Bootstrapping iOS Application Development

Day 1

Chris Zelenak

12/02/2013

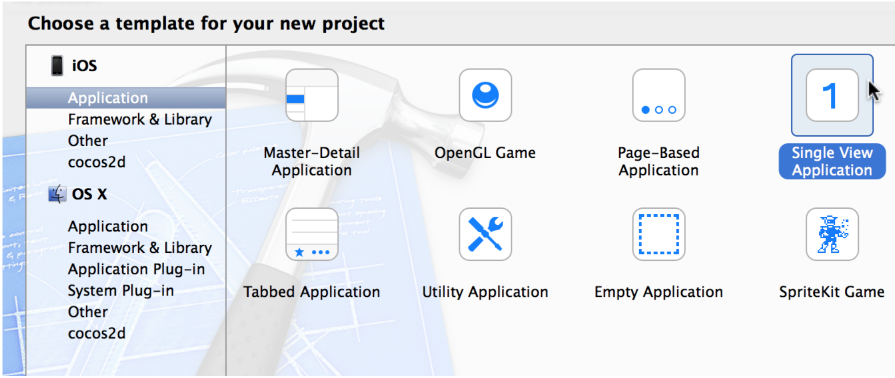
Welcome to the 5-Day Bootstrapping iOS Application Development class. Over the course of these next 5 days, we're going to be reviewing Objective C, UIKit, Core Animation, Core Data, and a host of other technologies you may only know by name.

Email me two pictures of you - one serious and one silly - to chris - at - fastestforward.com

1 Lab 1

Create a new Single View based application in XCode called Lab1

File › New › Project


Choose a template for your new project

iOS
- Application
- Framework & Library
- Other
- cocos2d

OS X
- Application
- Framework & Library
- Application Plug-in
- System Plug-in
- Other
- cocos2d

Figure : New Projects

Open Main.storyboard in Interface Builder

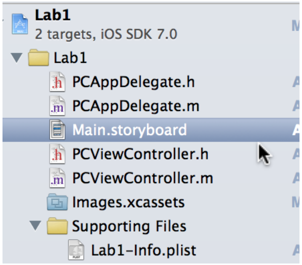
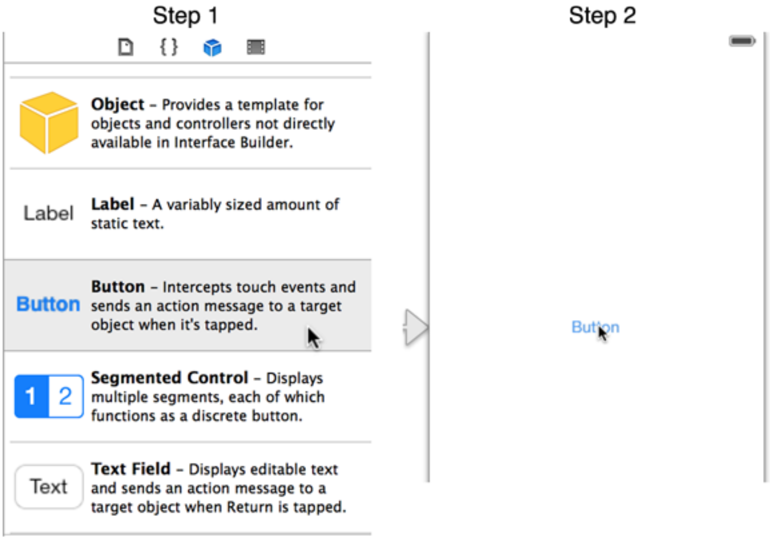


Figure : Main Storyboard

Add a button with the text “Hello iPhone”



Object – Provides a template for objects and controllers not directly available in Interface Builder.

Label – A variably sized amount of static text.

Button – Intercepts touch events and sends an action message to a target object when it's tapped.

Segmented Control – Displays multiple segments, each of which functions as a discrete button.

Text Field – Displays editable text and sends an action message to a target object when Return is tapped.

Figure : Drag Button to Storyboard

Add an IBAction selector to the view controller

```
// In the ViewController.h file
-(IBAction) helloWorldTapped:(id)sender;
```

Make it open an alert view

```
// In the ViewController.m file
-(IBAction) helloWorldTapped:(id)sender {
    [[[UIAlertView alloc] initWithTitle:@"Hello World"
                                message:@"Yay it worked"
                               delegate:nil
                      cancelButtonTitle:@"Dismiss"
                      otherButtonTitles: nil] show];
}
```

Connect the `IBAction` to the Button’s `Touch Up Inside` action.

