https://github.com/GroundSpeed/BreakingGround

# BREAKING GROUND WITH IOS

August 24, 2014
1:30p - 5:00p

Don Miller
Founder of GroundSpeed™™
Co-founder of Seed Coworking
@donmiller

# SCHEDULE

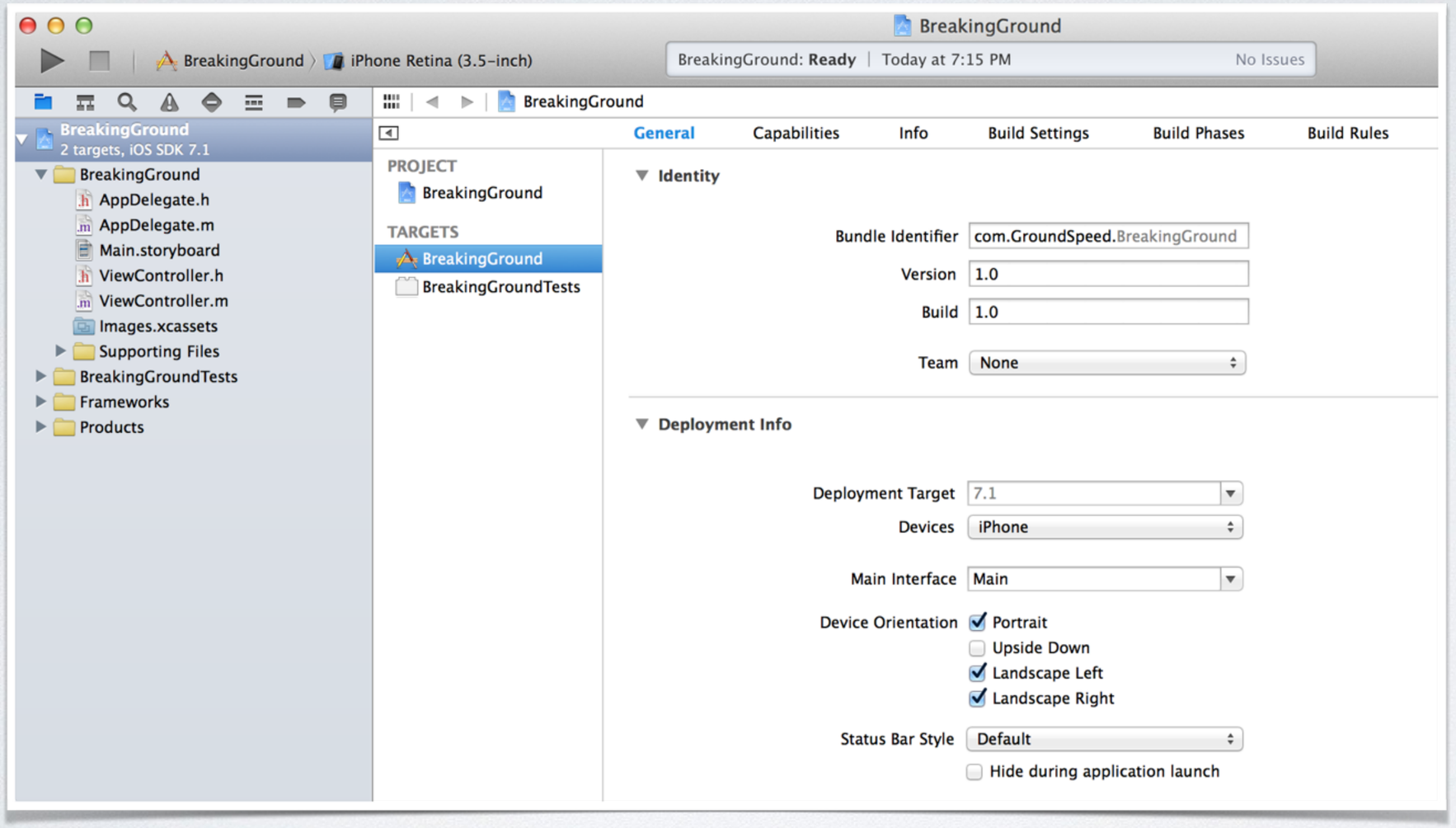
- Introductions
- Discuss Terminology, Xcode Overview, Hello World
- Break
- Command Line and GitHub
- NSDictionary, NSArray, and Objective-C
- Storyboards
- UITableViews (if we have time)

