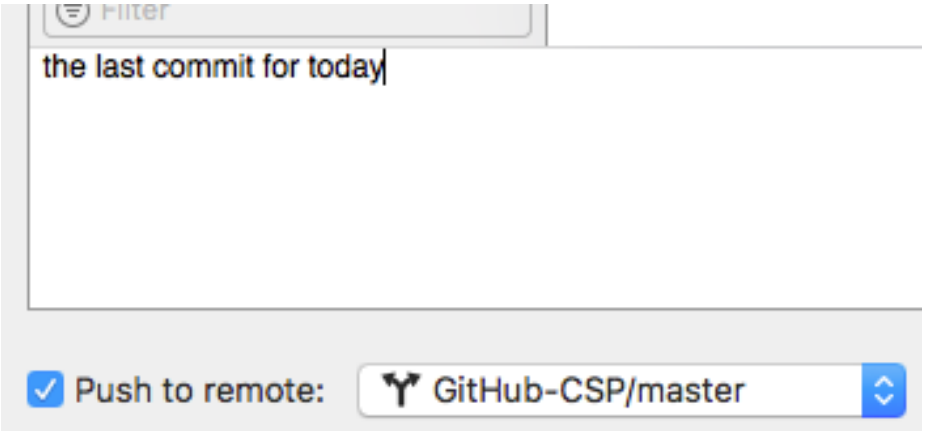
Click Add Remote. Click Done.
From **Source Control > Push** push the first (repo creation) commit



You will be asked to enter your GitHub credentials

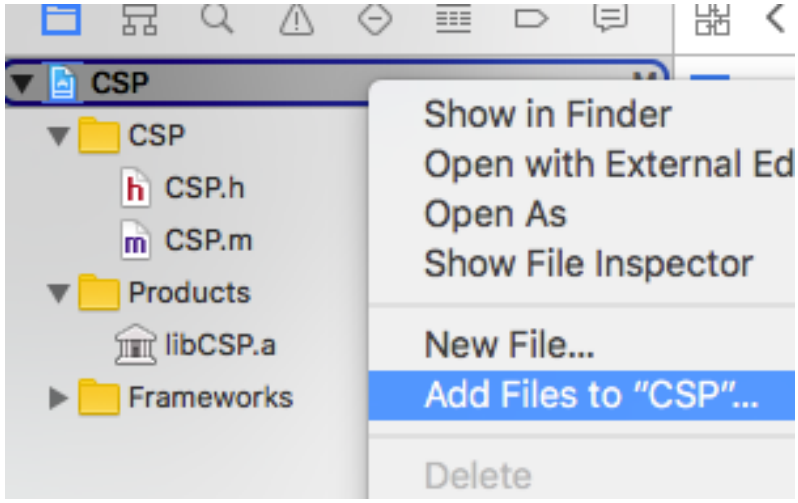


To push subsequent commits make sure to have checked the Push to remote box

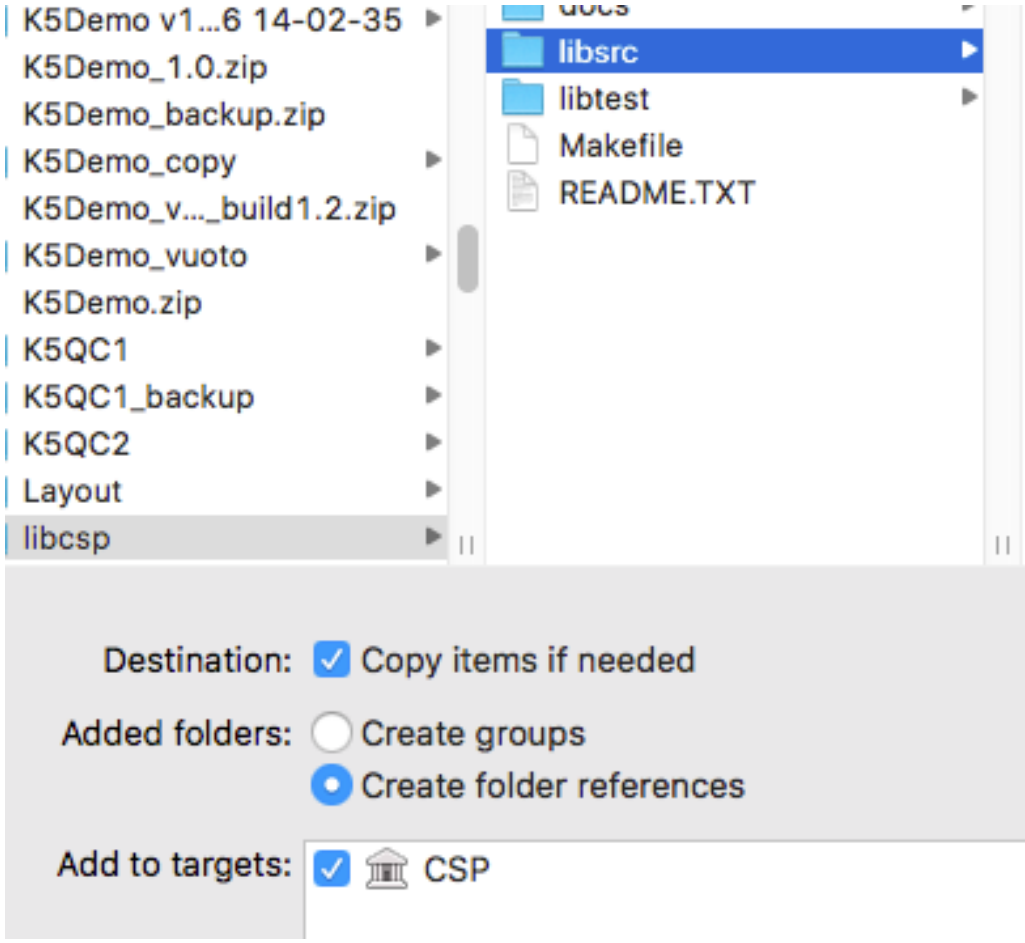


Importing external files into the Xcode project

Now we may need to import C sources already developed

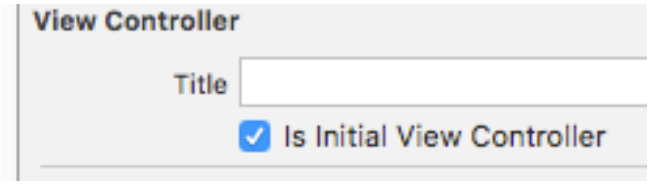


In this case we are going to import a folder called libsrc.
We have to select a couple of options

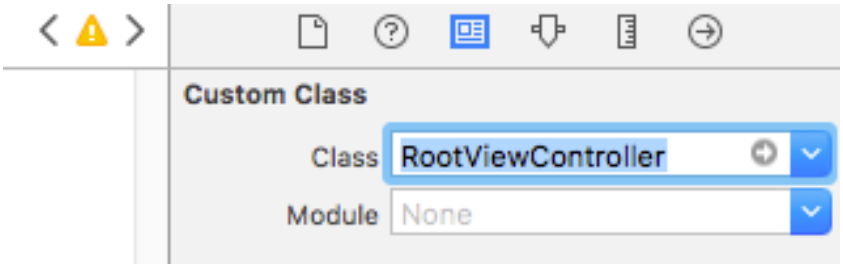


- Copy items if needed: this will create a copy of the files in the Xcode project. This is important also to keep track of them under Source Control (git)
- Create groups: this option collects files in yellow folders. This will help

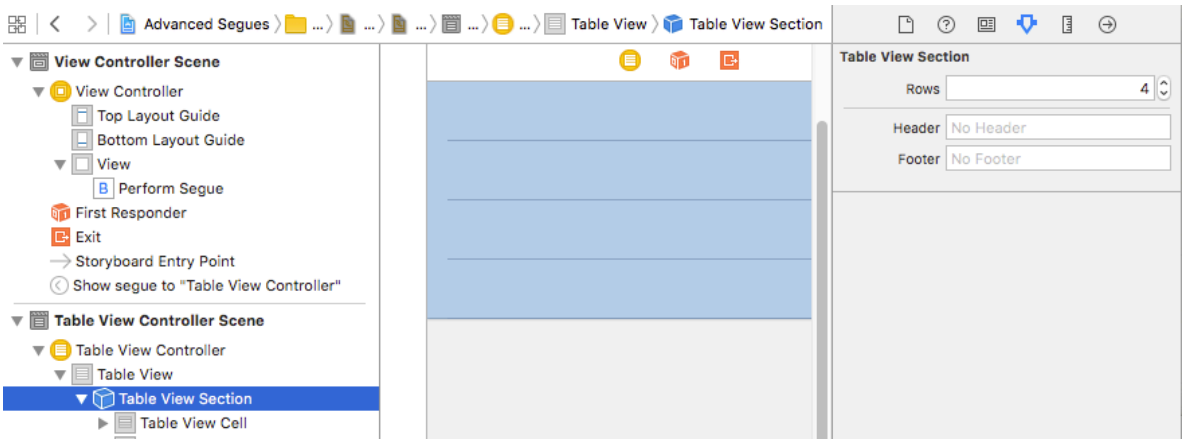
segregation in the Xcode project
- Create folder references: blue folders also map to real folders in the deployed bundle of the app. Use folder references if your app needs to separate assets too (eg if there are resources with the same name but different file paths)



In the storyboard select the scenes top bar and in the identity inspector, change the class to the name of your ViewController files.



Now you can link all of the view objects to the View Controller in the top bar as normal.



8. In the Document Outline pane, for each Table View Cell, create a segue to the initial VC by Ctrl + Drag
9. In the TableViewController.swift override the method willSelectRowAtIndexPath and there set the global variable rowCounter equal to the indexPath. Note that we don't use the didSelect method because it could be executed too late to make the assignment
10. In the TableViewController.swift update to 1 the return value of the numberOfSectionInTableView method
11. In the TableViewController.swift update to 4 the return value of the numberOfRowsInSectionIn method

Design Patterns

Singleton

The singleton pattern is when only one instance of the singleton class can be created or is alive at any one time. That unique instance is called singleton.

All iOS or OS X frameworks make use of the singleton pattern. An iOS application, for example, can only have one instance of the UIApplication class, which you can access through the sharedApplication class method, as follows

In Objective-C:
```
UIApplication *sharedApp = [UIApplication sharedApplication];
```
In Swift 2.1:
```
let sharedApp = UIApplication.sharedApplication()
```
In Swift 3:
```
let appDelegate = UIApplication.shared
```

Even though the UIApplication class gives you access to the UIApplication singleton, nothing prevents you from explicitly instantiating a UIApplication instance

In Objective-C:
```
    UIApplication *newApplication = [[UIApplication alloc] init];
```
In Swift 3:
```
let appDelegate = UIApplication()
```

The result, however, is a runtime exception.

```
2016-09-23 16:38:30.782 CoreDataDemo[4693:79294] *** Terminating app due to uncaught
exception 'NSInternalInconsistencyException', reason: 'There can only be one UIApplication
instance.'
```

The UIApplication class was designed with the singleton pattern in mind.

MVC

ToDo

MVVM

This pattern is very common in app development. One reason is to keep decoupled the application and the presentation layers, especially for testing purposes.

Downloading images from the web and save them, in Xcode

NSURLSession is the key object responsible for sending and receiving HTTP requests.
NSURLSessionTask is an abstract class that denotes a task object. A session creates a task, which does the actual work of fetching data and downloading or uploading files.
The NSURLSession has three main principal methods:

- dataTask - is for normal data requests (when for example you'll have to ask to a web server for some data
- downloadTask - is for downloading files. It can resume the downloads for you and, as soon as the data is ready you'll have the temporary path in which the data is stored
- uploadTask - is for uploading data

1. we want download an image from the internet.
2. open the web page containing the image you wish to download, https://it.wikipedia.org/wiki/File:Johann_Sebastian_Bach.jpg
3. get the URL of the image, https://upload.wikimedia.org/wikipedia/commons/6/6a/Johann_Sebastian_Bach.jpg
4. implement as follows

```
class ViewController: UIViewController {

    @IBOutlet var image: UIImageView!

    override func viewDidLoad() {
        super.viewDidLoad()

        let url = URL(string: "https://upload.wikimedia.org/wikipedia/commons/6/6a/
            Johann_Sebastian_Bach.jpg")
        let request = URLRequest(url: url!)
        let task = URLSession.shared.dataTask(with: request, completionHandler: {(data, response, error) ->
            Void in
            //print("This is the server response: \(response!)")
            if error != nil {
                print("some error!")
            } else {
                if let bach = UIImage(data: data!) {
                    self.image.image = bach
                }
            }
        })
        task.resume()
    }
```

5. the URLSession task is normally executed in background mode. It is not safe to update a UI object from within a thread running in background mode, because some contention may occur with other concurrent threads which may do the same. The solution is to submit the execution of the critical code in a queue managed by the system. When an app launches, the system automatically creates a special queue called the *main queue*. Work items enqueued to the main queue execute serially on the app's main thread (outside of the background context). You can access the main

queue using the *main* type property. In conclusion, we should use the DispatchQueue.main.async dispatcher

```
20          let task = URLSession.shared.dataTask(with: request, completionHandler: {(data, response,
                error) -> Void in
21              //print("This is the server response: \(response!)")
22              if error != nil {
23                  print("some error occurred!")
24              } else {
25                  DispatchQueue.main.async(execute: {() -> Void in
26                      if let bach = UIImage(data: data!) {
27                          self.image.image = bach
28                      }
29                  })
30              }
31          })
32          task.resume()
```

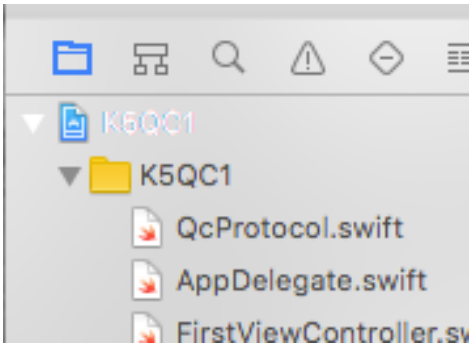
Save the image for offline usage

1. The code written until now works only when a network connection is available
2. We are going to save the downloaded image in other to use it also when offline
3. The NSSearchPathForDirectoriesInDomains can provide an array of objects that could be casted to file path strings, the first element of which is

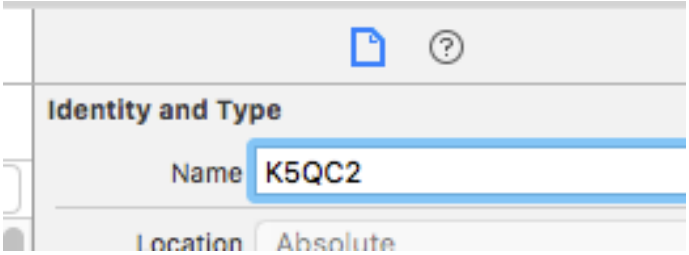
How to rename a Xcode project

We want to rename a K5QC1 project in K5QC2

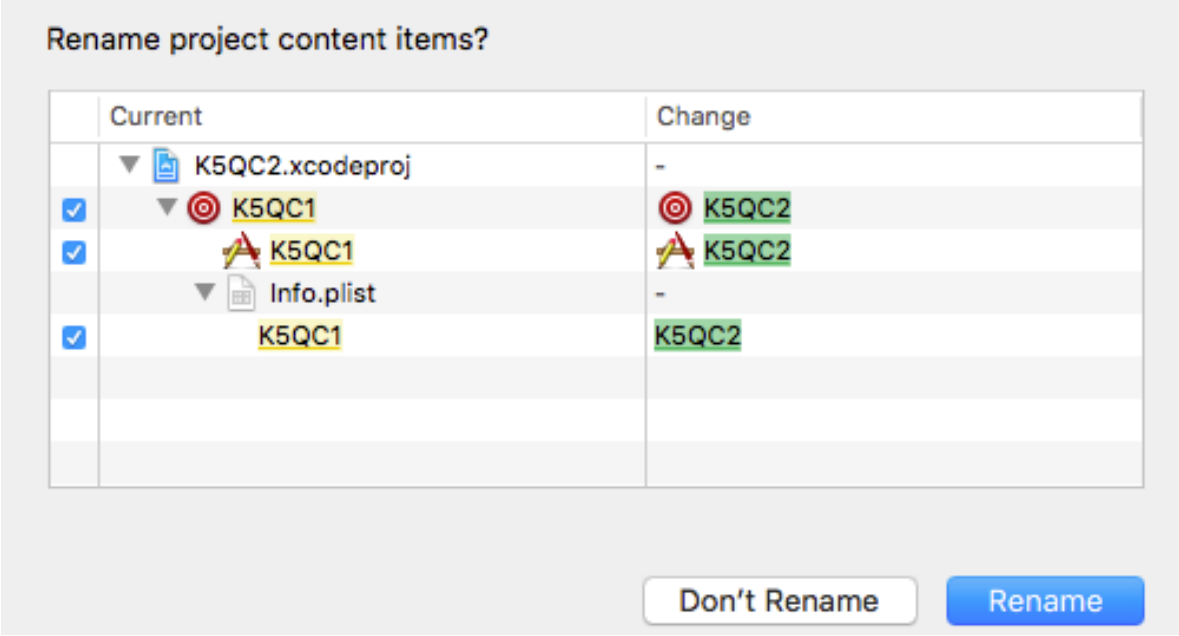
Open K5QC1 project
Select the top folder in the Navigator pane



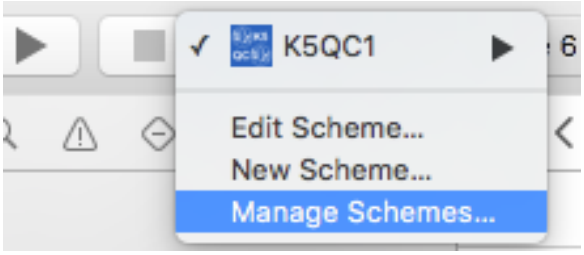
Form the File Inspector pane rename the Identity and Type Name