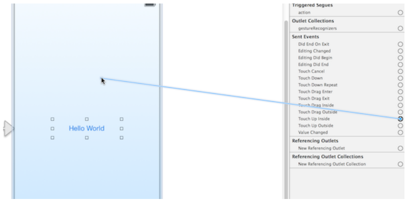


Figure : Connect Touch Up Inside event to IBAction

Run the simulator with ⌘ + R, the Product › Run menu, or the Play button

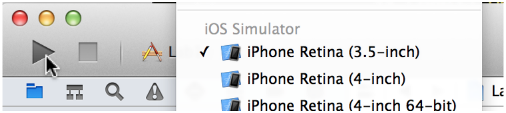
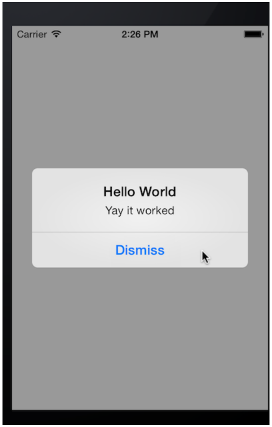


Figure : Launch Simulator



Hello World
Yay it worked
Dismiss

Figure : It Worked!

Objective C is a superset of C. It tacks on to C a message-passing object model similar to Smalltalk's (see Ruby), with support for reflection and limited runtime dynamic programming.

Objective C allows you to use weak and dynamicly typed language features along with the static typing of C; it also allows you to take advantage of runtime method resolution, so that your object's implementation of certain messages may be determined at call-time rather than at compile-time.

The language is still physically identical to C in many ways, in that the bulk of your development will be alternating between header files (.h files) and implementation files (.m files). You will also be doing a lot of memory management, which may fill you with either fear or joy, depending on your experience with garbage collected languages.

You will also be dealing a lot with many other C-family familiars, such as enumerated integer constants (enums), structs, pass-by-reference v. pass-by-value semantics and pointers. You don't need to have a very strong grasp on these things immediately, but the more iPhone programming and Objective C programming you do, the more your skills will benefit from a firm grasp of C.

Variable declaration in C follows the form `TYPE_DESCRIPTION NAME`, eg:

```c
int i = -1; // just an integer
unsigned int k = 0; // an unsigned integer
const unsigned int j = 0; // a constant unsigned integer, no reassignment
const char c = 'c'; // a constant char
const char * s = "a constant string"; // a pointer to a read-only char array
```

These are items you'll commonly encouter while doing iPhone programming whose roots belong in the C family:

```
typedef enum {
  NONE = 0,
  SOME,
  ALL
} HowMany;

HowMany i = SOME; // Use of enumerated constant
```

Note the `typedef` in this example, which is creating a type alias to be used later.

```
// Declaring a point in space type
typedef struct {
  float x;
  float y;
} APointInSpace;

APointInSpace point;
point.x = 20.0;
point.y = 410.0;

// C99 style struct instantiation
APointInSpace point2 = { .x= 21.0, .y=32.0 };
```

```c
#include <stddef.h>

// a constant byte array
char * constantCharArray = "Or in other words, a string";

// C99 style array instantiation
int aListOfNumbers[5] = { 1, 2, 3, 4, 5 };
int aListOfNumbers2[] = { 1, 2, 3, 4, 5};

// C99 Variable length array declaration
int someLength = 5;
float aListOfFloats[someLength];

// pointer declaration to NULL
int * someIntegerPtr = NULL;
int someInteger = 5;

// address dereference
someIntegerPtr = &someInteger;
```

```
float aSimpleAdditionMethod(float a, float b){
  return a + b;
}

float aSimpleSubtractionMethod(float a, float b){
  return a - b;
}

int aFloatAdditionThatReturnsAnInt(float a, float b){
  return (int) (a + b);
}

float (*anFnPtr)(float, float) = aSimpleAdditionMethod;
anFnPtr(1, 1); // returns 2
anFnPtr = aSimpleSubtractionMethod;
anFnPtr(1, 1); // returns 0
```

```c
#include <stdio.h>
#define FOO 1
#define IS_ZERO(n) (n == 0)

int main(){
  printf("FOO is %i\n", FOO);
  if(IS_ZERO(0)) {
    printf("This code is very useful.\n");
  }
}
```

If you never learned C or it’s been a long time, a fantastic book for you is “The C Programming Language”. It doesn’t cover many of the newer items in C99, but it is a thorough reference to the language fundamentals.

If you’d like to read more about the new features introduced by C99, read more at C99Changes.

```objc
NSString * anExampleString = @"This is a pointer to an NSString object";
int strLen = [anExampleString length];

NSString * aSecondString = [anExampleString
  stringByAppendingString:@" that had a second string appended to it"];

if([anExampleString
     compare:@"this is a pointer to an nsstring object"
     options:NSCaseInsensitiveCompare] == NSOrderedSame){
  NSLog(@"The string %@ was case insensitive equal", anExampleString);
}
```

Given some Objective C object, like:

```
  NSString * anObject = @"TheObject";
```

you can invoke functionality on that object via the form:

```
 [anObject theMethodName];
```

Method names in Objective C follow the form `initialMethodNameAndArgument:theSecondArgument:theThirdArgument:`
.

When you see references to “calling a method” or “calling a function” on an object in Objective C, it’s best to consider these synonymous with “dispatching a message”. Objective C objects respond to **messages**, and as you work more with Objective C, it will be more beneficial for you to perceive this as such.