GroundSpeed™
rapid web + mobile software

# TERMINOLOGY

- iOS
- iPhone
- iPad
- iPod
- MacBook
- iMac
- SSD
- SDK
- IDE
- Xcode
- OSX
- Mavericks
- Mountain Lion
- Lion
- Snow Leopard, etc.
- Retina

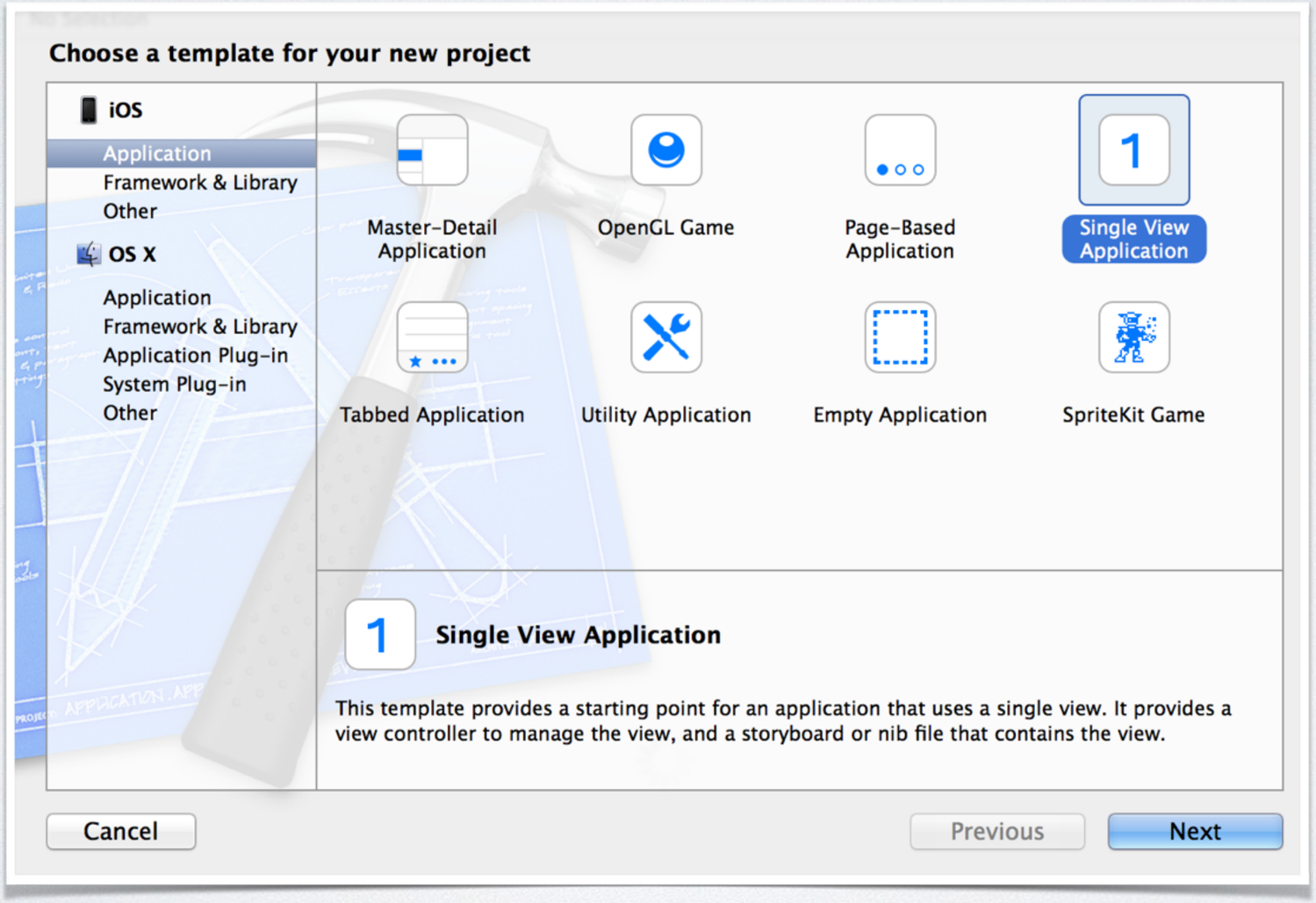
# XCODE 5.1.1 & IOS 7.1



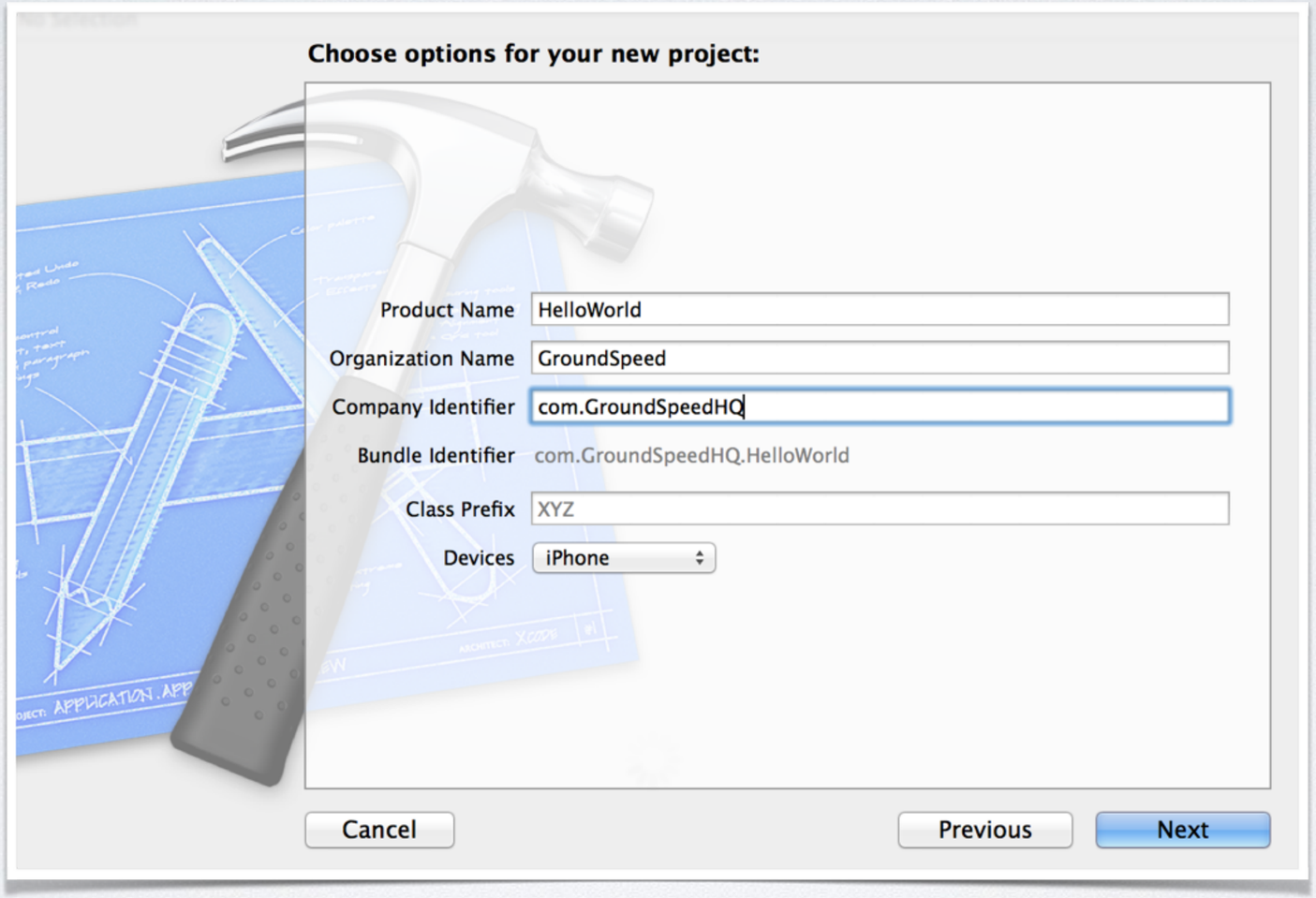

GroundSpeed™
rapid web + mobile software