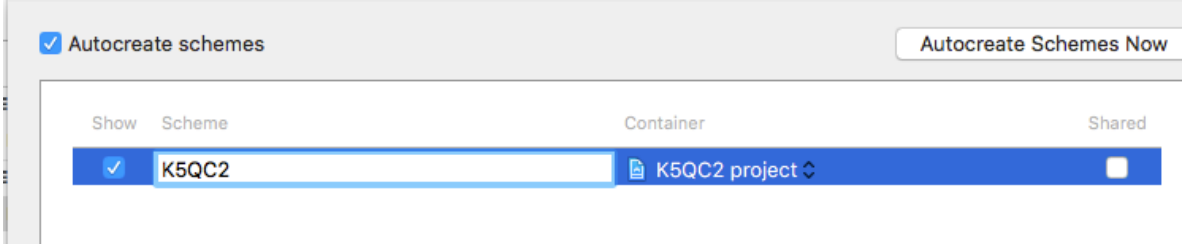
Confirm rename



Open the Manage Scheme panel

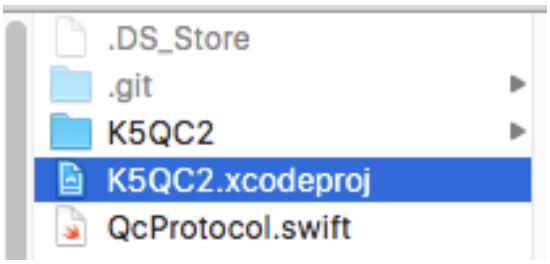


and rename the scheme

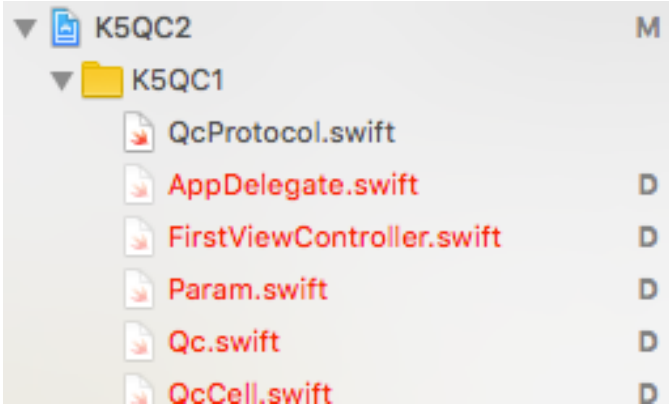


Close xcode
Find the project folder and rename the source folder

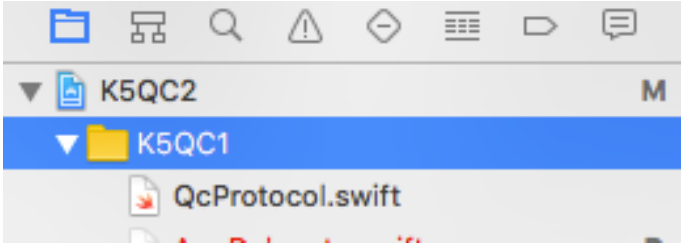
Double click on project file



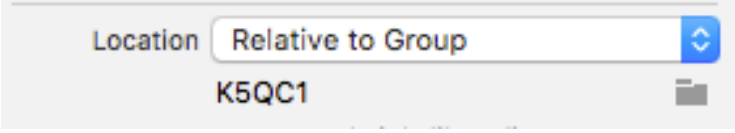
and now we got a bunch of red files



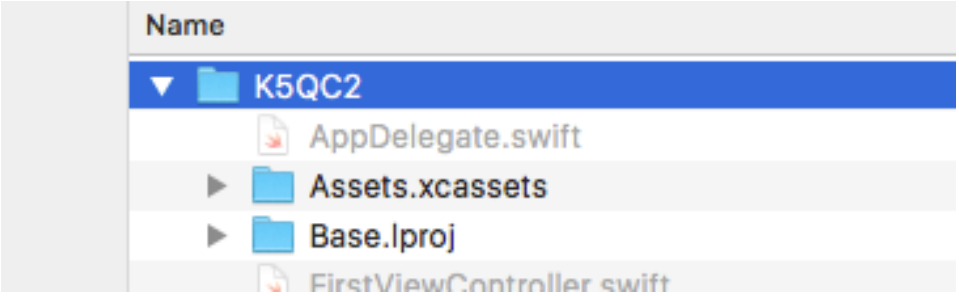
that is because xcode is pointing to the folder we have just renamed.
Then select the folder in the Navigator pane



click on the folder symbol next to the Location section of the File Inspector pane

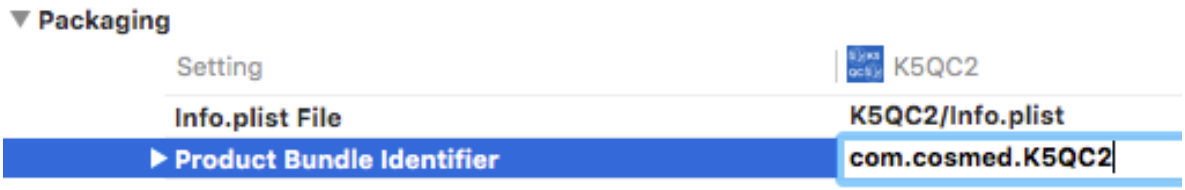


select and choose the correct folder you want



Open the Build Setting pane and rename the plist path

Then rename also the Product Bundle Identifier



Build and Run the project!

Jailbreak

- Il jailbreak mette in evidenza vulnerabilità dei device Apple, ma non é fortemente combattuto da Apple in quanto per essere attuato richiede di essere proprietari del device stesso, infatti bisogna collegarlo al mac, fare molteplici riavvii, ecc
- Il gruppo evaders ha creato evasiOn, il jailbreak di iOS 6
- Sfrutta exploit sia in kernel che in user-land
- Un servizio, o daemon, è un processo esterno che gira in background e senza una UI
- Il primo step di evasiOn consiste in un accesso al filesystem del device basandosi sul servizio Mobile Backup in esecuzione sul device connesso in USB al mac.
- Il secondo step é l'ottenimento dei privilegi dell'utente root
- Launchd é un servizio che gestisce il run di molti altri servizi
- Viene rimontato il fs permettendo lettura e scrittura a tutti gli utenti
- Questo é un jailbreak untethered cioè che sopravvive allo spegnimento del device
- IOKit è il framework che gestisce il collegamento con hw esterno. Vi è una vulnerabilità che viene sfruttata per patchare il kernel in maniera da evitare il controllo della firma delle applicazioni

LLDB commands

LLVM: Apple compiler

Debugger commands:

```
apropos         -- List debugger commands related to a word or subject.
breakpoint      -- Commands for operating on breakpoints (see 'help b' for
                shorthand.)
bugreport       -- Commands for creating domain-specific bug reports.
command         -- Commands for managing custom LLDB commands.
disassemble     -- Disassemble specified instructions in the current
                target.  Defaults to the current function for the
                current thread and stack frame.
expression      -- Evaluate an expression on the current thread.  Displays
                any returned value with LLDB's default formatting.
frame           -- Commands for selecting and examing the current thread's
                stack frames.
gdb-remote      -- Connect to a process via remote GDB server.  If no host
                is specifed, localhost is assumed.
gui             -- Switch into the curses based GUI mode.
help            -- Show a list of all debugger commands, or give details
                about a specific command.
kdp-remote      -- Connect to a process via remote KDP server.  If no UDP
                port is specified, port 41139 is assumed.
language        -- Commands specific to a source language.
log             -- Commands controlling LLDB internal logging.
memory          -- Commands for operating on memory in the current target
                process.
platform        -- Commands to manage and create platforms.
plugin          -- Commands for managing LLDB plugins.
process         -- Commands for interacting with processes on the current
                platform.
quit            -- Quit the LLDB debugger.
register        -- Commands to access registers for the current thread and
                stack frame.
script          -- Invoke the script interpreter with provided code and
                display any results.  Start the interactive interpreter
                if no code is supplied.
settings        -- Commands for managing LLDB settings.
source          -- Commands for examining source code described by debug
                information for the current target process.
target          -- Commands for operating on debugger targets.
thread          -- Commands for operating on one or more threads in the
```

```
                 current process.
  type           -- Commands for operating on the type system.
  version          -- Show the LLDB debugger version.
  watchpoint       -- Commands for operating on watchpoints.

Current command abbreviations (type 'help command alias' for more info):

  add-dsym  -- Add a debug symbol file to one of the target's current modules
            by specifying a path to a debug symbols file, or using the
            options to specify a module to download symbols for.
  attach    -- Attach to process by ID or name.
  b         -- Set a breakpoint using one of several shorthand formats.
  bt        -- Show the current thread's call stack.  Any numeric argument
            displays at most that many frames.  The argument 'all' displays
            all threads.
  c         -- Continue execution of all threads in the current process.
  call      -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
  continue  -- Continue execution of all threads in the current process.
  detach    -- Detach from the current target process.
  di        -- Disassemble specified instructions in the current target.
            Defaults to the current function for the current thread and
            stack frame.
  dis       -- Disassemble specified instructions in the current target.
            Defaults to the current function for the current thread and
            stack frame.
  display   -- Evaluate an expression at every stop (see 'help target
            stop-hook'.)
  down      -- Select a newer stack frame.  Defaults to moving one frame, a
            numeric argument can specify an arbitrary number.
  env       -- Shorthand for viewing and setting environment variables.
  exit      -- Quit the LLDB debugger.
  f         -- Select the current stack frame by index from within the current
            thread (see 'thread backtrace'.)
  file      -- Create a target using the argument as the main executable.
  finish    -- Finish executing the current stack frame and stop after
            returning.  Defaults to current thread unless specified.
  image     -- Commands for accessing information for one or more target
            modules.
  j         -- Set the program counter to a new address.
  jump      -- Set the program counter to a new address.
  kill      -- Terminate the current target process.
  l         -- List relevant source code using one of several shorthand
            formats.
  list      -- List relevant source code using one of several shorthand
            formats.
```

```
n        -- Source level single step, stepping over calls.  Defaults to
            current thread unless specified.
next      -- Source level single step, stepping over calls.  Defaults to
            current thread unless specified.
nexti     -- Instruction level single step, stepping over calls.  Defaults to
            current thread unless specified.
ni       -- Instruction level single step, stepping over calls.  Defaults to
            current thread unless specified.
p        -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
parray    -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
po       -- Evaluate an expression on the current thread.  Displays any
            returned value with formatting controlled by the type's author.
poarray   -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
print     -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
q        -- Quit the LLDB debugger.
r        -- Launch the executable in the debugger.
rbreak    -- Sets a breakpoint or set of breakpoints in the executable.
repl      -- Evaluate an expression on the current thread.  Displays any
            returned value with LLDB's default formatting.
run       -- Launch the executable in the debugger.
s        -- Source level single step, stepping into calls.  Defaults to
            current thread unless specified.
si       -- Instruction level single step, stepping into calls.  Defaults to
            current thread unless specified.
sif      -- Step through the current block, stopping if you step directly
            into a function whose name matches the TargetFunctionName.
step      -- Source level single step, stepping into calls.  Defaults to
            current thread unless specified.
stepi     -- Instruction level single step, stepping into calls.  Defaults to
            current thread unless specified.
t        -- Change the currently selected thread.
tbreak    -- Set a one-shot breakpoint using one of several shorthand
            formats.
undisplay -- Stop displaying expression at every stop (specified by stop-hook
            index.)
up       -- Select an older stack frame.  Defaults to moving one frame, a
            numeric argument can specify an arbitrary number.
x        -- Read from the memory of the current target process.

For more information on any command, type 'help <command-name>'.
(lldb)
```

(lldb) version
lldb-370.0.42
  Swift-3.1

(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  * frame #0: 0x000000010737610b MyContainer`ViewController.viewDidLoad(self=0x00007fbaaac0a7d0) -> () at ViewController.swift:29
    frame #1: 0x0000000107376312 MyContainer`@objc ViewController.viewDidLoad() -> () at ViewController.swift:0
    frame #2: 0x0000000107fc7cca UIKit`-[UIViewController loadViewIfRequired] + 1235
    frame #3: 0x0000000107fc810a UIKit`-[UIViewController view] + 27
    frame #4: 0x0000000107e9063a UIKit`-[UIWindow addRootViewControllerViewIfPossible] + 65
    frame #5: 0x0000000107e90d20 UIKit`-[UIWindow _setHidden:forced:] + 294
    frame #6: 0x0000000107ea3b6e UIKit`-[UIWindow makeKeyAndVisible] + 42
    frame #7: 0x0000000107e1d31f UIKit`-[UIApplication _callInitializationDelegatesForMainScene:transitionContext:] + 4346
    frame #8: 0x0000000107e23584 UIKit`-[UIApplication _runWithMainScene:transitionContext:completion:] + 1709
    frame #9: 0x0000000107e20793 UIKit`-[UIApplication workspaceDidEndTransaction:] + 182
    frame #10: 0x000000010bcd15f6 FrontBoardServices`__FBSSERIALQUEUE_IS_CALLING_OUT_TO_A_BLOCK__ + 24
    frame #11: 0x000000010bcd146d FrontBoardServices`-[FBSSerialQueue _performNext] + 186
    frame #12: 0x000000010bcd17f6 FrontBoardServices`-[FBSSerialQueue _performNextFromRunLoopSource] + 45
    frame #13: 0x000000010a567c01 CoreFoundation`__CFRUNLOOP_IS_CALLING_OUT_TO_A_SOURCE0_PERFORM_FUNCTION__ + 17
    frame #14: 0x000000010a54d0cf CoreFoundation`__CFRunLoopDoSources0 + 527
    frame #15: 0x000000010a54c5ff CoreFoundation`__CFRunLoopRun + 911
    frame #16: 0x000000010a54c016 CoreFoundation`CFRunLoopRunSpecific + 406
    frame #17: 0x0000000107e1f02f UIKit`-[UIApplication _run] + 468
    frame #18: 0x0000000107e250d4 UIKit`UIApplicationMain + 159
    frame #19: 0x0000000107378167 MyContainer`main at AppDelegate.swift:12
    frame #20: 0x000000010b56165d libdyld.dylib`start + 1

MacOS and iOS internals

- Un file appname.ipa é un file zip. Cambiando infatti il nome in appname.zip lo si può decomprimere e vederne il contenuto, la cartella Payload e, all’interno la appname.app, il bundle dell'app
- A Mac is also a Personal Computer, but a Mac is not a PC
- Hackintosh: projects aimed to port OS X on a PC
- History:
  - 1975: SW creates Apple I (6502)
  - 1976: Apple is founded
  - 1977: Apple II (6502)
  - 1979-1983: the Lisa project. (68000)
  - 1981-1984: the Macintosh project, for the Person In The Street (PITS)
  - 1984-2000: Mac OS Classic 1,..,9. This is a cooperative (non-preemptive) multitasking environment
  - 1987: SJ founds NeXT
  - 1988: SJ presents NeXT cube
  - 1990: Microsoft Windows 3.x is released
  - 1991: Apple, IBM and Motorola join to create PowerPC architecture. The first implementation was PPC 601 in 1993
  - 1997: Apple acquires NeXT for $400M
  - 1997: NeXTSTEP, based on Mach microkernel and Objective-C language
  - 1997: Rapsody (never released to public)
  - from 2001: Mac OS X 10.x.y, based on Darwin x+4.y. This is a preemptive multitasking environment.
  - 2004: Mac OS X becomes intel x86 compatible (universal binary)
  - 2005: SJ announced migration to x86
  - 2006: the first iMac and MacBook 1Pro
  - 2007: the first iOS and iPhone
  - 2008: App Store

- Darwin is a core component for macOS, iOS, tvOS and watchOS
- My current Mac’s Darwin version: 16.5.0
- Darwin = XNU kernel + runtime. It is a open source UNIX project.
- XNU = Mach + BSD + libKern + IOKit
- Mac OS X: Darwin, Cocoa, Carbon, Xcode IB, AppKit, IOKit, NKEs, Core Services, Applic. Services, QuickTime, JDK, Aqua, Quartz
- In 2000 SJ presented this slide, in which he presented Aqua

- Carbon and Cocoa are the most important umbrella frameworks (because they rely on other frameworks)
- Carbon is deprecated, but still important
- Cocoa imports AppKit, Foundation, CoreData
- From a historical point of view, Java and C# were strongly inspired to Objective-C
- Objective-C was inspired to Smalltalk. In Smalltalk any variables is an object, an instance of a base class named Object. An operation consists in sending a message to a receiver object through a selector. If the method specified by the selector is not found in the set of its methods (protocol), the object looks up in its superclass, and so on.
- AppKit and Foundation have class prefix NS, which stands for NeXTSTEP
- Mac bootloader: EFI
- iOS bootloader: iBoot
- Bundle: é sia la struttura interna di una directory che contiene eseguibile e dati (tipo package), sia il formato di una shared obj library che deve essere caricata esplicitamente da un processo (tipo plugin), a differenza di librerie standard che vengono caricate implicitamente. Anche i framework, sia Apple che di terze parti, sono bundles, ma di tipo diverso. Anche i file playground sono bundles
- The bundle's main advantage is that the executable is smaller and certain upgrades may just consist of replacing or adding resource files without the need to rebuild the executable.
- Anche le localizzazioni sono nel bundle della app.
- I bundles dei frameworks hanno subfolders per le varie versioni e un link simbolico "Current" alla versione preferita
- plist files can have one of three formats: XML, binary or JSON
- .xib files are XML plist files describing GUIs
- .nib files are binary plists obtained by .xib via IB (Interface Builder)
- CSS: Cascading Style Sheet
- POSIX: Portable Object System Interface, is a standard API for system calls. It is twofold: system call prototypes (for user source program portability) and numbers (for binary portability across multiple platforms yet same architecture). The latter does not apply to OS X because the object format, Mach-O, is not compatible with ELF
- OS X does not use the convention libname.so.X.Y.Z. OS X uses the convention libname.X.dylib

- Kext: Kernel Extension. It is a module loaded in kernel space. For example a driver is a kext. Since Yosemite any kext must be signed. To sign a kext, a developer has to ask Apple a certificate. A signed kext can be loaded and used by any macs. Yosemite had the chance to bypass this control by adding the bootarg --kext-dev-mode=1. This is not permitted anymore since SIP (System Integrity Protection) is in force
- Apple Watch: it has BLE, BT classic, WiFi, NFC

Mobile app Development

Software development can be done on web, desktop or mobile platforms. Mobile devices generally fall into two categories, phones and tablets, with a few crossover devices in between.

SDLC

The process of software development is called the Software Development Lifecycle (SDLC), which includes

- Inspiration: define and refine the idea in terms of use cases, actors, features and functionalities
- Design: define the UX and a UI with the help of a graphic designer.
- Development: build the app. This phase usually starts very early. Focused on functionalities, usability and performances
- Stabilization: a QA or even a selected group of users test the app. Bugs are fixed. beta version
- Deployment
- Maintenance

SDLC methodologies: Agile, Spiral, Waterfall

The UX can be designed via wireframes or mockups using tools such as Balsamiq, Mockingbird, Visio,

The app should "feel at home" on each platform it is aimed to.
The design guidelines for each platform are:

1. **Apple** - Human Interface Guidelines
2. **Android** – Design Guidelines
3. **Windows Phone** – Design library for Windows Phone

The hardware itself also dictates UX decisions. For example, iOS devices have no physical back button, and therefore introduce the Navigation Controller.

Each platform has it's own design language, so a well-designed application may still look different on each platform.

For good UI design inspiration, check out some of the following sites:
1. pttrns.com – (iOS only)
2. androidpttrns.com - (Android only)
3. lovelyui.com – (iOS, Android, and Windows Phone)

Typically, applications go into Prototype, Alpha, Beta, and Release Candidate stages.