An object in Objective C may be nil; nil is a “special” object which may have any message at all sent to it, to which it will respond with a nil object.

```
NSString * anObject = nil;
if([anObject whoaThisProbablyDoesntExist] == nil){
  NSLog(@"Ah well");
}
```

nil is the appropriate null object when dealing with pointers to Objective C classes; NULL, on the other hand, is the appropriate null value to use when dealing with pointers to all other types.

Much of the interaction with Objective C objects equates to the usual getter/setter functionality commonly associated with Object Oriented Programming. To make it easier for programmers to declare this functionality, properties were introduced as a way to eschew all the boilerplate code and provide a common framework upon which to enhance Objective C objects.

```
@interface MyObject : NSObject {

}
@property(strong) NSString * instanceVariable;
@property float numericVariable;
@end

@implementation MyObject

@end
```

```
MyObject * mo = [[MyObject alloc] init];

// Bracket Syntax
[mo setInstanceVariable:@"Test"];
[mo setNumericVariable:20.0f];
NSLog(@"The values I set were %@ and %f",
        [mo instanceVariable],
        [mo numericVariable]);

// Dot Syntax
mo.instanceVariable = @"Test 2";
mo.numericVariable = 25.0f;
NSLog(@"The values I set were %@ and %f",
        mo.instanceVariable,
        mo.numericVariable);
```

You can specify the memory management semantics (strong, weak, copy), atomicity (locking behavior), and access level in the property declaration. You can also specify your own implementation of the getter and setter methods.

```
    @property(readonly) NSString * privateName;
    @property(weak, nonatomic, setter=myAgeSetter:) NSNumber * age;
```

Properties do not merely provide getter/setter functionality. The use of properties also implicitly adds Key-Value-Observing functionality to your class, letting you automatically monitor classes for change events and performing specific code in such cases. Read more in the SDK Documentation on properties and Key Value Observing.

Selectors in Objective C are a way to indicate a message as a variable. They allow you to dynamically send a message to an object, as well as to query an object to see if it responds to a given message.

```
SEL aSelector = @selector(length);
NSString * foo = @"Foo";
if([foo respondsToSelector:aSelector]){
  NSLog(@"The length of foo is %i",  [foo performSelector:aSelector]);
}
```

Selectors can be built from strings, and can refer to any Objective C message that is forwarded to an object.

```
NSString * anAllCapsString = @"THIS STRING SHOULD BE LOWERCASE";
NSString * theMessageToSend = @"lowercaseString";
SEL aSelector = NSSelectorFromString(theMessageToSend);
if([anAllCapsString respondsToSelector:aSelector]){
  NSLog(@"All caps string (%@) converted: %@",
              anAllCapsString,
              [anAllCapsString performSelector:aSelector]);
}
```

Objective C provides only single-inheritance for its objects; to allow for situations where a class may provide functionality outside of its inheritance chain, the language provides Protocols, which are roughly analogous to interfaces in Java and C#. There are informal protocols (only referred to in documentation) and formal protocols, which are compiler checked. The majority of protocol usage in iOS programming is formal protocols.

```objc
@protocol CameraDevice

-(NSString *) manufacturerName;

@optional

-(int) flashStrength;

@end

@interface AnExpensiveCanonCamera : NSObject<CameraDevice>
@end

@interface TheCheapestCameraEver : NSObject<CameraDevice>
@end
```

```
NSArray * cameras = @[
  [[AnExpensiveCanonCamera alloc] init],
  [[TheCheapestCameraEver alloc] init]
];
for(NSObject<CameraDevice> * camera in cameras){
  if([camera respondsToSelector:@selector(flashStrength)]){
    NSLog(@"%@ camera has flash strength %i",
                 [camera manufacturerName],
                 [camera flashStrength]);
  } else {
    NSLog(@"%@ camera has no flash", [camera manufacturerName]);
  }
}
```

Categories allow you to mix in new code to existing classes without having to change the original source code for those classes. Examples of categories include automatically adding special serialization rules to NSObject.

You “can’t” add instance variables to a class with categories; they are purely for adding new methods to a class, but cannot change the memory layout of a class after the fact.

```objc
@interface NSDictionary(AsJson)

-(NSString *) asJson;

@end

@implementation NSDictionary(AsJson)

-(NSString *) asJson {
    NSData * jsonData = [NSJSONSerialization dataWithJSONObject:self
                                                         options:0
                                                           error:nil];
    return [[NSString alloc] initWithData:jsonData
                                 encoding:NSUTF8StringEncoding];
}

@end
```

```
#import "NSDictionary+AsJson.h"

void main(){
  NSLog(@"Dictionary as json: %@",
              [@{ @"A string": @"String!",
                  @"A number": @(20),
                  @"An Array": @[@(1), @(2), @(3)]} asJson]);
}
```

Tony Orlando & Dawn/Greatest Hits
Tie A Yellow Ribbon 'Round The Ole Oak Tree
Say, Has Anybody Seen My Sweet Gypsy Rose
You're A Lady
Who's In The Strawberry Patch With Sally
You Say The Sweetest Things
Steppin' Out, I'm Gonna Boogie Tonight
Candida
Knock Three Times
What Are You Doing Sunday
Summer Sand
Look In My Eyes Pretty Woman

Figure : Memory Management circa 1973

C based languages have traditionally used `malloc` and `free` as the means by which memory was allocated on demand. This code looked something like:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
  int * anInteger = malloc(sizeof(int));
  *anInteger = 10;
  printf("Integer value is %i", *anInteger);
  free(anInteger);
  return 0;
}
```

`malloc` and `free` allocated memory on the heap; it was the programmer's responsibility to indicate when a particular item was ready to be released back to the operating system.

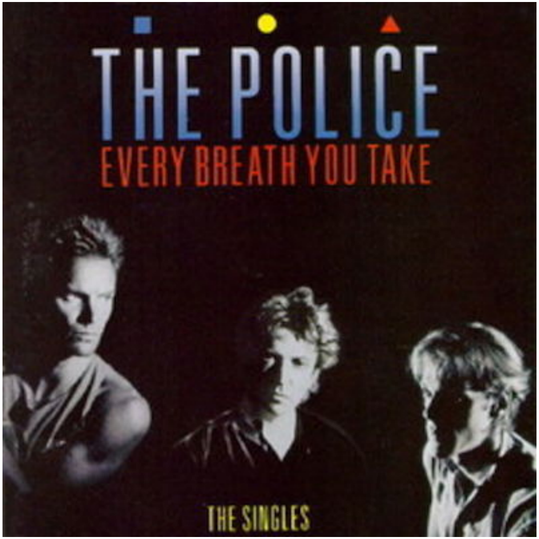


Figure : Memory Management circa 1983

Objective C used an “easier” means of managing the memory for its objects: reference counting.
Reference counting allowed Objective C programmers to write code that looked like this:

```
NSString * c = [[NSString alloc] initWithString:@"This is a string"];
NSLog(@"String is %@", c);
[someOtherObject setString:c];
[c release];
```

Each object would respond to two methods: `retain` and `release`. Retain would increment an object's retain count, and release would decrement. When an object's retain count reached 0, it was considered "deallocated", and its memory on the heap would be available for use by other objects.

Objects could also be flagged as `autorelease`, which would send a release to an object at some point in the future. Typically this meant after one main NSRunLoop cycle (one UI loop).

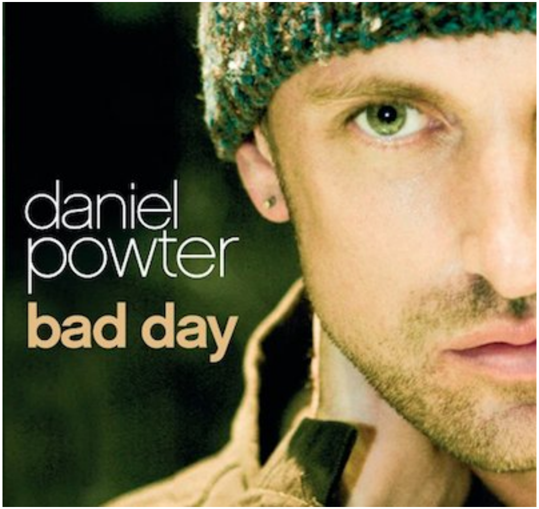


Figure : Memory Management circa 2006 - Apple adds Garbage Collection to Objective C

Figure : Memory Management circa 2008 - The iOS SDK 2.0 and App Store launch. The iPhone does not support Garbage Collection due to performance concerns.

Figure : Memory Management circa 2011 - ARC

Apple introduces a new tool called ARC (*Automatic Reference Counting*). Improvements in static code analysis allow Apple to detect and automatically insert retain/release calls for most code.
In other words, *most of the basic memory management is now done for you*.

What ARC *will do* for you:

- Let you avoid calling `retain` or `release` on Objective C objects

What ARC **will not do** for you:

- Prevent cyclic memory ownership
- Allow you to ignore `malloc` or `free` when using C-based apis
- Allow you to totally ignore memory management

```objc
NSString * s = [[NSString alloc] initWithString:@"Some string"];
NSLog(@"My string is %@", s);
// That's it.
```

```objc
NSString * s = [[NSString alloc] initWithString:@"Some string"];
[someOtherObject setString:s];

// and in the someOtherObject's implementation

@interface SomeOtherObject

@property(strong,nonatomic) NSString * string;