Tools that allow for user feedback

1. **Testflight** – This is an iOS product that allows you to distribute apps for testing as well as receive crash reports
2. **LaunchPad (launchpadapp.com)** – Designed for Android, this service is very similar to TestFlight.
3. **hockeyapp.com** - Provides a testing service for iOS, Android and Windows Phone.

Xamarin.iOS and Objective-C apps are distributed in exactly the same way:

1. **Apple App Store** – it is a globally available online application repository that is built into Mac OS X via iTunes
2. **In-House Deployment** – it is meant for internal distribution of corporate applications that aren't available publicly
3. **Ad-Hoc Deployment** – it is intended primarily for development and testing to a limited number of properly provisioned devices. When you deploy to a device via Xcode or Xamarin Studio, it is known as ad-hoc deployment.

Google Play is Google's official app store, but there are many others. Android takes a very open approach to app distribution. Devices are not locked to a single, approved app store. Instead, anyone is free to create an app store.

Sandbox

iOS apps run in what's known as a Sandbox, an environment that enforces security constraints that restrict what your app can access. For instance, an app can read from and write to its own directory, but if it attempts to write to another app directory, it will be terminated.

Android Fragmentation

Unlike iOS, which has a small set of devices, Google doesn't impose any limits on which devices can run the Android OS. This results in a product environment populated by a myriad of different devices with very different hardware, screen resolutions and ratios, and capabilities. Most people choose the most popular 5 or 6 devices to design and test for, and prioritize those.

Manifest Permissions

Android apps all run under a distinct, isolated identity with limited permissions. Without special permissions, an app cannot send a text message, determine the phone state, or even access the Internet. In order to access these features, apps must specify in their manifest file which permissions they would like. When they're being installed the OS reads those permissions, notifies the user that the app is requesting those permissions, and then allows the user to continue or cancel the installation.

Database

Both iOS and Android include the SQLite database engine that allows for sophisticated data storage that also works cross-platform.

The Cross-platform development

If you need to develop an app for more than one platform, you may decide to use the Black Box approach.
There is one single code base language and multiple output builds, one for each platform. This approach has the disadvantage it leverages on those features which are common to all platforms.

An alternative approach is Xamarine, which builds native apps using C# and .NET, sharing only the logic, but exploiting all the features specific to each platform. Anything you can do with Swift, Objective-C, .. you can do with Xamarine.

If you want to develop an app for a single platform the choice is quite easy, because there is no choice. For example for an iOS platform you are going to install Xcode on your mac, and use apple SDK in Objective-C or, for new brand project swift. For Android targets you install Android Studio on your PC and code in java and you are done.
However, when you come to the problem of deploying the same app over multiple platforms, then you may want to optimise the effort and reuse your code over all of them.
Thus you need to introduce an intermediate level of abstraction. You will need tools, language and APIs which are, so to speak, platform agnostic.
Now the complexity of this operation could span from the straightforward to the impossible, depending on the application.
If the app is to download some chunks of JSON data and perform kind of like of string parsing and display data on the screen, it is quite easy. But if you are going to use the front camera, or other platform specific stuff, you need to put the hands at the lower level.

You might use TIZEN, which is very samsung oriented. But it has a lot of disadvantage because it is a blackbox approach.
The only valuable alternative i see is Xamarin, which is the best available cross platform dev environment today.
You can use Xamarin in Visual Studio with C#, but i strongly recommend to install Xamarin Studio on a Mac.
I've had extensive exchange on that with a couple of smart entrepreneurs in the Silicon Valley, and all of them sponsored this choice.

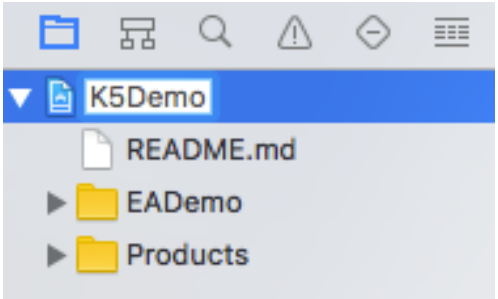
Miscellaneous

- A Universal Windows App (UWP) can run on Windows 10 (desktop) and on Windows 10 Mobile (phone)
- A MVVM is typically used for decoupling application from presentation, like

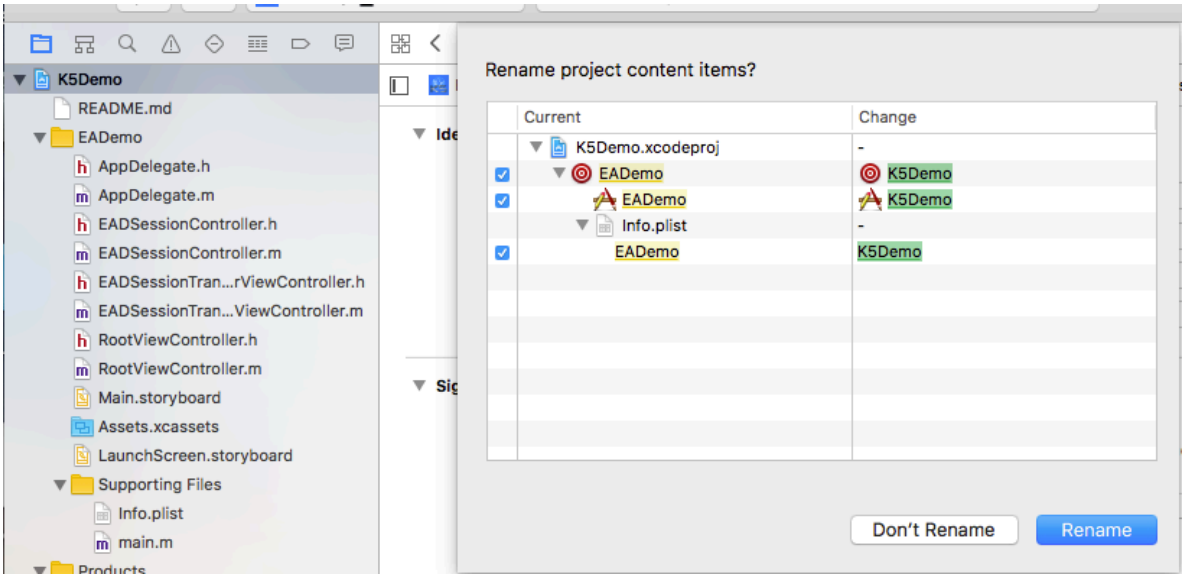
Xamarin. We do this for testability
- Just-in-time compilation (JIT) is a compilation of IL code into native code at run time.
- OpenGL Core is the OpenGL equivalent of DirectX 11

Porting EADemo to K5Demo

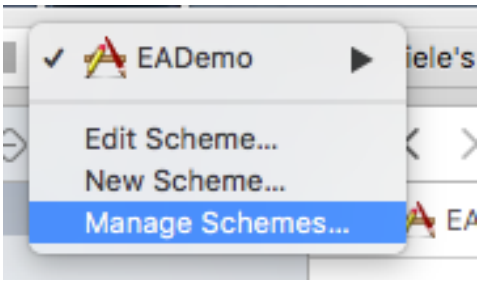
Download the source project EADemo v1.2
In Finder rename the project folder from EADemo to K5Demo
Open the project in Xcode
Rename the project top folder



Confirm renaming all

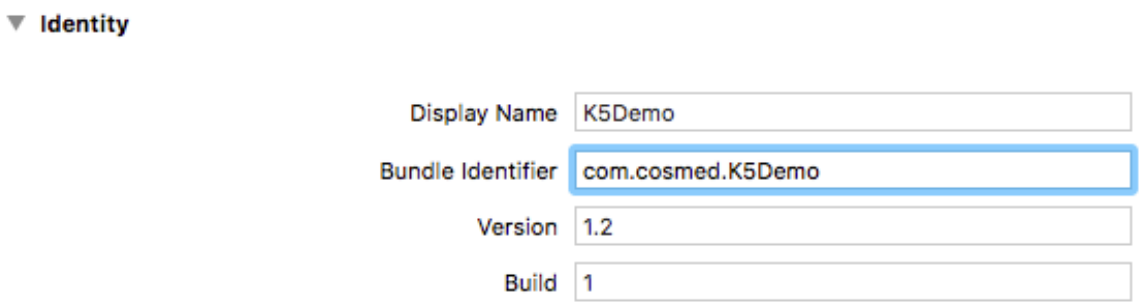


Open the scheme manager

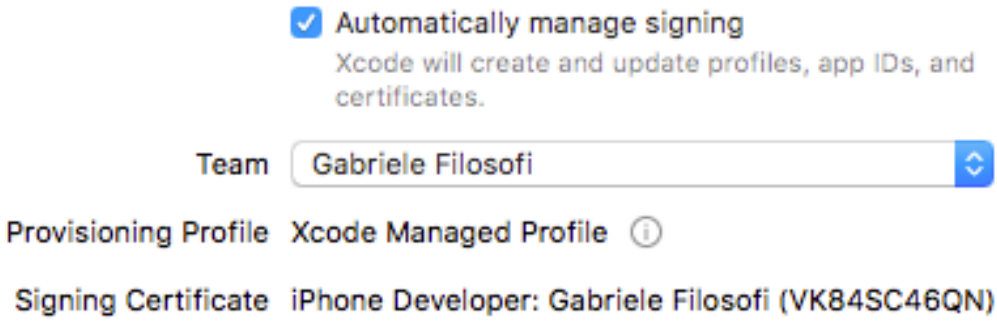


and rename the EADemo scheme as K5Demo.

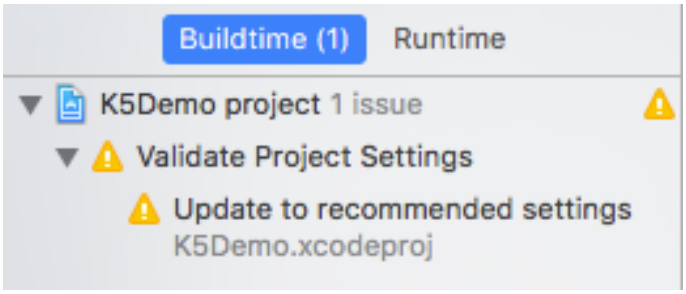
In the Targets > K5Demo > General > Identity section, change the app name and the BI



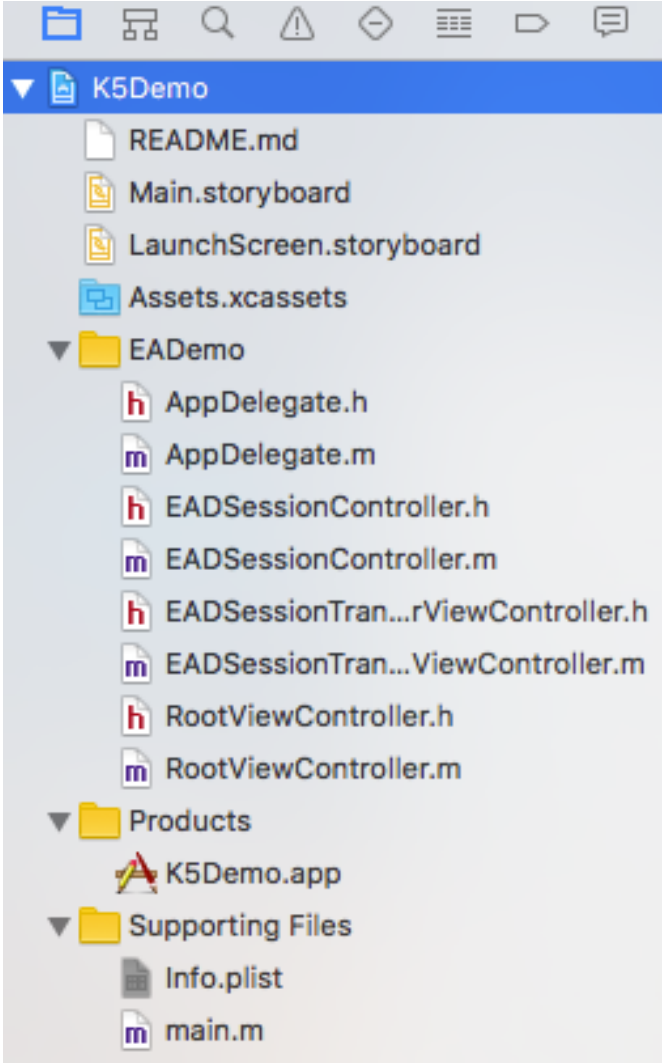
In the Targets > K5Demo > General > Signing section, select the development team. This will enable the signing certificate



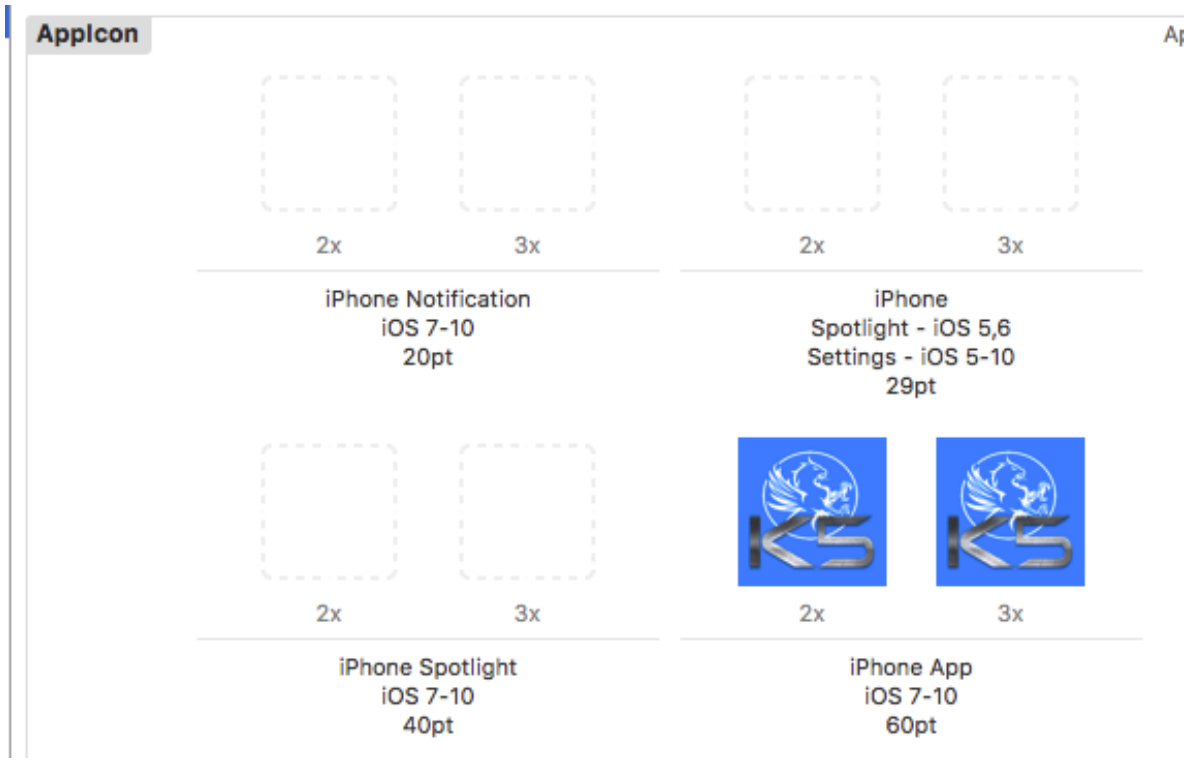
Remove the build warning by updating the project to the recommended settings



Drag the following files/resources outside EADemo folder
*.storyboard
Supporting Files
Asset.xcassets
README.md



In the Assets.scassets > AppIcon add the 2x and 3x images in for the iPhone App



In this case the required size is 60pt, thus the real image resolutions are 120x120 (2x) and 180x180 (3x)

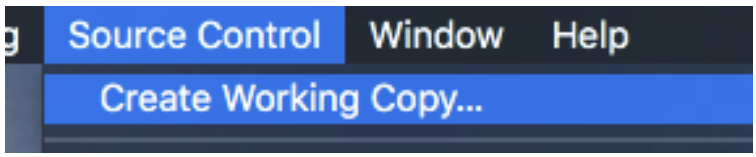
Now we have an accessory which exposes the following IAP2 parameters
PROTOCOL: “com.cosmed.cuml”
APP: “com.cosmed.K5Demo”
We have to make our app matching those two parameters

Open the Info.plist file
Under the Supported external accessory protocol section include com.cosmed.cuml” protocol

Rebuild.

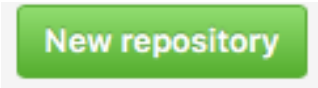
Add the project to a GitHub git repo

First of all, create a local git repo on your mac.
Go to **Source Control > Create Working Copy...**

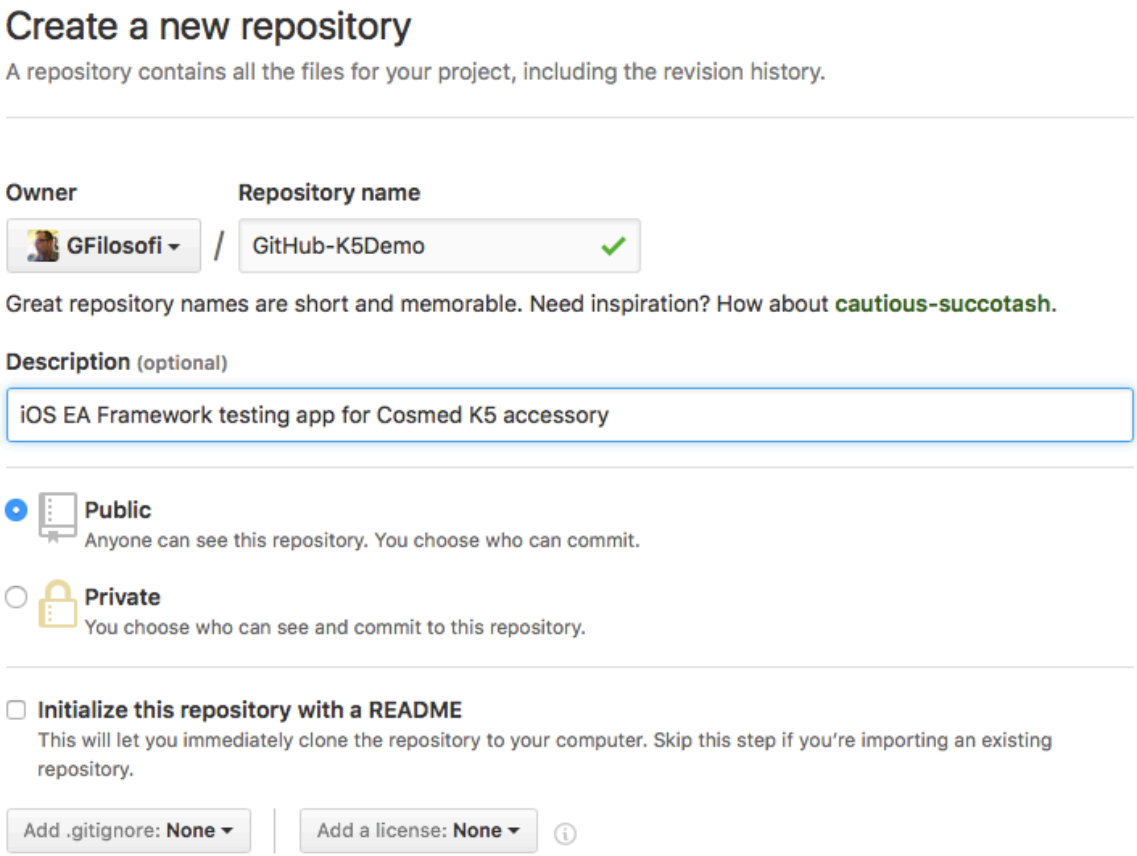


Select the project and hit **Create**, Xcode will automatically perform an initial commit for you.

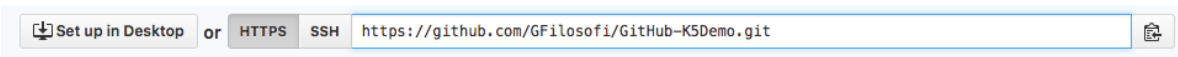
To add a remote repo,
Sign in to GitHub site
Click on New Repository button next to your account name



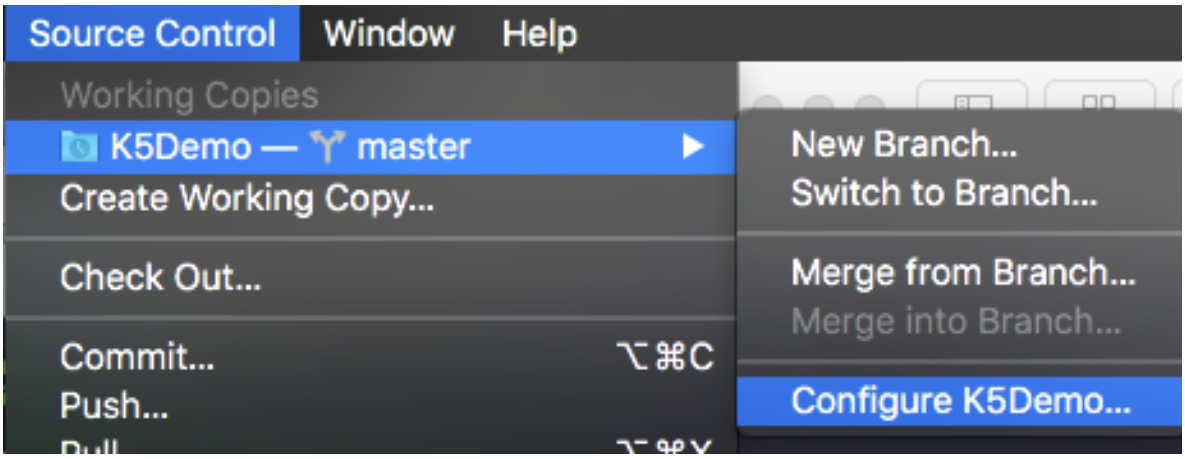
Type the repository name and description (use a name different from Xcode project name).
Do not initialize your repository with a README.



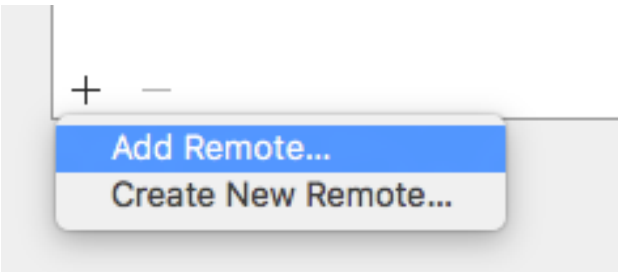
You get this



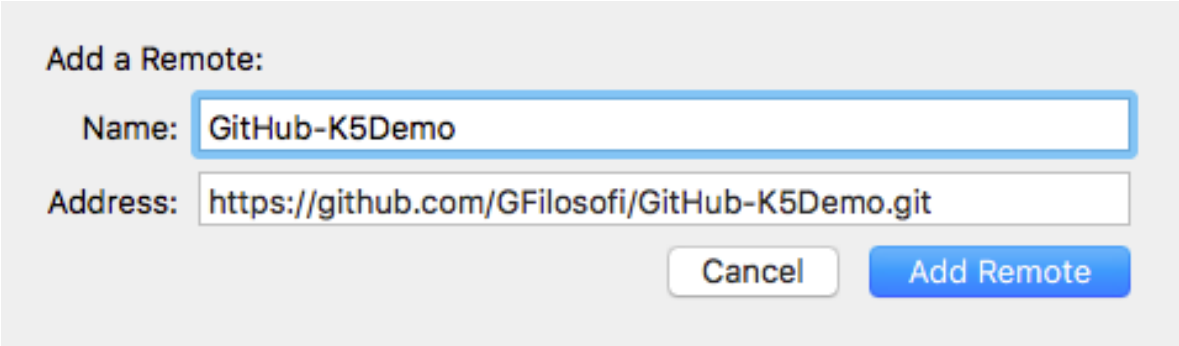
Copy the repo https/ssh link and go back in Xcode and select **Source Control > Configure > Remotes**



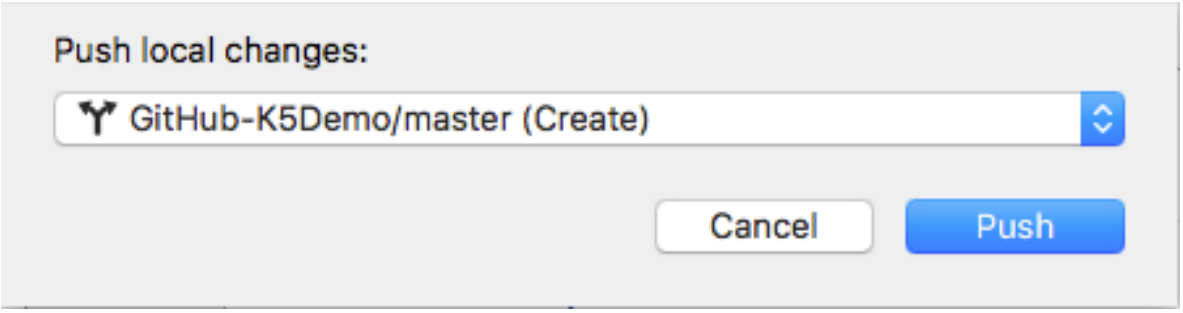
Click on the “+” sign in the bottom left corner and select Add Remote



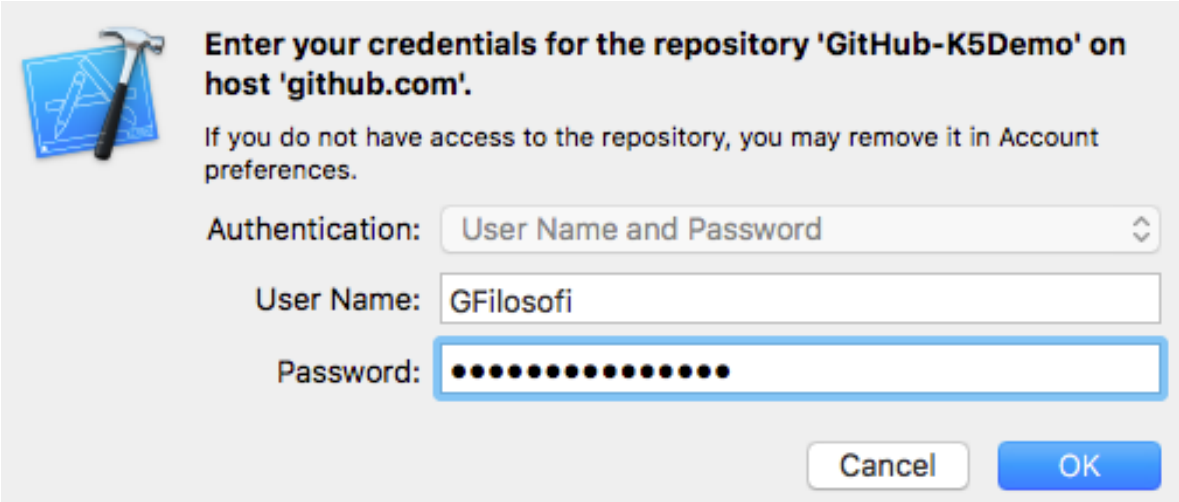
In the Address text field paste the repo link
In the Name field copy the same name you used in GitHub



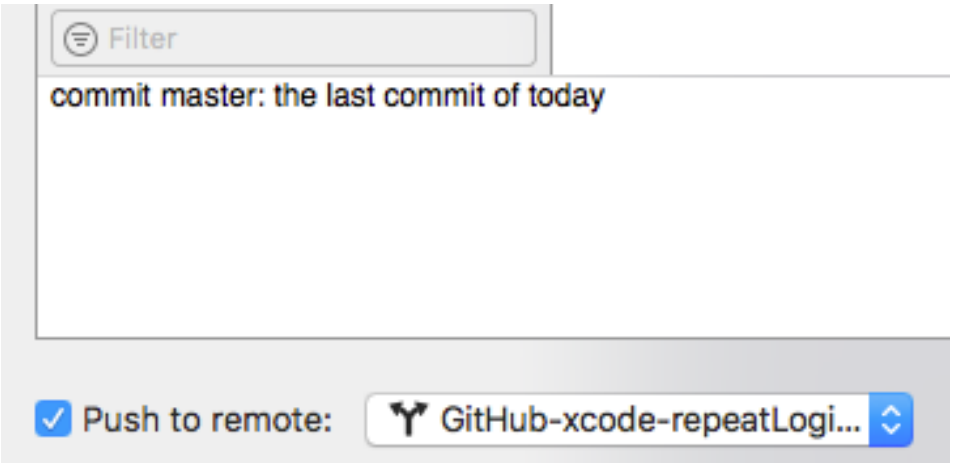
Click Add Remote. Click Done.
From **Source Control > Push** push the first (repo creation) commit



You will be asked to enter your GitHub credentials

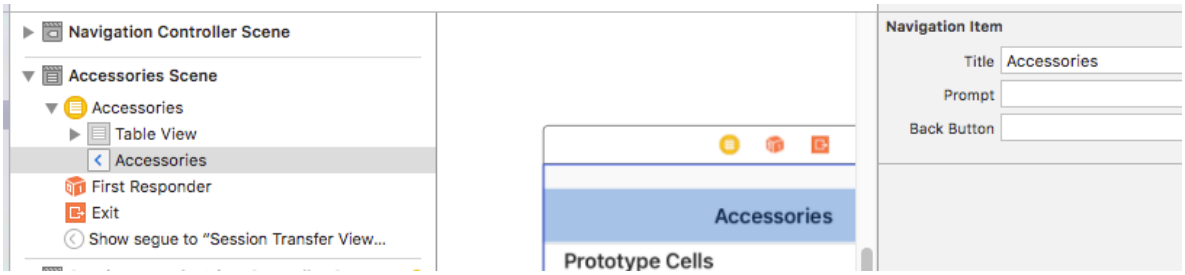


To push subsequent commits make sure to have checked the Push to remote box



Customization

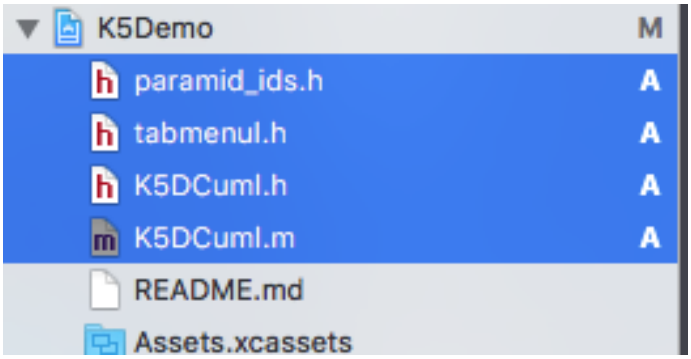
Open the main storyboard
Delete the Accessory table view
Add a new TVC and link it to the RootViewController class
Add a view segue from the Navigation Controller (NC) to the TVC
Assign the Title “Accessories” to the Navigation Item of the TVC



Add a view segue from the Accessories VC to the Session Transfer VC
In the EADSessionTransferViewController.m, comment out the send10KButtonPressed IBAction and add the cmdRingButtonPressed IBAction

```
//gf: send a IDOBJ_PARAM_RING to K5 to enable a beep
- (IBAction)cmdRingButtonPressed:(id)sender {
    uint8_t buf[2] = {0x00,0x01};
    cumlSystemDataTransmit(IDOBJ_PARAM_RING, buf, 2); // = {0xFC, 0x01, 0x00, 0x06, 0x10, 0x4A, 0x00, 0x00, 0x00, 0x00};
}
```

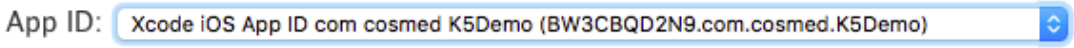
Add the following files to the project



Registering app Test Devices

Once you have completed the app development, you may want submit it to a limited number of selected tester. You can do it by accessing the developer account and creating an Ad Hoc provisioning profile

- specify the App ID



- add the test devices (for each one specify Device Name and UDID)
- specify the distribution certificate

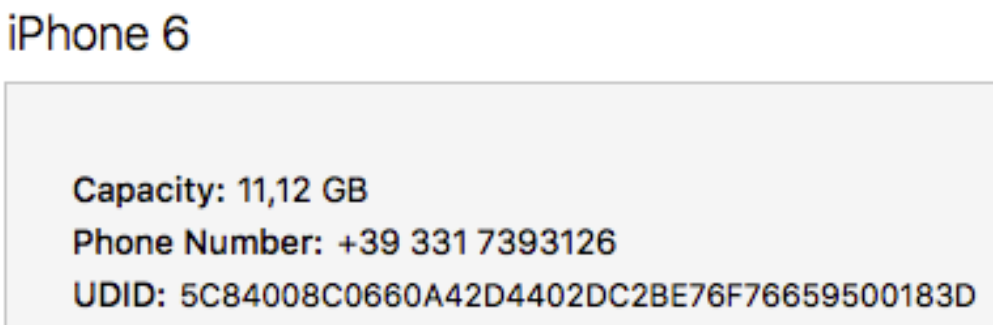
How do a tester locate the UDID of its device ?

He can either use iTunes or Xcode

- run iTunes on Mac
- connect the device to the Mac
- in the upper-right corner, select the device
- in the Summary pane, click the Serial Number label

UDID to be added for the adhoc app:
Sample 256 - 41717766dfe85b6970e080a8089e2d3de86b6a24
Sample 123 - ec0fddd7032c0eea5d15e33f07bdc557dbb75fcd

The last one is an iPhone 4s



Register the tester device

- Sign in to developer.apple.com/account, and click Certificates, IDs & Profiles.
- Under Devices, select All
- Click the Add button (+) in the upper-right corner
- Select Register Device
- Enter a device name and the device ID (UDID)

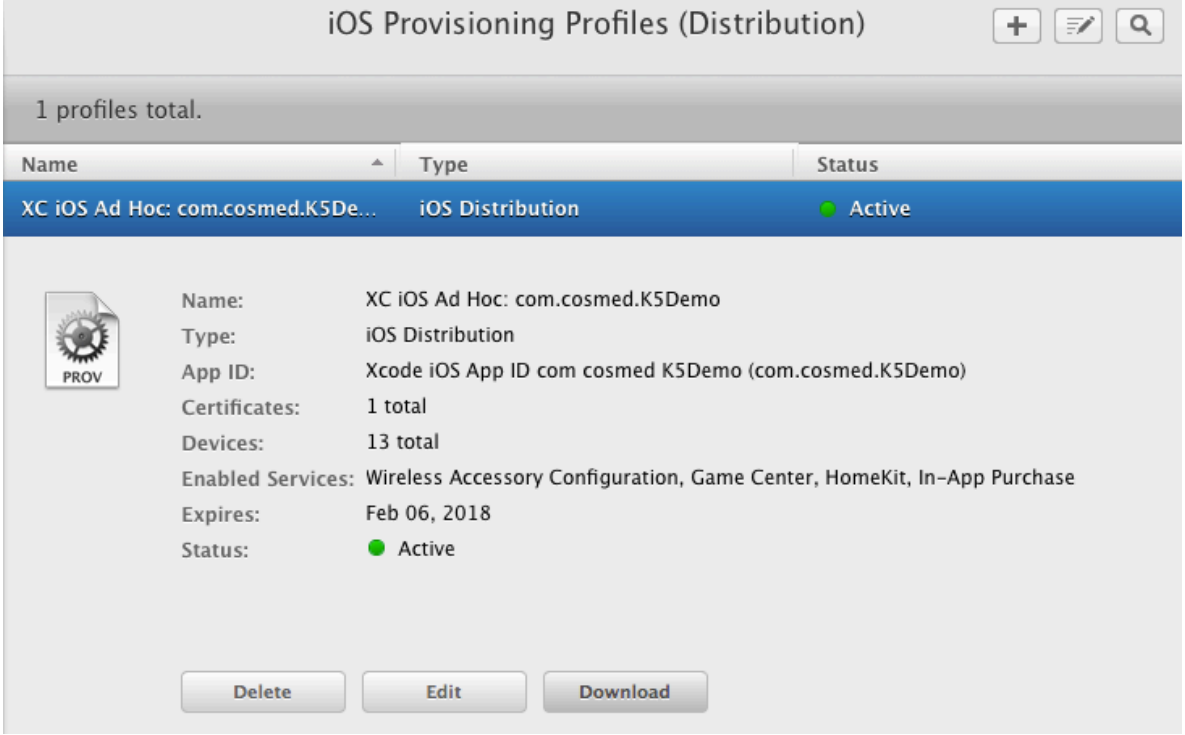
Register Device
Name your device and enter its Unique Device Identifier (UDID).