@end
```

Usage of ARC is by default on all new iOS projects. You can choose to disable it and manually manage memory. You can also still manually manage memory created through `malloc` and `free` if necessary.

Objective C is a deep language. There are lots of resources on the web and your computer where you can learn more. The best is easily the XCode documentation, but there are blogs out there that can provide invaluable help:

- Mike Ash's NSBlog
- objc.io
- NSHipster
- Apple Developer Forums

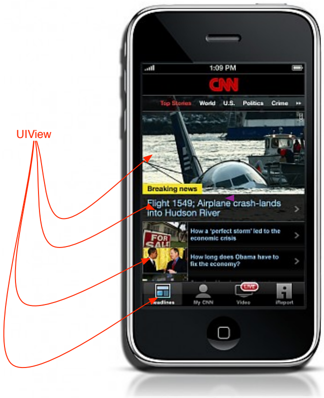
1 iPhone App Layout Conventions

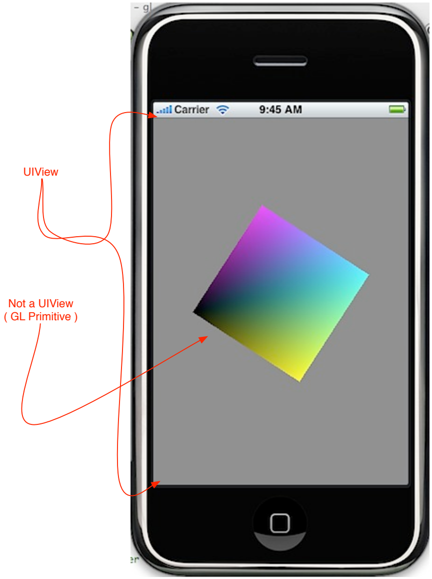
Apple strongly encourages you to adhere to Model View Controller pattern

Figure : Model, View and Controller

On iOS, a UIApplication (your app) typically has one UIWindow, which is the primary UIView upon which everything else displays. A UIView is “just” a rectangle upon which you draw things.

Almost everything visual is a UIView (except the things that aren’t).


UIView
1:09 PM
CNN
Top Stories
World
U.S.
Politics
Crime
Breaking news
Flight 1549; Airplane crash-lands into Hudson River
How a 'perfect storm' led to the economic crisis
How long does Obama have to fix the economy?
Headlines
My CNN
Video
iReport


- gl
Carrier
9:45 AM
UIView
Not a UIView
(GL Primitive)

UIViewControllers are objects that manage what the views on screen are currently doing; typically where you write event handling code, user interaction code, and interact with your models.

There is no “official” model object in this pattern. Your business objects are the models, however you decide to implement them.

Storyboards let you visualize the interaction between UIViewControllers. You can design both the UIView layout and the interaction between UIViewControllers using the Storyboard designer.

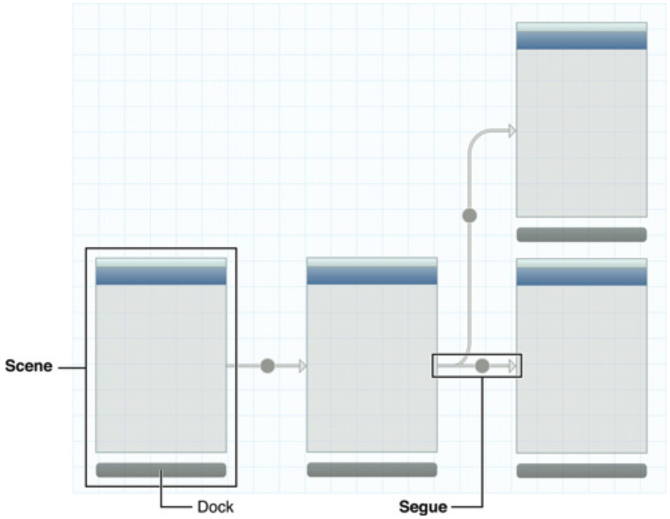


Figure : The Storyboard designer

Storyboards are comprised of **scenes** and **segues**. A **scene** represents a UIViewController's presentation. A **segue** is the transition between **scenes**, and can package essential information about how that transition occurred.

Create a new “Single View” project

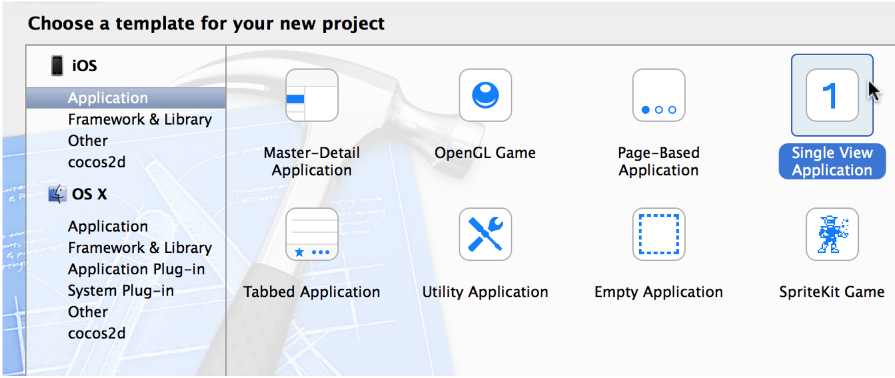


Figure : Create a new project

Add a new NSObject subclass of “UIViewController” to the project named “DetailViewController”

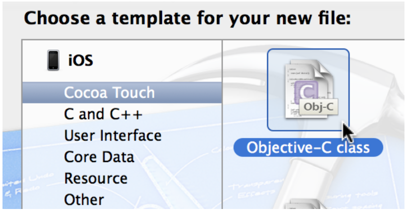
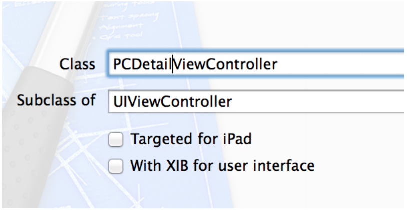


Figure : New Objective C class



Class PCDetailViewController

Subclass of UIViewController

Targeted for iPad