Name: Sample 256

UDID: 41717766dfe85b6970e080a8089e2d3de86b6a24

Create AdHoc provisioning profile

- Sign in to developer.apple.com/account, and click Provisioning Profiles, Distribution.
- Under Devices, select All

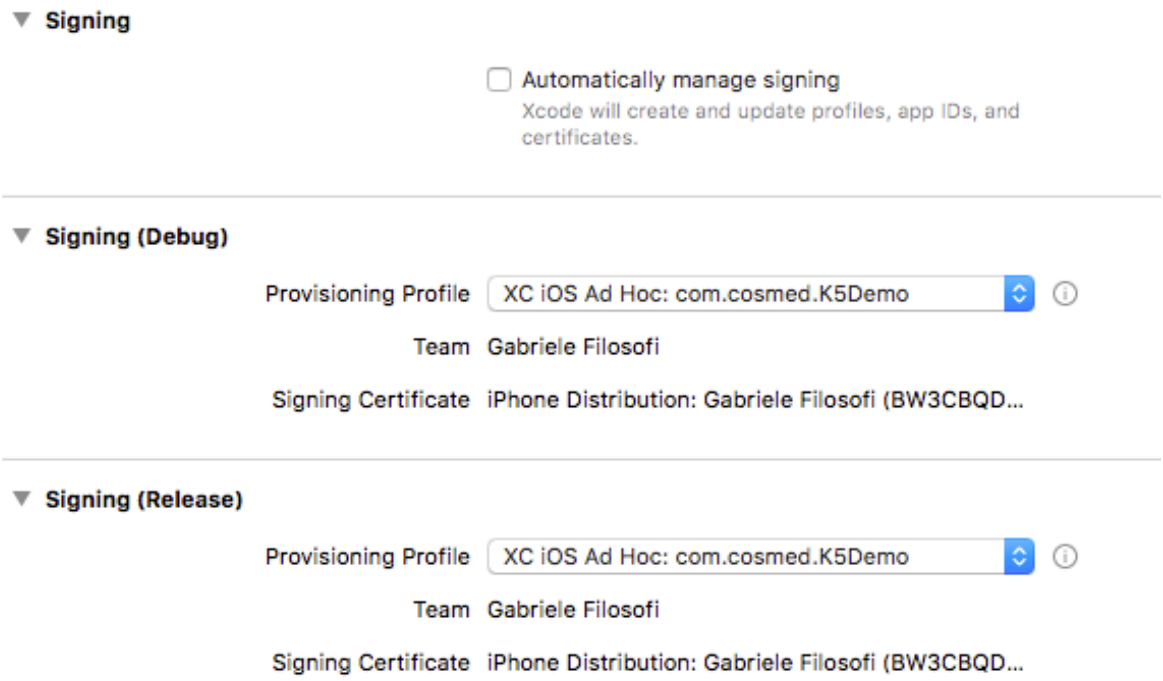


- In order to add a device to the existing Provisioning Profile press Edit and checkout the device. Then click Generate

Download provisioning profile

Provisioning Profiles	Expires	Action
K5Demo_provisioning_profile	29/11/2017	
XC iOS Ad Hoc: com.cosmed.K5Demo	06/02/20...	
iOS Team Provisioning Profile: *	06/02/20...	
iOS Team Provisioning Profile: *	06/02/20...	Download
iOS Team Provisioning Profile: com.cosmed.CosmedProdu...	14/06/2017	

Signing the app



Archive the app

- open Xcode
- select a generic iOS device (or your physical device) in the active scheme
- chose Product > Archive

Exporting the app for testing outside the Store

Because testers don't have Xcode to run your app, you create an iOS App file (a file with an .ipa filename extension) that they can then use to install your app on their device. Use this method to test a universal app that runs on all supported devices or test device variants that the store distributes later to users.

- Open the Archives organiser (choose Organizer from the Window menu), and select the archive.
- Click the Export button, select an export option, and click Next.

- To distribute your app to users with designated devices, select “Save for Ad Hoc Deployment.”
- In the dialog that appears, choose a team from the pop-up menu and click Choose
- In the Device Support dialog, choose whether to export the universal app or a variant for a specific device, and click Next

check provisioning profile

- Open Terminal and type
- in Xcode, under Product folder, right-click on K5Demo and find in Finder
- select the .

The iOS developer program with Xcode 8 and Swift 3

Getting Started

- ◯ Register as an Apple Developer
- ◯ Downloading Xcode 8
- ◯ Installing Xcode 8
- ◯ Building your first Swift 3 iOS App

Code with Swift 3

- ◯ Swift 3 types
- ◯ Variables
- ◯ Functions
- ◯ Booleans and conditional logic
- ◯ Constants and logical operators
- ◯ Arrays
- ◯ Loops
- ◯ Dictionaries
- ◯ Optionals
- ◯ OOP
- ◯ Inheritance
- ◯ Polymorphism

Version Control with Git and Github

- ◯ Coding Warmup Loops
- ◯ Git basics
- ◯ Setting up Github
- ◯ Github abd Bitbucket
- ◯ Local and remote git repositories
- ◯ Working through Git merge conflicts
- ◯ Github Desktop
- ◯ Pushing your code to Github

Foundational iOS

- ◯ Auto layout and project creation
- ◯ Width, height, leading and trailing constraints

- UIPickerView
- UiScrollView
- UIStackView
- Creating the data model
- IBOutlets and IBActions
- UISlider, math and logic
- Changing screen with segues
- UINavigationController
- How to use tab bars
- Custom fonts on iOS 10
- Playing audio files on iOS 10
- Custom table cells
- Showing Youtube videos in a web view
- MVC

Working with REST & Web Requests

- How web requests work on iOS 10
- Understanding JSON on iOS 10
- Talking to APIs with URLSession
- Parsing CSV files
- Search bar and Search filtering
- The API, Github, Cocoapods & Alamofire
- Downloading and parsing data

Protocol Oriented Programming with Swift 3

- Intro to Protocol Oriented Programming
- Write your first protocol
- Creating protocol extension
- Generics & protocols

iOS 10 Hot new features

- iOS 10 iMessage Sticker App Extension

Building Full Stack Apps with Firebase Push Notifications

- Creating the project
- Creating a push cert
- Connecting via code to Firebase messaging

Maps, GPS, Geolocation

Creating a Social Network or a Snapchat Cole with Firebase

- ◯ Sync data using the Firebase Realtime Database
- ◯ Manage Identity and Sign In with Firebase Authentication
- ◯ Store and access files using Firebase Storage
- ◯ Configure an application with Firebase Remote Config
- ◯ Track application usage flows with Firebase Analytics
- ◯ Display ads with AdMob

Designing mobile apps with Sketch 3

Objective-C and Swift 3: Getting Started

- ◯ Build an iOS app in Objective-C
- ◯ Anatomy of an Objective-C file
- ◯ Objective-C properties & instance variables
- ◯ Objective-C getters & setters
- ◯ Objective-pointers
- ◯ Objective-C Strings with NSString
- ◯ Objective-C numbers
- ◯ Objective-C conditionals & BOOL
- ◯ Objective-C methods
- ◯ Objective-C NSArray & NSMutableArray
- ◯ Objective-C NSDictionary & NSMutableDictionay
- ◯ Objective-C loops & fast enumeration
- ◯ Objective-C object oriented programming
- ◯ Objective-C weak vs strong & retain cycles
- ◯ Objective-C initialisers
- ◯ Objective-C Nullability
- ◯ Objective-C categories
- ◯ Swift & Objective-C interoperability
- ◯ Objective-C project creation and singleton
- ◯ Objective-C creating our Node server and designing the API for a Youtube app

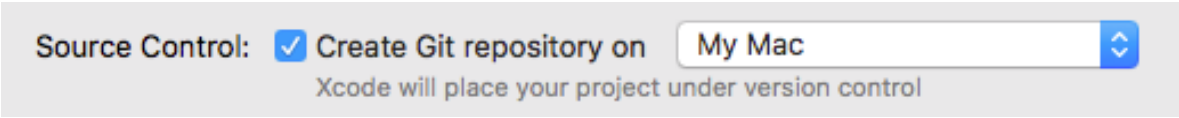
Provisioning, Testing, and App Submission

- [ ] Designing an iOS app icon in Photoshop
- [ ] Exporting 1x,2x and 3x images in Photoshop
- [ ] All about iOS 10 provisioning
- [ ] installing development certificates & profiles
- [ ] installing production certificates & profiles
- [ ] Adding externa beta testers
- [ ] Submitting an app to the App Store

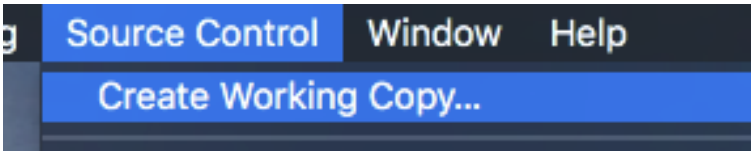
Using Git in Xcode

Local repo on Mac

For a new project, check the specific checkbox



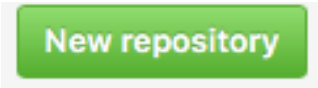
For an existing project, go to **Source Control > Create Working Copy...**



Select the project to add to the local Git repo and hit **Create**, Xcode will automatically perform an initial commit for you.
Change the project..
Now let's commit our changes. In the top menu, click **Source Control > Commit**
Add your commit message and click Commit in the bottom right once you're done

Remote repo on GitHub

Here are the main steps to add a remote repo for your Xcode project.
Sign in to GitHub site
Click on New Repository button next to your account name



Type the repository name and description (use a name different from Xcode project name).
Do not initialize your repository with a README.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner **Repository name**

GFilosofi / GitHub-xcode-K5Demo

Great repository names are short and memorable. Need inspiration? How about **congenial-octo-potato**.

Description (optional)

iOS EA framework testing app for Cosmed K5 system

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

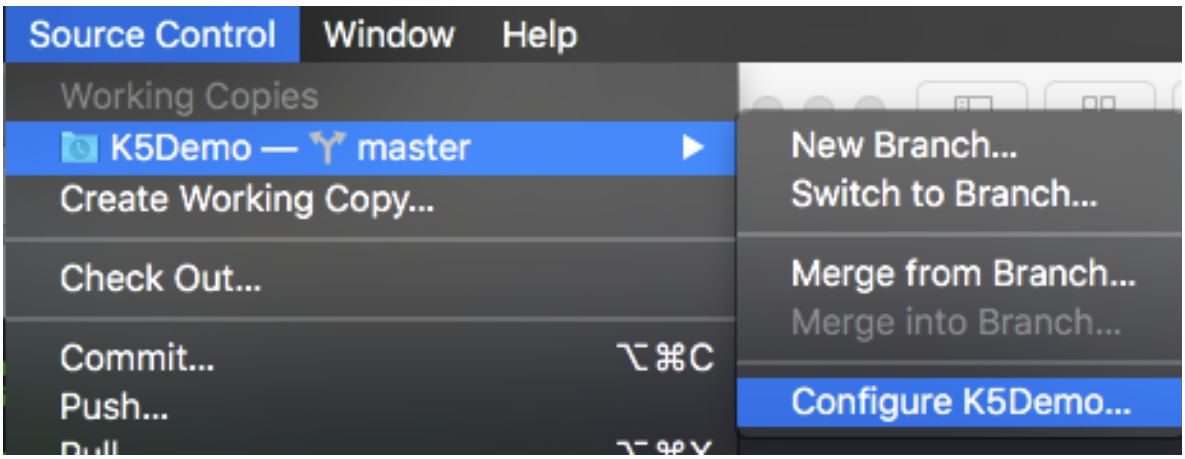
Click on Create repository button

Create repository

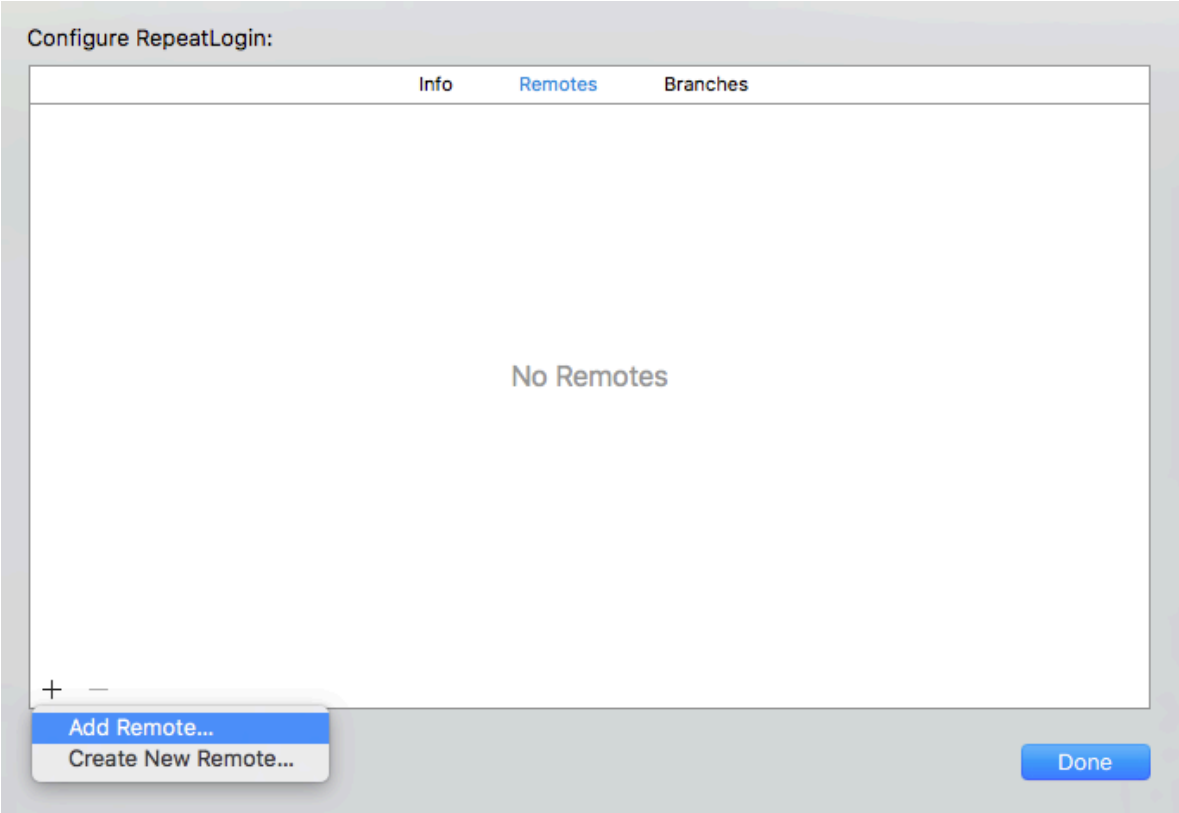
You will get something like this

Set up in Desktop or HTTPS SSH https://github.com/GFilosofi/GitHub-xcode-K5Demo.git

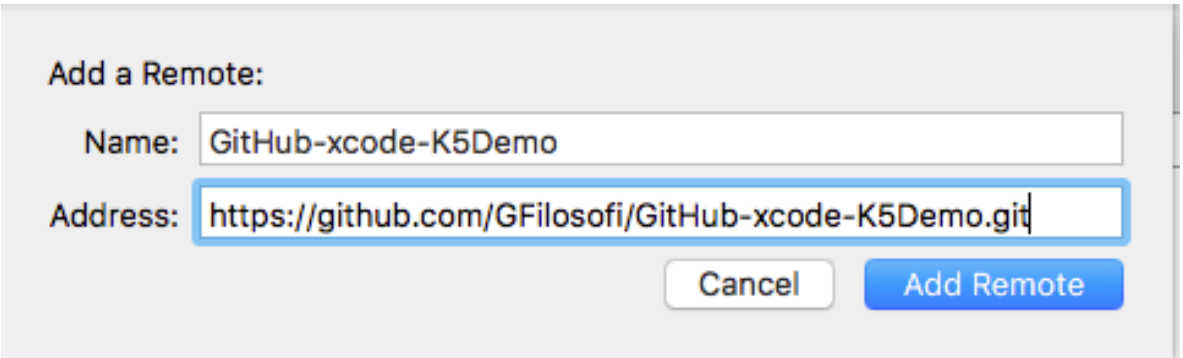
Copy the repo https/ssh link and go back in Xcode and select **Source Control > Configure > Remotes**



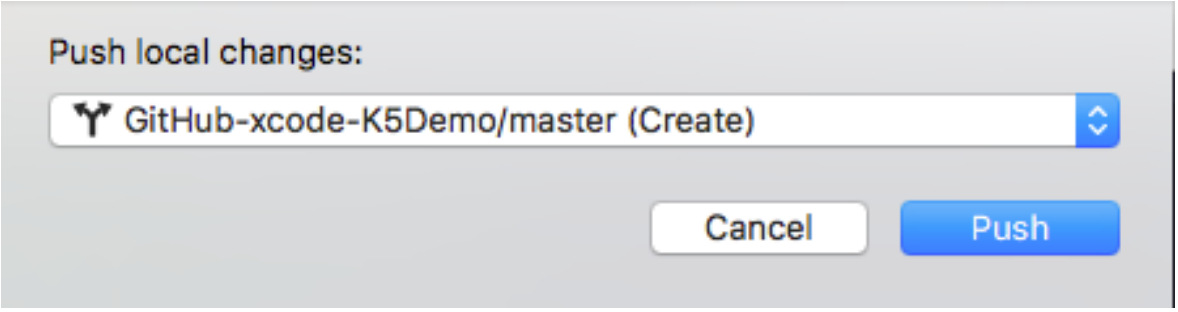
Click on the “+” sign in the bottom left corner and select Add Remote



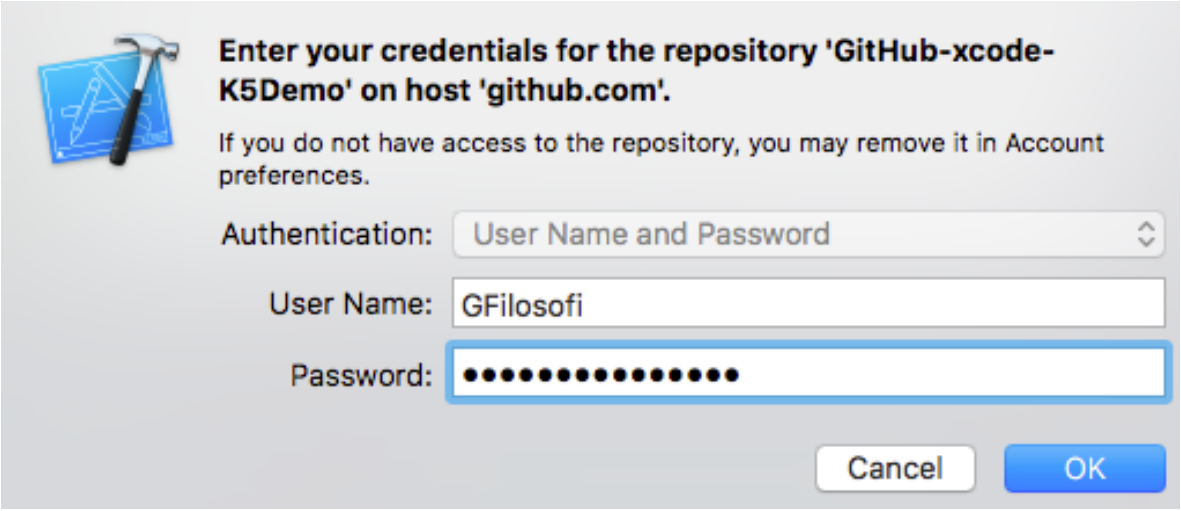
In the Address text field paste the repo link
In the Name field copy the same name you used in GitHub



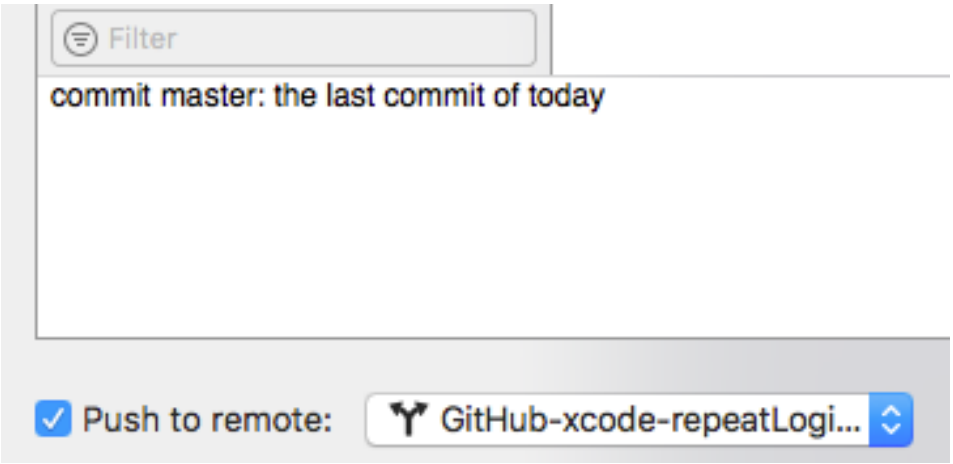
Click Add Remote.
From **Source Control > Push** push the first (repo creation) commit



You will be asked to enter your GitHub credentials



To push subsequent commits make sure to have checked the Push to remote box



Using JSON data in iOS

JSON (JavaScript Object Notation) is a standard way to encode data, either for storing or for client-server communication. Furthermore, JSON is an alternative to XML.
Typical constructs are
objects
  {string : value, string : value, ...}
array
  [value, value, ...]
value
  string | number | object | array | true | false | null

Example: An array of dictionaries looks like this

```
{"employees":[
	{"firstName":"Gabriele", "lastName":"Filosofi"},
	{"firstName":"Gianni","lastName":"Colarossi"}
]}
```

The same thing in XML is

```
<employees>
	<employee>
		<firstname>Gabriele</firstname> <lastname>Filosofi</lastname>
		<firstname>Gianni</firstname> <lastname>Colarossi</lastname>
	</employee>
</employees>
```

IP Geolocation API

A site which makes use of JSON APIs is
http://ip-api.com
This site provides free usage of its Geo IP API through multiple response formats.

It gives API to obtain geolocation information of a given IP
IP: **79.19.247.27**
Country: **Italy**
Country code: **IT**
Region: **Latium**
Region code: **62**
City: **Rome**

Zip Code: **00132**
Latitude: **41.8919**
Longitude: **12.5113**
Timezone: **Europe/Rome**
ISP: **Telecom Italia**
Organization: **Telecom Italia**
AS number/name: **AS3269 ASN-IBSNAZ**
DNS server: **85.37.17.16 (Italy - Telecom Italia)**
TCP/IP fingerprint: **1492 MTU, PPPoE, Mac OS X**
User-Agent: **Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Safari/602.1.50**

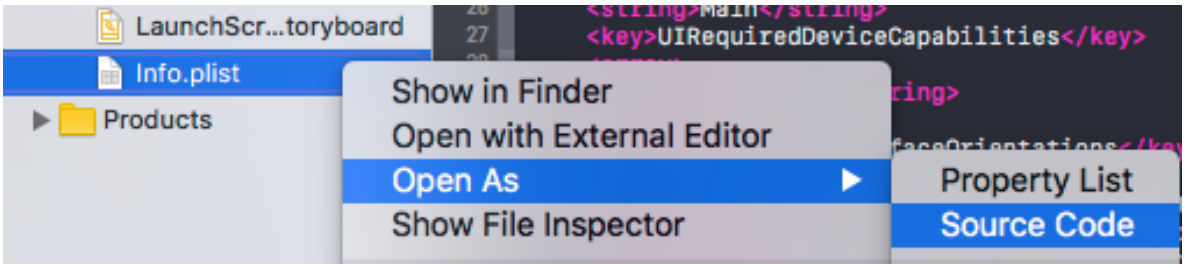
http://ip-api.com/json
{"as":"AS3269 ASNIBSNAZ",
"city":"Rome",
"country":"Italy",
"countryCode":"IT",
"isp":"Telecom Italia",
"lat":41.8919,
"lon":12.5113,
"org":"Telecom Italia",
"query":"79.19.247.27",
"region":"62",
"regionName":"Latium",
"status":"success",
"timezone":"Europe/Rome",
"zip":"00132"}

Another Example:
https://freegeoip.net/json/
{"ip":"79.19.247.27",
"country_code":"IT",
"country_name":"Italy",
"region_code":"62",
"region_name":"Latium",
"city":"Rieti",
"zip_code":"02100",
"time_zone":"Europe/Rome",
"latitude":42.3828,
"longitude":12.9073,
"metro_code":0}

JSON Example (simple app)

NSURLSession is the key object responsible for sending and receiving HTTP requests.

1. If the URS is https then you need an additional property in the Info.plist file. In this case you scan do that in a couple of different ways. So right click on plist file and open it as source code



Then add the following key and dictionary

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

Note that it is recommended to always use https url in your apps. ATS (App Transport Security) is an option that is good for you and your user and you shouldn't disable it!

1. In VC.swift, viewDidLoad, create a URLSession to download the json data

```
let url = URL(string: "http://ip-api.com/json")!
let task = URLSession.shared.dataTask(with: url, completionHandler: {(data, response, error) -> Void in
    guard let data = data, error == nil else { return }
    do {
        let json = try JSONSerialization.jsonObject(with: data, options: JSONSerialization.ReadingOptions.
            mutableContainers)
        print(json)
    } catch let error as NSError {
        print(error)
    }
})
task.resume()
```

2. Build and Run. In the output console you will see something like this

```
{
  as = "AS5396 MC-LINK";
  city = "Monte Porzio Catone";
  country = Italy;
  countryCode = IT;
  isp = "Mc-link SpA";
  lat = "41.8167";
  lon = "12.7167";
  org = "Mc-link SpA";
  query = "84.253.134.2";
  region = 62;
  regionName = Latium;
  status = success;
  timezone = "Europe/Rome";
  zip = 00040;
}
```

1. We need the JSONSerialization object to grab the son content just downloaded

```
        let url = URL(string: "http://ip-api.com/json")!
        let task = URLSession.shared.dataTask(with: url, completionHandler: {(data, response, error) -> Void in
            guard let data = data, error == nil else { return }
            do {
                let json = try JSONSerialization.jsonObject(with: data, options: JSONSerialization.ReadingOptions.
                    mutableContainers) as! [String:Any]
                //let's print some useful information grabbed from the json api
                print(json["city"]!)
                print(json["country"]!)
                print(json["countryCode"]!)
                print("lat: \(json["lat"]!)")
                print("lon: \(json["lon"]!)")
            } catch let error as NSError {
                print(error)
            }
        })
        task.resume()
```

WebViews (simple app)

1. create a new project and name it WebView
2. In the sloryboard add a UIWebView object at full screen
3. Ctrl + Drag the WebView object to the VC class as an Outlet variable
4. create a URLRequest for the web page you want to load on the screen, then load it using the loadRequest method

```
    @IBOutlet var webView: UIWebView!
    override func viewDidLoad() {
        super.viewDidLoad()
        let url = URL(string: "https://www.ecowebhosting.co.uk")!
        let request = URLRequest(url: url)
        webView.loadRequest(request)
    }
```

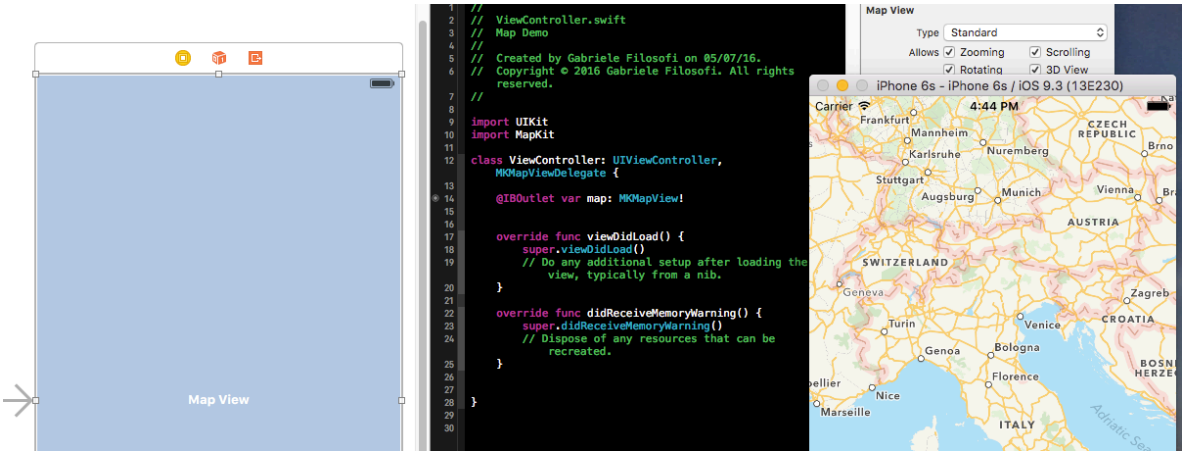
5. Build and Run. You will get the web page displayed on screen
6. Instead of displaying an existing web content you can also decide to build and display your own html content

```
        let html = "<html><body><h1>My Page</h1><p>This is my web page</p></body></html>"
        webView.loadHTMLString(html, baseURL: nil)
```

Using MapKit in iOS

To add a map to our Swift single view app

1. import UIMapKit in ViewController.swift and add the MKMapViewDelegate protocol to the class definition
2. add a Map Kit View object to the storyboard
3. Create an outlet for the map view in the ViewController.swift
4. Build and Run and you get the map view



1. Now we want to load the map on a specific location. To do that declare the following constants and then call the setRegion method

```
    override func viewDidLoad() {
        super.viewDidLoad()
        let latitude: CLLocationDegrees = 41.688959
        let longitude: CLLocationDegrees = 12.642983
        let location: CLLocationCoordinate2D = CLLocationCoordinate2DMake(latitude, longitude)

        let latDelta: CLLocationDegrees = 0.001
        let lonDelta: CLLocationDegrees = 0.001
        let span: MKCoordinateSpan = MKCoordinateSpanMake(latDelta, lonDelta)

        let region: MKCoordinateRegion = MKCoordinateRegionMake(location, span)
        map.setRegion(region, animated: true)
```

2. If we want add an annotation point in a particular location onto the map, define this in viewDidLoad

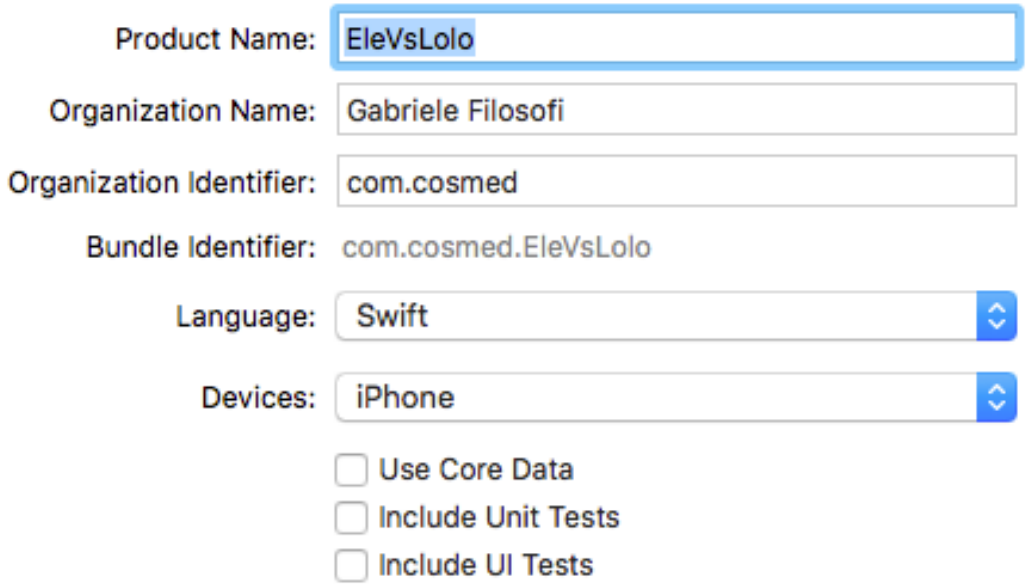
```
        //let's create an annotation on the map
        let annotation = MKPointAnnotation()
        annotation.coordinate = location
        annotation.title = "Via Marsala, 54"
        annotation.subtitle = "Here is LUISS ENLABS"
        map.addAnnotation(annotation)
```

3. If you need to add a gesture recogniser, for example a log press on the screen,

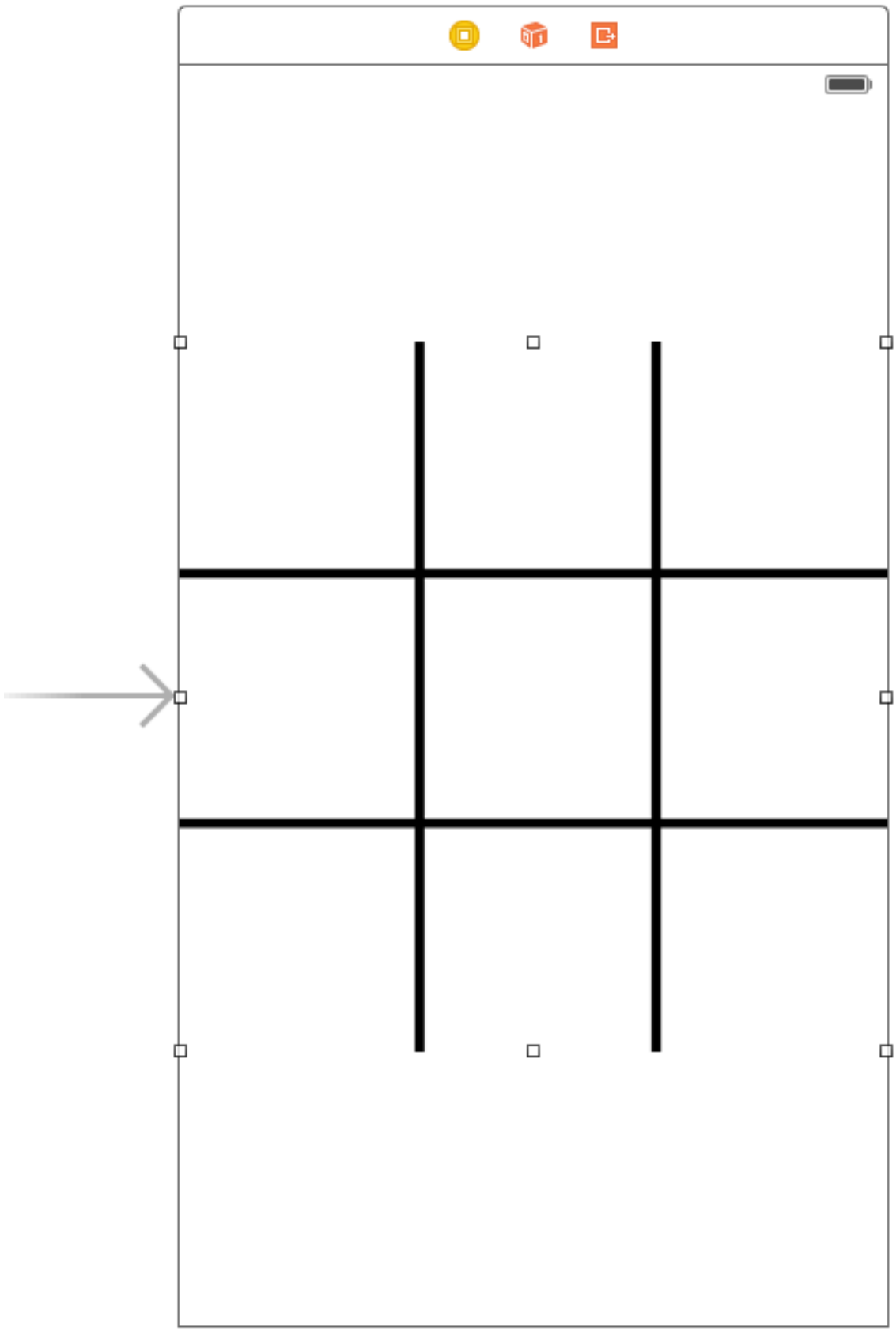
Using tags in iOS

Tags are integers you can associate with the view objects contained in a application.
Ion this app we recreate the classical Noughts and Crosses game.