With XIB for user interface

Figure : Subclassing UIViewController

Add a “UIViewController” object from the Object library on to your storyboard.

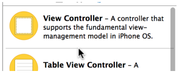

View Controller – A controller that supports the fundamental view–management model in iPhone OS.
Table View Controller – A

Figure : The UIViewController object in the Library

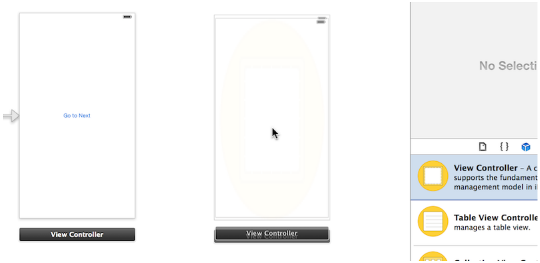
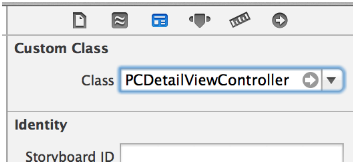

Go to Next
View Controller
View Controller
No Selecti
View Controller – A c supports the fundament management model in i
Table View Controlle manages a table view.

Figure : Drop UIViewController on Storyboard

Set the subclass of the UIViewController to “DetailViewController”.


Custom Class

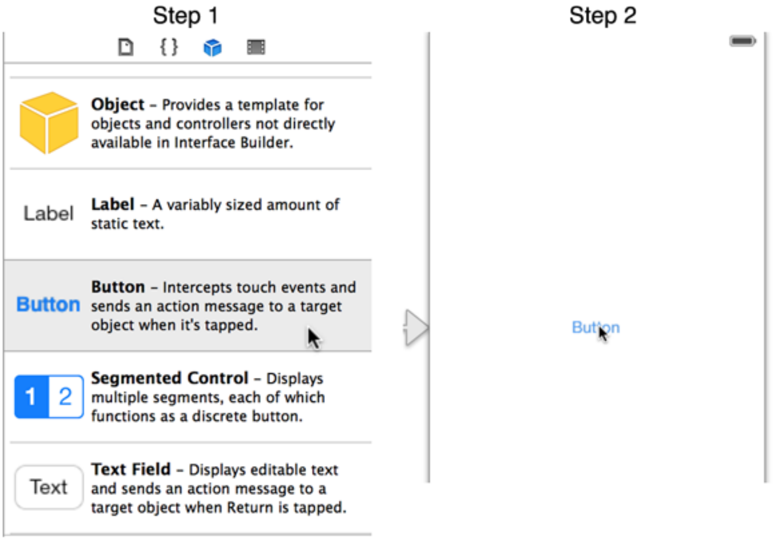
Class: PCDetailViewController

Identity

Storyboard ID:

Figure : Setting the UIViewController subclass

Double click on the first UIViewController in the stoyrboard and add a button with the text “Show Detail” to its scene.



Object – Provides a template for objects and controllers not directly available in Interface Builder.

Label **Label** – A variably sized amount of static text.

Button **Button** – Intercepts touch events and sends an action message to a target object when it's tapped.

1 2 **Segmented Control** – Displays multiple segments, each of which functions as a discrete button.

Text **Text Field** – Displays editable text and sends an action message to a target object when Return is tapped.

Figure : Drag a button to the scene

Select the button, and view its outlets in the inspector pane. Drag from the `action` outlet under `Triggered Segues` to the “Detail View Controller” scene in your storyboard. Choose “Modal” when asked what sort of segue style will be used.

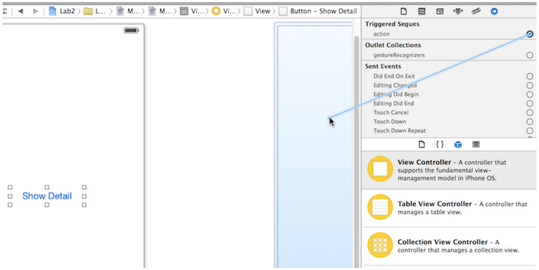

Lab2 › L... › M... › M... › Vi... › Vi... › View › Button - Show Detail
Show Detail
Triggered Segues
action
Outlet Collections
gestureRecognizers
Sent Events
Did End On Exit
Editing Changed
Editing Did Begin
Editing Did End
Touch Cancel
Touch Down
Touch Down Repeat
View Controller - A controller that supports the fundamental view-management model in iPhone OS.
Table View Controller - A controller that manages a table view.
Collection View Controller - A controller that manages a collection view.

Figure : Connect the action to the Detail View Controller

Double click on the Detail View Controller and add a button to its scene with the text “Close”.

Add the following code to `DetailViewController.h`