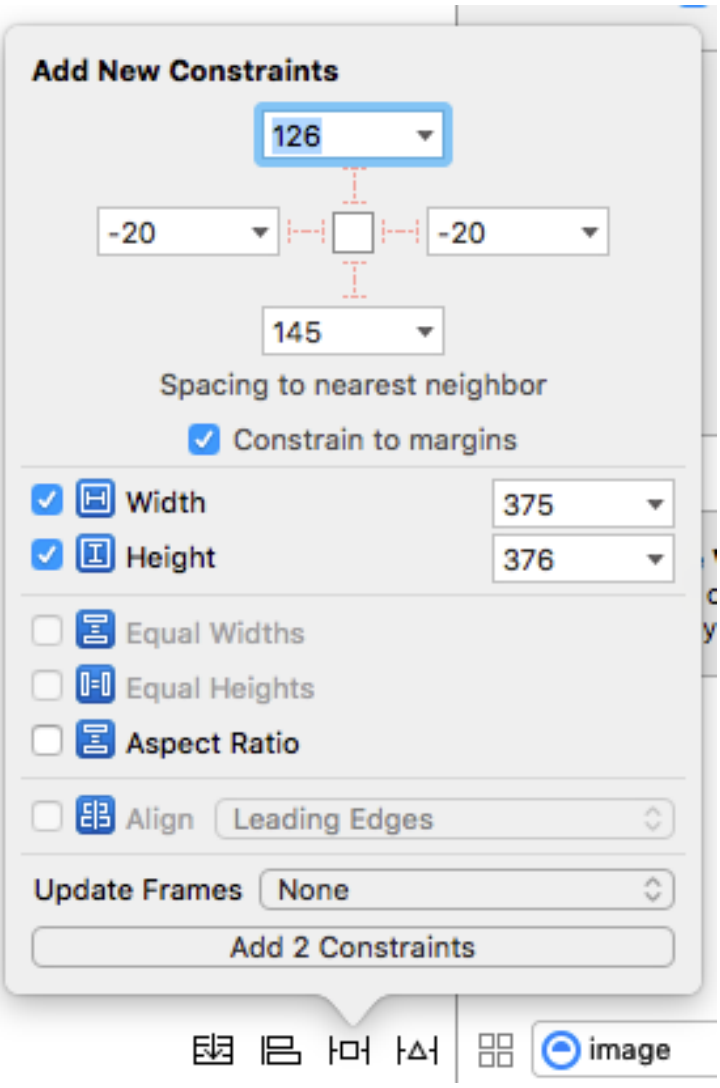
1. create a new Single View Application project for iOS

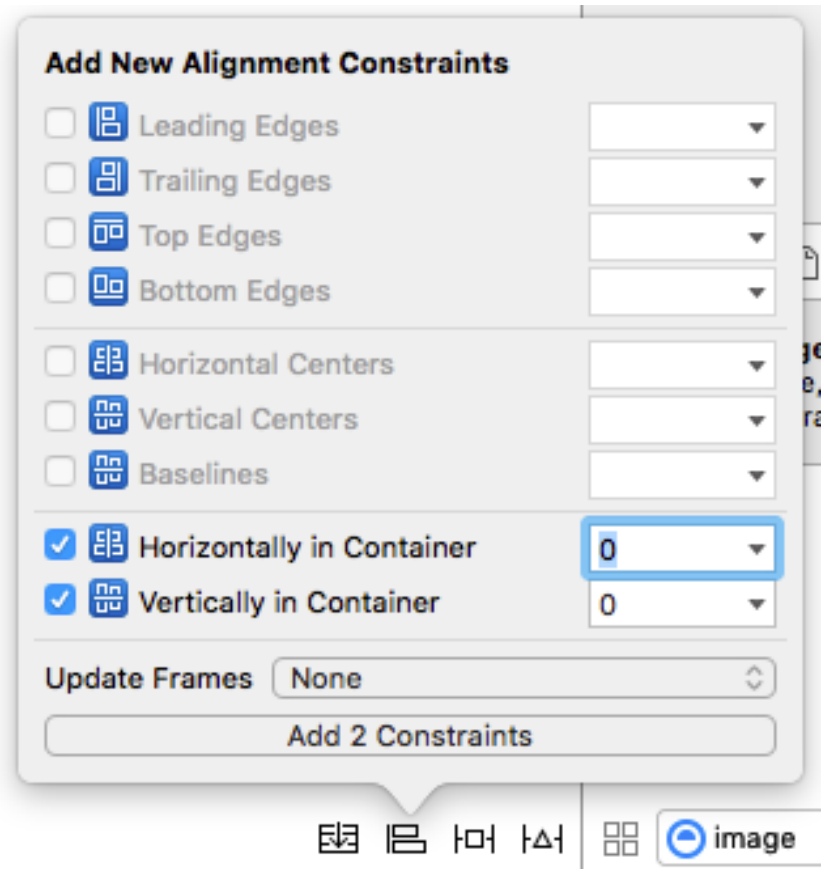


2. select the preferred device and the screen size
3. add an image object to the Storyboard and load the board.png image

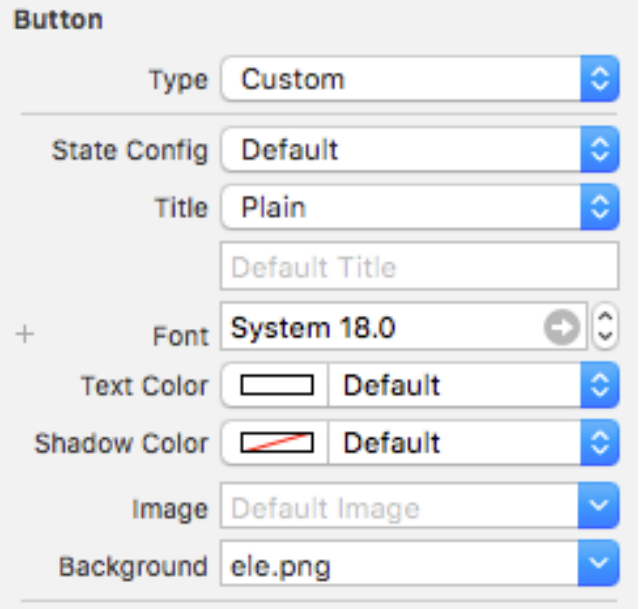


4. resize and add constraints (height and width, center)


Add New Constraints
126
-20
-20
145
Spacing to nearest neighbor
Constrain to margins
Width
375
Height
376
Equal Widths
Equal Heights
Aspect Ratio
Align
Leading Edges
Update Frames
None
Add 2 Constraints
image

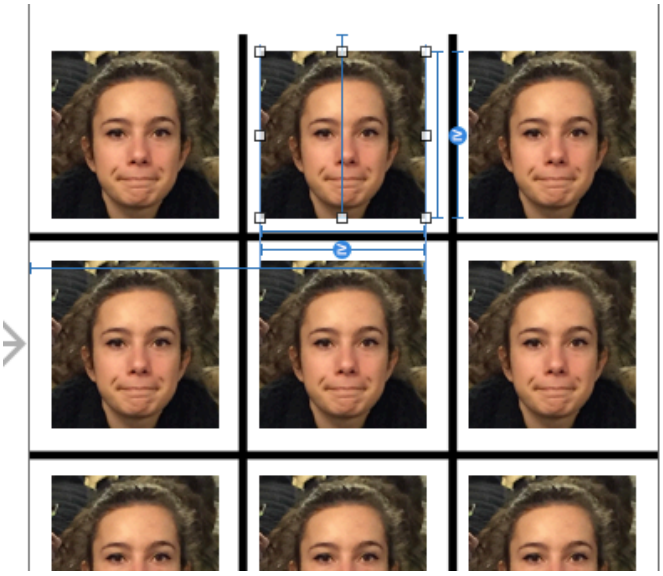


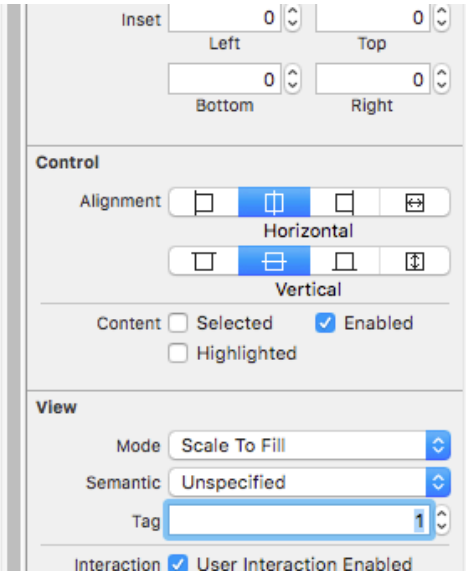
5. add a button in the first quadrant and load ele.png as the background image



6. resize and add constraints (height and width, horizontal and vertical spacing with respect to the board)

7. repeat the steps for all nine quadrants of the board
8. For each button asking a different tags, from 0 to 8





9. Add a IBOutlet and IBAction linked to the first button
10. In the IBAction toggle the button image form else to lolo and vice versa

```
11  class ViewController: UIViewController {
12
13      var playerId = 1 //1: ele, 2: lolo
14
15      @IBOutlet var button: UIButton!
16      @IBAction func buttonPressed (sender: AnyObject) {
17          if playerId == 1 {
18              button.setImage(UIImage(named: "ele.png"),
                    forState: .Normal)
19              playerId = 2
20          } else {
21              button.setImage(UIImage(named: "lolo.png"),
                    forState: .Normal)
22              playerId = 1
23          }
24      }
```

11. From each button, Ctrl+Drag to the IBAction
12. Add the array buttonState which contains the state of each button (0: empty, 1: ele, 2: lolo)
13. Add the check for win condition

```
11  class ViewController: UIViewController {
12
13      var gameState = 1 //0: inactive, 1: active
14      var playerId = 1 //1: ele, 2: lolo
15      var buttonState = [0,0,0,0,0,0,0,0,0]
16      var winPatterns = [[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],
            [2,4,6]]
17
18      @IBOutlet var button: UIButton!
19      @IBAction func buttonPressed (sender: AnyObject) {
20          if gameState == 1 && buttonState[sender.tag] == 0 {
21              buttonState[sender.tag] = playerId
22              if playerId == 1 {
23                  sender.setImage(UIImage(named: "ele.png"), forState: .Normal)
24                  playerId = 2
25              } else {
26                  sender.setImage(UIImage(named: "lolo.png"), forState: .Normal)
27                  playerId = 1
28              }
29
30              // check for win condition
31              for pattern in winPatterns {
32                  if buttonState[pattern[0]] != 0
33                  && buttonState[pattern[0]] == buttonState[pattern[1]]
34                      && buttonState[pattern[1]] == buttonState[pattern[2]] {
35                      print ("winner")
36                      gameState = 0
37                  }
38              }
39          }
40      }
```

14. Add gameState var to stop the game when someone wins
15. Add the check for game over with no winner.

```
        let uilgr = UILongPressGestureRecognizer(target: self, action: #selector(ViewController.action(_:)))
        uilgr.minimumPressDuration = 2
        map.addGestureRecognizer(uilgr)
```

where action is the func you are going to define which does what you want. For example, do the action add a new annotation point in the map point corresponding to the touch screen point

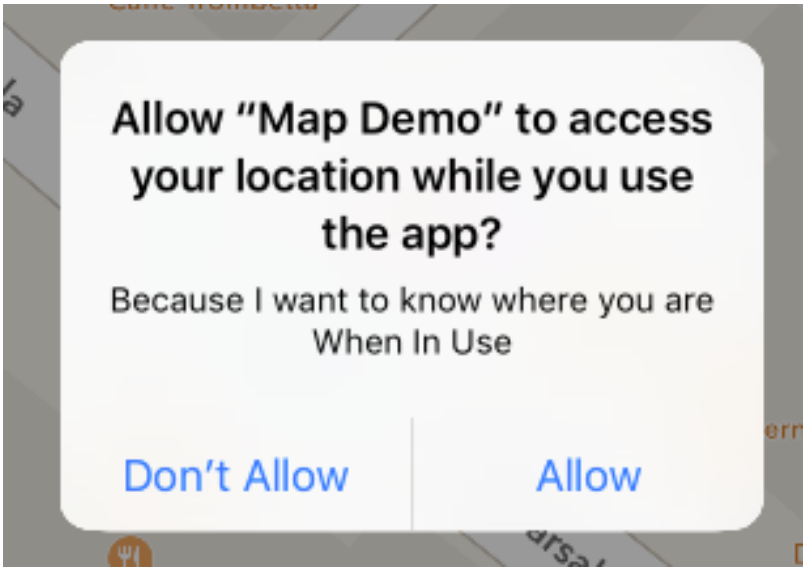
```
    func action(gestureRecognizer: UIGestureRecognizer) {

        //we want to create a new annotation point onto the map where we touched
        let touchPoint = gestureRecognizer.locationInView(self.map)
        let locationOnMap: CLLocationCoordinate2D = map.convertPoint(touchPoint, toCoordinateFromView: self.map)
        let annotation = MKPointAnnotation()
        annotation.coordinate = locationOnMap
        annotation.title = "Here you touched"
        annotation.subtitle = "Want to go there?"
        self.map.addAnnotation(annotation)
    }
```

4. Now we are going get and use the user's geolocation. To do that we have first to add the CoreLocation support by

- adding the binary library Core Location to the project's Build Phases
- adding the following two key-value items to the project's .plist file

NSLocationWhenInUseUsageDescritpion "Because I want to know where you are When In Use"
NSLocationAlwaysUsageDescription "Because I want to know where you are Always"

The first string will appear when the app is launched the first time, asking your permission to get location data.



The second string appears when the location data are going to be used by the app in background mode.
In the ViewController.swift we have to import CoreLocation and add CLLocationManagerDelegate protocol to the VC class

- Then we create an instance locationManager of the class CLLocationManager and initialize it in viewDidLoad with some settings as follows

```
import UIKit
import MapKit
import CoreLocation

class ViewController: UIViewController, MKMapViewDelegate,
    CLLocationManagerDelegate {

    @IBOutlet var map: MKMapView!

    var locationManager = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()

        //let's find the user's geolocation (uses CoreLocation and
            CLLocationManagerDelagate)
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.requestWhenInUseAuthorization()
        locationManager.startUpdatingLocation()
```

Not add a method outside the viewDidLoad to grab the location every time it is updated

```
    func locationManager(manager: CLLocationManager, didUpdateLocations
        locations: [CLLocation]) {
        print(locations)
    }
```

Now, when run the up on the device, you'll see the following location data in the Xcode output console

[<+41.71823873,+12.61179239> +/- 5.00m (speed 0.00 mps / course 210.94) @ 25/08/16 19 h 02 min 18 s Central European Summer Time]

If you are working with the simulator, go to Debug > Location to simulate different kind of location data
Now let's extract the latitude and longitude data from the location struct and use it to draw the map entered on it

```
    func locationManager(manager: CLLocationManager, didUpdateLocations
        locations: [CLLocation]) {
        let userLocation: CLLocation = locations[0]
        //extract the user's location
        let latitude = userLocation.coordinate.latitude
        let longitude = userLocation.coordinate.longitude
        //and draw a map centered in it
        let location: CLLocationCoordinate2D = CLLocationCoordinate2DMake
            (latitude, longitude)
        let latDelta: CLLocationDegrees = 0.001
        let lonDelta: CLLocationDegrees = 0.001
        let span: MKCoordinateSpan = MKCoordinateSpanMake(latDelta, lonDelta)
        let region: MKCoordinateRegion = MKCoordinateRegionMake(location,
            span)
        self.map.setRegion(region, animated: true)
    }
```

5. Now we want use reverse geocoding to translate the location data latitude

and longitude in the more familiar street, city, state, etc.
The geocoded object uses a network service to do that. The output is the placemark, a collection of user-friendly informations.
The forward process is known as forward geocoding
placemark —> forward geocoding —> location data
It is used to search for map location by name, address, etc. use MKLocationSearch API.
But let’s turn now to the reverse geocoding
location data —> reverse geocoding —> placemark
To implement it we use CLGeocoder class, create an instance of it and call the reverseGeocodeLocation:completionHandlaer: method. The result is delivered to the block object you provide.


```
{
"cache_control" = CACHEABLE;
"start_index" = 0;
status = "STATUS_SUCCESS";
ttl = 43200;
type = "COMPONENT_TYPE_ADDRESS";
value =    (
        {
    address =        {
      "known_accuracy" = PARCEL;
      "localized_address" =        (
                {
          address =        {
            formattedAddressLine =        (
              "Apple Inc.",
              "1 Infinite Loop",
              "Cupertino, CA  95014",
              "United States"
            );
            structuredAddress =        {
              administrativeArea = California;
              administrativeAreaCode = CA;
              areaOfInterest =        (
                "Apple Inc."
              );
              country = "United States";
              countryCode = US;
              fullThoroughfare = "1 Infinite Loop";
              geoId =        (
              );
              locality = Cupertino;
              postCode = 95014;
              subAdministrativeArea = "Santa Clara";
```

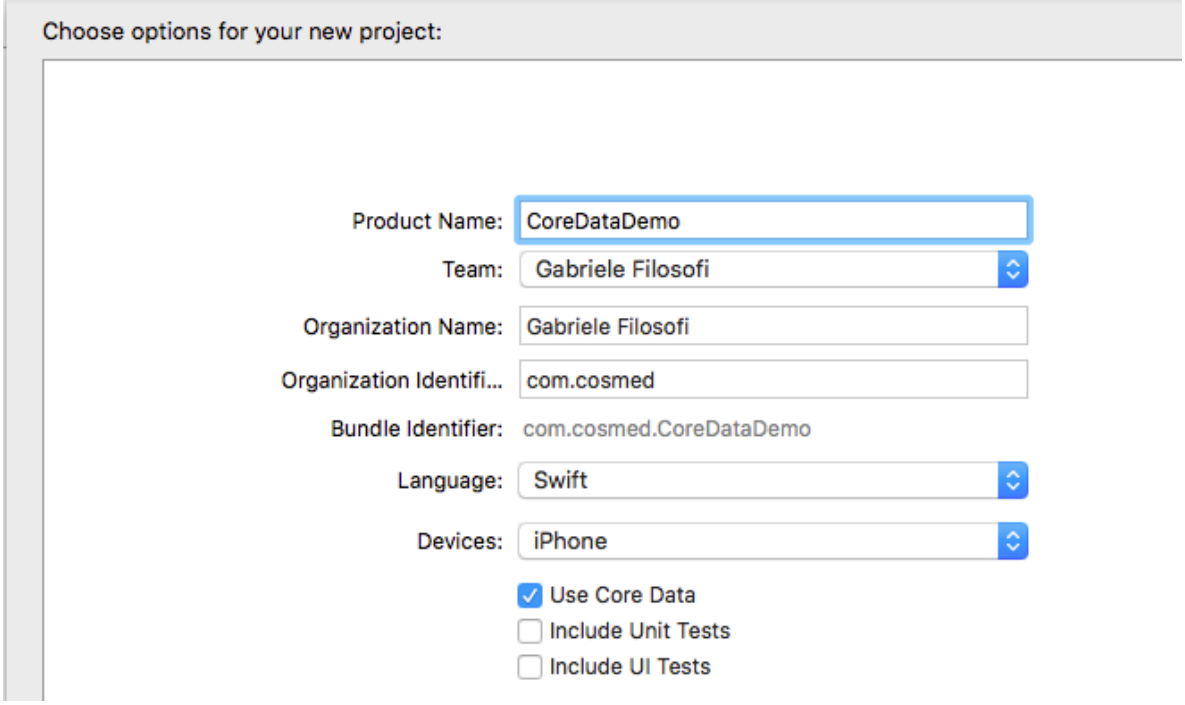
```
                    subThoroughfare = 1;
                    thoroughfare = "Infinite Loop";
                  };
                };
                language = en;
              }
            );
          };
        }
      );
      "values_available" = 1;
      version = 10;
      "version_domain" =              (
        apple,
        revgeo,
        US
      );
    },
```

Working with CoreData

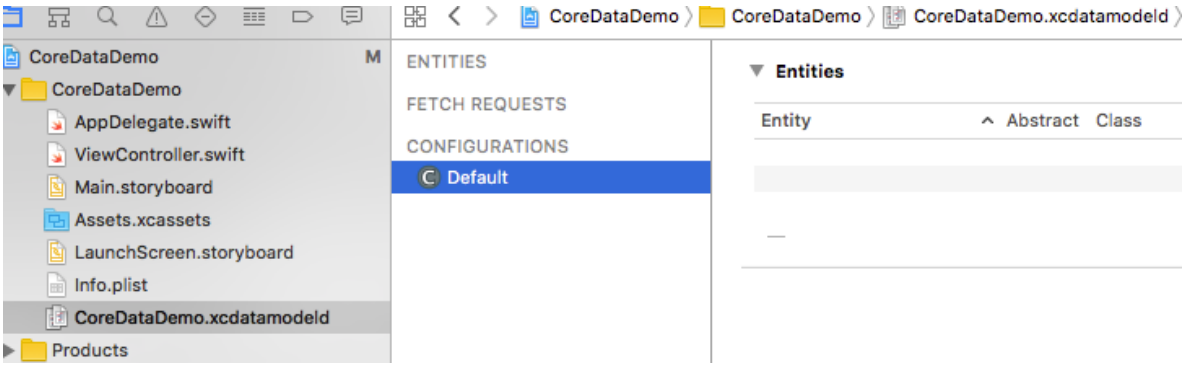
Core data is Apple's proprietary system for persistent data storage. It was INSANELY complicated to setup in iOS versions before iOS 10. It works very similar to a database. This is more powerful than a array or a dictionary, because you can perform multiple searches and provide data persistence.

CoreDataDemo app

1. We want to create a database of users and passwords.
2. create a new single view app, but check the "Use Core Data" box



3. You will notice a <projectName>.xcdatamodeld file has been added to the app bundle. It contains so called entities, which are tables of data.
4. So, let's create our data Model



5. Press Add Entity and rename it by clicking on it
6. An Entity is composed of many Attributes (or "managed objects"), which

are equivalent to the fields of a database

7. Add two Attributes, called “username” and “password”, of type String
8. In the VC.swift file let’s import CoreData. In didLoad function create a reference constant to the app delegate (AppDelegate.swift)
9. Create a context. It is the handler to access the Entity. Let’s create the function

```
    func getContext () -> NSManagedObjectContext {
        //get reference to the app delegate
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        //return the context to get access to managed objects
        return appDelegate.persistentContainer.viewContext
    }
```

10. context is an instance of NSManagedObjectContext which represents a single “object space” or scratch pad in an application. Its primary responsibility is to manage a collection of managed objects. These objects form a group of related model objects that represent an internally consistent view of one or more persistent stores. The context object has a central role in the life-cycle of managed objects, with responsibilities from life-cycle management (including faulting) to validation, inverse relationship handling, and undo/redo.
11. Now let’s create the function to add and store new users to our model

```
    func storeUser (_ name: String, _ passw: String) {
        let context = getContext()
        let entity = NSEntityDescription.entity(forEntityName: "Users", in: context)
        let user = NSManagedObject(entity: entity!, insertInto: context)
        user.setValue(name, forKey: "username")
        user.setValue(passw, forKey: "password")
        do {
            try context.save()
            userField.text = ""
            passwField.text = ""
            print("saved!")
        } catch let error as NSError {
            print("Could not save \(error), \(error.userInfo)")
        } catch {}
        userCount.text = "\(getUserCount())"
    }
```

12. And now create the function that search for the user having the longest password

```
55      func getUserWithLongestPassword () {
56          let context = getContext()
57          let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: "Users")
58          var longestPassw: String = ""
59
60          //scan all the users to find the longest password
61          do {
62              let results = try context.fetch(fetchRequest)
63              for user in results as! [NSManagedObject] {
64                  if (user.value(forKey: "password")! as! String).characters.count > longestPassw.characters.count {
65                      longestPassw = user.value(forKey: "password")! as! String
66                  }
67              }
68          } catch {
69              print("Error with request: \(error)")
70          }
71
72          //fetch the username associated to the longest password. To do that create a search predicate
73          fetchRequest.predicate = NSPredicate(format: "password = %@", longestPassw)
74          fetchRequest.returnsObjectsAsFaults = false
75          //submit the fetch request
76          do {
77              let results = try context.fetch(fetchRequest)
78              for user in results as! [NSManagedObject] {
79                  //print the username
80                  userLabel.text = "\(user.value(forKey: "username")!)"
81              }
82          } catch {
83              print("Error with request: \(error)")
84          }
85      }
```

Working with Gesture Recognizers in Xcode

WorkingWithSwipes

1. We want to print a string whenever the user swipes either left or right.
2. We need to add a UISwipeGestureRecognizer object for every swiping direction we want to detect. Let's add all four direction objects

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let directions: [UISwipeGestureRecognizerDirection] = [.right, .left, .up, .down]
        for direction in directions {
            let gesture = UISwipeGestureRecognizer(target: self, action: #selector(ViewController.
                handleSwipe(_:)))
            gesture.direction = direction
            view.addGestureRecognizer(gesture)
        }
    }

    func handleSwipe(_ sender: UISwipeGestureRecognizer) {
        switch sender.direction {
        case UISwipeGestureRecognizerDirection.right:
            print("UP")
        case UISwipeGestureRecognizerDirection.left:
            print("LEFT")
        default:
            break
        }
    }
```

WorkingWithShake

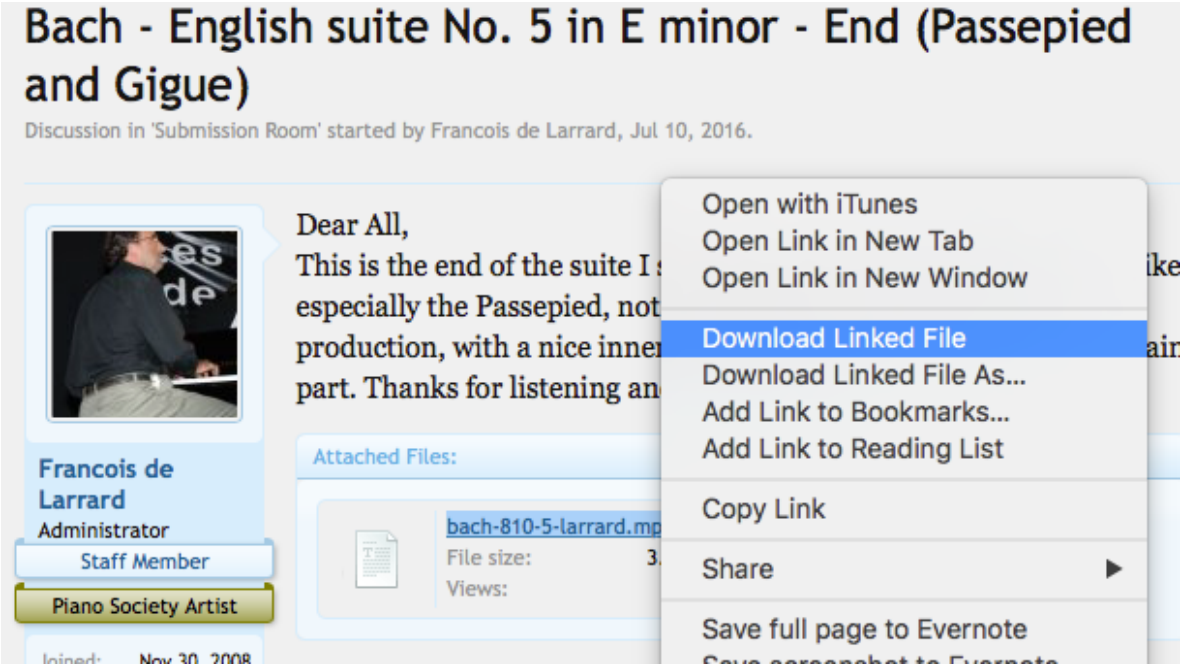
1. We want to print a string whenever the user shakes the device

```
    override func motionEnded(_ motion: UIEventSubtype, with event: UIEvent?) {
        if event?.subtype == UIEventSubtype.motionShake {
            print("SHAKED")
        }
    }
```

WorkingWithAudio App

Simple file playback

1. from www.pianosociety.com get some mp3 audio file



2. rename the file (ex. bach_audio.mp3)
3. Create a new project, name it WorkingWithAudio
4. drag the mp3 file to the project navigator pane
5. import AVFoundation in VC.swift
6. declare a variable of type AVAudioPlayer within the scope of the VC class. Initially it would be an empty object

```
class ViewController: UIViewController {
    var player = AVAudioPlayer()
```

7. as soon as the app loads we want play the mp3 file. Tp create a variable to store the file path use the NSBundle class to obtain the path to the file you want. Apple uses bundles to represent apps, frameworks, plug-ins,..The application is the main bundle. An Bundle object helps you access the code and resources in a bundle directory on disk. To get access to the main bundle and retrieve the audio file path

```
    override func viewDidLoad() {
        super.viewDidLoad()
        let audioPath = Bundle.main.path(forResource: "bach_audio", ofType: "mp3")
        do {
            try player = AVAudioPlayer(contentsOf: URL(fileURLWithPath: audioPath!))
        } catch {
            print("unexpected error!")
        }
    }
```

8. Now we add two buttons to the main View, Play and Pause, each with an

IBAction in the VC.swift. We also add a slider object called adjustVolume. In this case we need both an IBAction and an IBOutlet.

```
14      @IBAction func play(_ sender: AnyObject) {
15          player.play()
16      }
17
18      @IBOutlet var slider: UISlider!
19      @IBAction func pause(_ sender: AnyObject) {
20          player.pause()
21      }
22
23      @IBAction func adjVolume(_ sender: AnyObject)
24          player.volume = slider.value
25      }
```

Add scrubbing function

1. Now we want a more complete app with a Title Bar on top and a Bottom Bar. On the top bar we want a Play button. On the bottom bar we want a Pause button (left), and a Stop button (right). Use the Flexible Space Bar Button Item in order to create a distance among the Pause and Stop buttons.
2. download from internet a beautiful image of J.S.Bach, copy it into the main bundle and apply on a image view. In order to fit the image with any orientation it is better to fix width and height as constraints, not the aspect ratio.
3. add two sliders just below the image, one for volume control and one for scrubbing (change position in the soundtrack)
4. The scrub function can be accomplished using the property player.currentTime

```
34      @IBOutlet var scrubSlider: UISlider!
35      @IBAction func scrubSong(_ sender: UISlider) {
36          player.currentTime = Double(scrubSlider.value)
37      }
```

5. Before to start play don’t forget to set the maximum value of the scrub slider equal to the time duration of the soundtrack

```
14      @IBAction func playButton(_ sender: AnyObject) {
15          scrubSlider.maximumValue = Float(player.duration)
16          player.play()
17      }
```

6. the complete app looks like this

```
9   import UIKit
10  import AVFoundation
11
12  class ViewController: UIViewController {
13      var player = AVAudioPlayer()
14      @IBAction func playButton(_ sender: AnyObject) {
15          player.play()
16      }
17
18      @IBAction func pauseButton(_ sender: AnyObject) {
19          player.pause()
20      }
21
22      @IBAction func stopButton(_ sender: AnyObject) {
23          player.stop()
24          player.currentTime = 0.0
25          scrubSlider.value = 0.0
26      }
27
28      @IBOutlet var volSlider: UISlider!
29      @IBAction func adjVolume(_ sender: UISlider) {
30          player.volume = volSlider.value
31      }
32
33      @IBOutlet var scrubSlider: UISlider!
34      @IBAction func scrubSong(_ sender: UISlider) {
35          player.currentTime = Double(scrubSlider.value)
36      }
37
38      func updateScrubSlider() {
39          scrubSlider.value = Float(player.currentTime)
40      }
41
42      override func viewDidLoad() {
43          super.viewDidLoad()
44          let audioPath = Bundle.main.path(forResource: "bach_audio", ofType: "mp3")
45          do {
46              try player = AVAudioPlayer(contentsOf: URL(fileURLWithPath: audioPath!))
47          } catch {
48              print("unexpected error!")
49          }
50          scrubSlider.maximumValue = Float(player.duration)
51          scrubSlider.value = 0.0
52
53          //we use a timed function to animate the scrub slider position every second
54          _ = Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector
                (ViewController.updateScrubSlider), userInfo: nil, repeats: true)
55
56      }
57
58      override func didReceiveMemoryWarning() {
59          super.didReceiveMemoryWarning()
60          // Dispose of any resources that can be recreated.
61      }
62
63
64  }
```

Xcode Tips

How to install Xcode:
https://developer.apple.com/xcode/downloads

New, iOS Application, Single View Application
ToDoList, com.example,Objective-C language, iPhone
Prefix: TDL

The apple “Command" key (⌘) is the Windows key on a standard keyboard.
We denote it with “cmd" throughout this note

TIp: in order simulate quitting an app in iOS Simulator: "cmd + Shift + H"
tapping H twice simulates ..

Snapshots allow you to recover from a change in source code (recovering snapshots..)

Tip: press "cmd + 1" to open the Project Navigator
Tip: press "cmd + ," to open the Xcode Preference
Tip: press "ctrl + drag" to create actions and outlets from Storyboard to ViewController.swift
Tip: press “cmd + b” to build app
Tip: press “cmd + r” to run iOS Simulator
Tip: press “cmd + k” to switch on the keyboard on iOS Simulator
Tip: Take a screenshot of your whole screen

1. Press "cmd + Shift + 3"
2. Find the screenshot as a .png file on your desktop (unless redirected elsewhere)

Take a screenshot of a area

1. Press “cmd + Shift + 4”, then drag the area
2. Find the screenshot as a .png file on your desktop

Take a screenshot of a window

1. Press “cmd + Shift + 4”, then drag the area
2. Find the screenshot as a .png file on your desktop

Info: The "First Responder" is the element in the Storyboard that takes the priority at any given time

Remember to connect the delegating object (e.g. the outlet textField) to the delegate object (e.g our ViewController class)

A common mistake: If you made a mistake in connecting the elements or naming your properties and you want to re-do it, you can delete the property in

the .h file (or .swift file) but you also have to break the connection by going to your storyboard, right-clicking the element and clicking the “x” beside the outlet reference. If you don’t do this and you only delete the property from the .h file, then the element will be connected to a property that no longer exists and your app will crash

Info: MVC stands for Model-View-Controller, and it is a Design Pattern that holds also in C#, Java, .. Delegation is another Design Pattern. Is is similar to inheritance in C++

Info: method declaration in swift and Objective-C:
swift:
optional func textFieldShouldReturn(_ `textField`: UITextField) -> Bool
obj-C:
(BOOL)textFieldShouldReturn:(UITextField *)`textField`

Tip: to create an UIAlertController work directly in code

Info: The Button is one of the few iOS UI classes that don't need any delegation to work

Info: to add images click on Images.xcassetts and drag the image .png into the AppIcon section

Tip: to add NSConstraintLayout ctrl + drag the object into IB

Tip: sometimes you get ../Main.storyboard Frame for...will be different at run time, event you are sure everything is correct. To fix it, select the item in the storyboard and click Editor->ResolveAutoLayoutIssue->UpdateConstraints

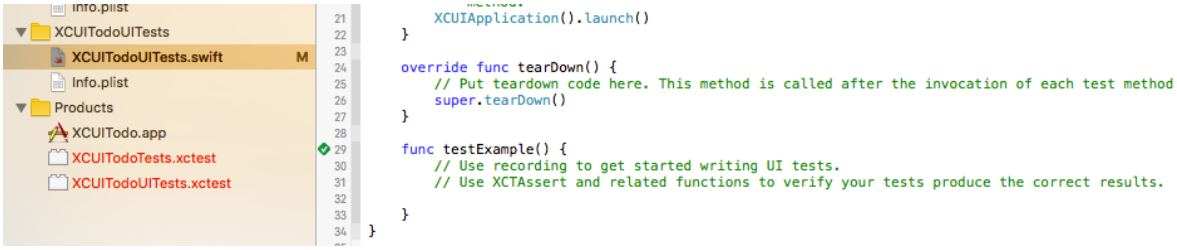
Tip: In Xcode 6, the Frameworks folder is not added by default. You can drag and drop your .framework files into the project navigator (tick 'Copy items if needed'), then *select them all > right click > "New Group from Selection"* and name the folder ‘Frameworks'

How to clone and run a project from a remote repo:
git clone https://github.com/bignerdranch/blog-ios-xcui-todo.git
cd xcui-todo
git checkout ui-tests-coming-soon
open */*.xcodeproj

Tip: In Xcode, to highlight the open curl bracket go to the closing bracket and move the cursor back and forth

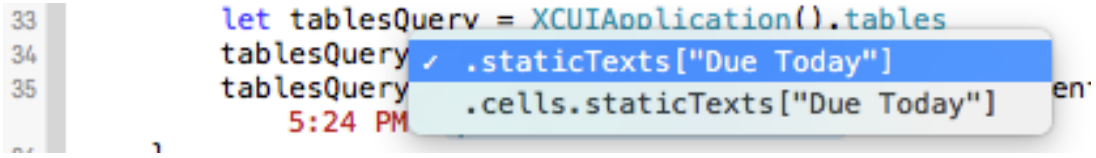
XCUI testing in Xcode

Open <prjName>UITests.swift
and put the cursor at the end of the testExample function

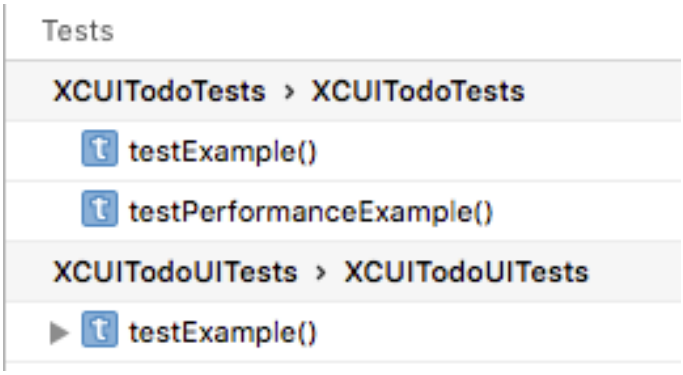


then click the little red dot at the bottom left corner of the editor pane, which is your “Record UI Test” button (if you don’t find the red button try to delete ~/Library/Developer/Xcode/DerivedData/ and let xcode to reindex your project). Now the function is going to be filled with the UI interactions.

Stop recording by clicking the Record button again.
A portion of statements show up inside a clickable token. When you click this token a menu pops up to allow to you select a different way of writing the recorded action. Double-click to accept the selected option.



Now, to run the recorded test automatically, Product > Test (or Cmd + U).
In the Report navigator, select Test. You will notice that it first runs the logic tests, then starts the UI tests, then launches the app.



Since we don’t have any logic tests, we can speed up our UI testing by editing the Test scheme to run only the UI Test :

- Navigate to menu item Product > Scheme > Edit Scheme...
- Select the Test action from the left pane.
- Select the Info tab from the right pane.
- Ensure that, in the Test column, the checkbox in the XCUITodoTests row is not checked, while the checkbox in the XCUITodoUITests row is checked.

Now, when you mash Cmd-U, it will run only the UI tests.