```
-(IBAction) dismissDetailView:(id) sender;
```

Add the following code to `DetailViewController.m`

```
-(IBAction) dismissDetailView:(id) sender {
  [self dismissViewControllerAnimated:YES completion:nil];
}
```

In your Storyboard, select the “Close” button you added to the Detail View Controller, and connect its “Touch Up Inside” outlet to the Detail View Controller. Select the “dismissDetailView:” selector when prompted which selector to connect to.

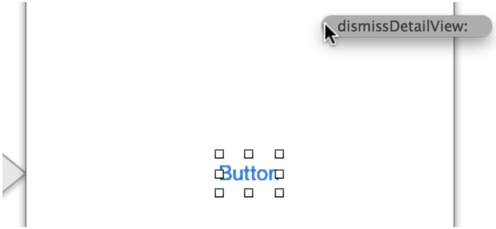
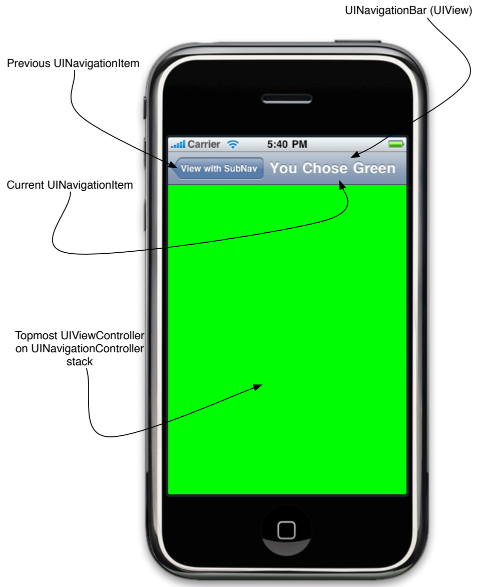


Figure : Connect the action to the dismiss selector

Run the project!

Figure : Yay, it works!

UINavigationController is a stack based manager of view controllers that the user can navigate through


UINavigationBar (UIView)
Previous UINavigationItem
Current UINavigationItem
Topmost UIViewController on UINavigationController stack
Carrier
5:40 PM
View with SubNav
You Chose Green

Every UIViewController managed by a UINavigationController has a reference to that UINavigationController

```
MyViewController * controller = [[MyViewController alloc] initWithNibName:nil
                                                          bundle:nil];
[[self navigationController] pushViewController:controller animated:YES];
[controller release];
```

UITabBarController swaps in UIViewControllers assigned to it when users click on the associated button

Each UIViewController has a tabBarItem property that UITabBarController uses to populate its UITabBar

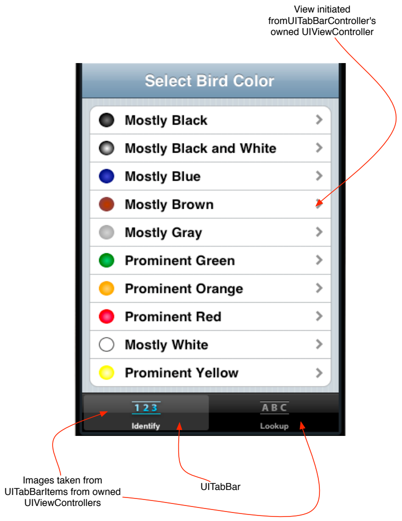


Figure : UITabBarController example

UITableViewController instructs a UITableView about the number of rows it contains, what to do when clicked, the number of sections, and other information

UITableViewController is just a convenience class that saves you from manually conforming an object to the UITableViewDataSource and UITableViewDelegate protocols

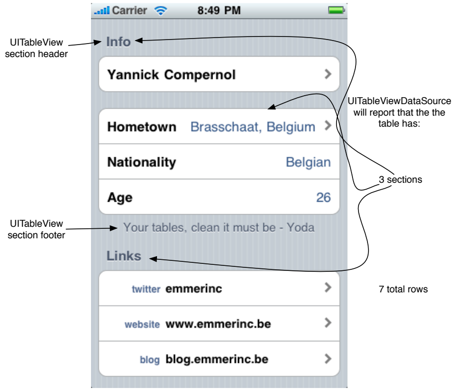

UITableView section header
UITableView section footer
UITableViewDataSource will report that the the table has:
3 sections
7 total rows
Carrier 8:49 PM
Info
Yannick Compernol
Hometown Brasschaat, Belgium
Nationality Belgian
Age 26
Your tables, clean it must be - Yoda
Links
twitter emmerinc
website www.emmerinc.be
blog blog.emmerinc.be

Figure : UITableView example

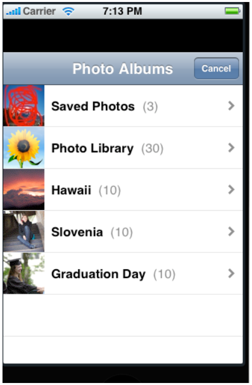
Add a UITableViewController to the project

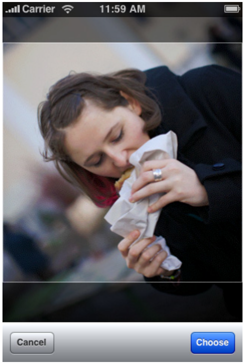
Add an NSArray containing names from the class to the controller

Change the Interface builder reference to the UITableViewController

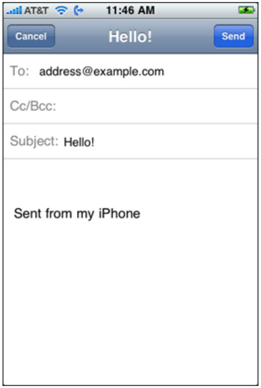
Not all view controllers fit within the “display in UINavigationController” or “display in UITabBarController” model.

UIImagePickerController





MFMailComposeViewController



To: address@example.com

Cc/Bcc:

Subject: Hello!

Sent from my iPhone

You can present a view controller modally by using the `presentModalViewController:animated:` message

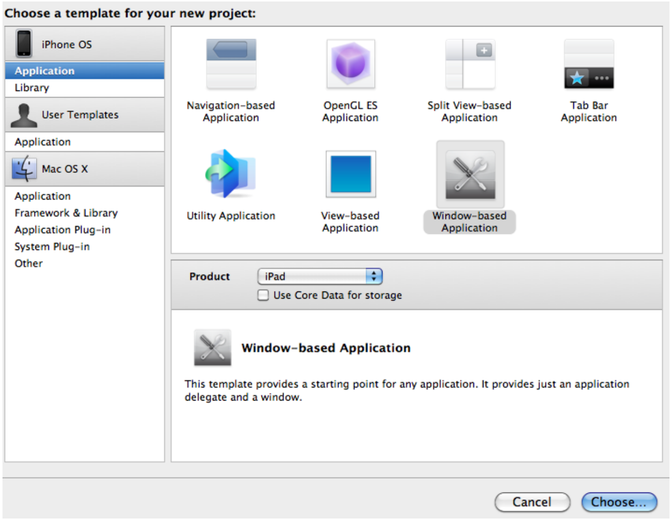
```
-(IBAction) buttonClicked:(id) sender {
  MyViewController * mvc = [[MyViewController alloc]
                               initWithNibName:@"TheNib"
                               bundle:nil];
  [self presentModalViewController:mvc animated:YES];
  [mvc release];
}
```

You can specify the transition style with which the view controller appears via the UIViewController message
`setModalTransitionStyle:(UIModalTransitionStyle)style`

1. XCode and Interface Builder

XCode and Interface Builder combined provide a development, debugging and interface development tool for you to develop your iPhone apps. It can integrate with SVN, CVS or Perforce (and soon, Git!), it can run automated tests on your code, provide inline documentation assist, code completion and a lot more. It also provides a developer level interface to all the devices you will be debugging your code on; you can manage devices and device profiles through XCode as necessary.

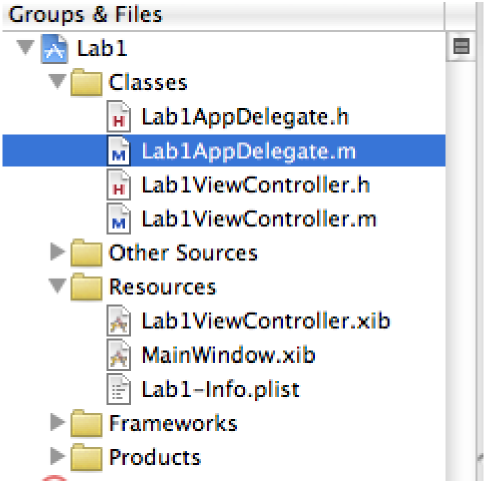
XCode provides a number of application templates for starting an app. Most basic types of applications can get a fast start by using one of these app templates.



This template provides a starting point for any application. It provides just an application delegate and a window.

Figure : XCode New Project window

Once you've selected the type of app you want to create, your project will be created with an application delegate and usually at least one view controller that will manage what goes on the screen. It will also have a .xib file for your application, and for each view controller in the app, that is used by Interface Builder to layout the screen contents.



Double clicking on a .xib file will open it in Interface Builder, where you can drag components from the Interface Builder toolbox into the view. You can easily change the orientation of objects.

Help - Developer Documentation is an invaluable resource for researching Cocoa internals.

Learn hotkeys! (CMD+SHIFT+D and CMD+0 will buy you hours)

Make XCode fit you (Single window interface, XCode themes, application templates)

XCode plugins (Code Pilot
`http://mac.brothersoft.com/code-pilot-for-xcode.html`,
Accessorizer
`http://www.kevincallahan.org/software/accessorizer.html`)