# HELLO WORLD!

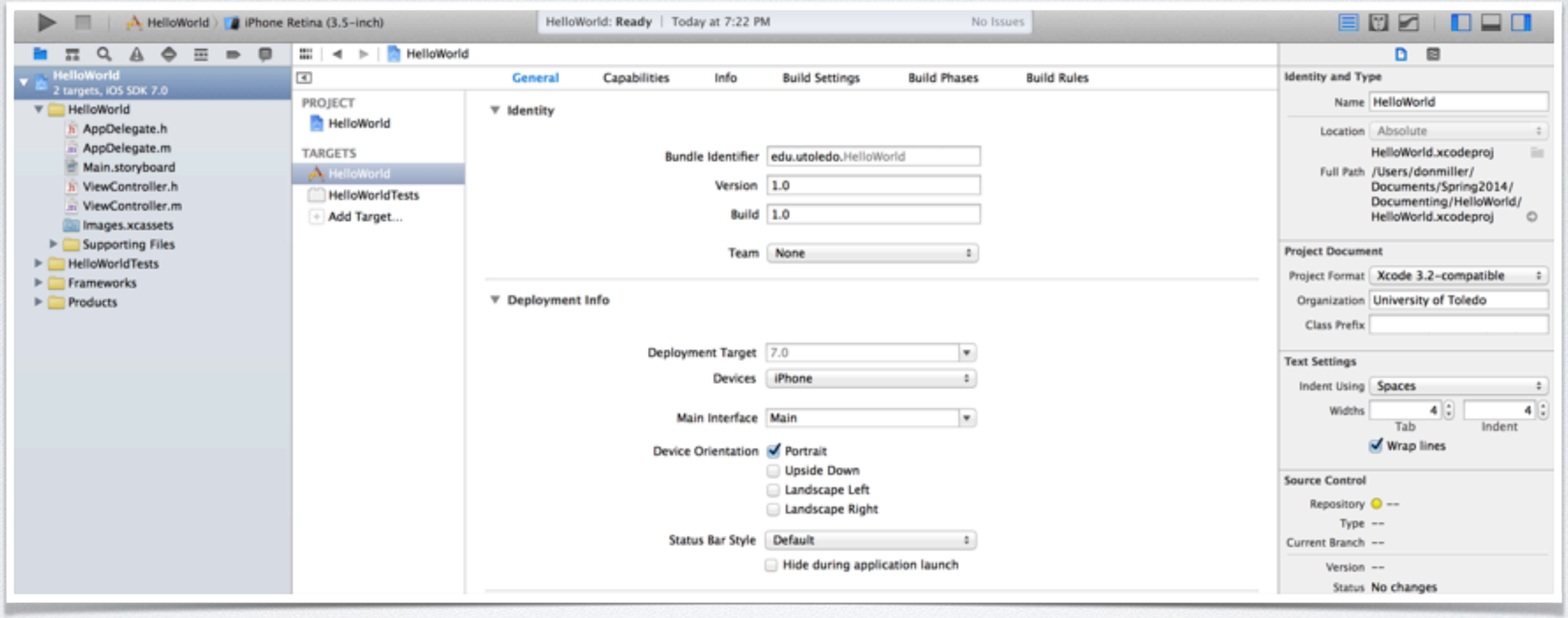
# CREATE A NEW SINGLE VIEW APPLICATION



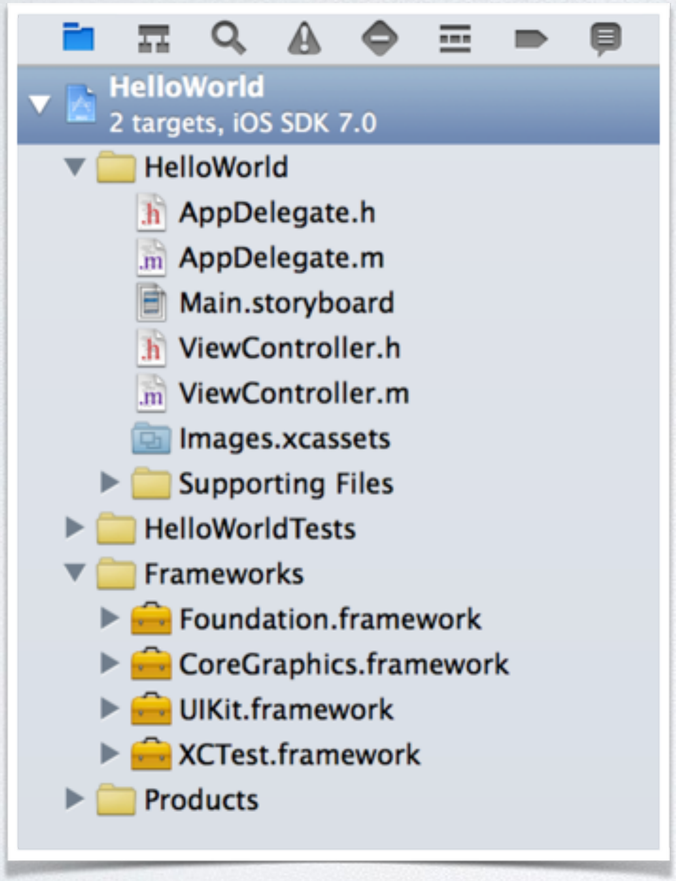
# ENTER THE PRODUCT NAME



# UPDATE THE GENERAL PROJECT ATTRIBUTES

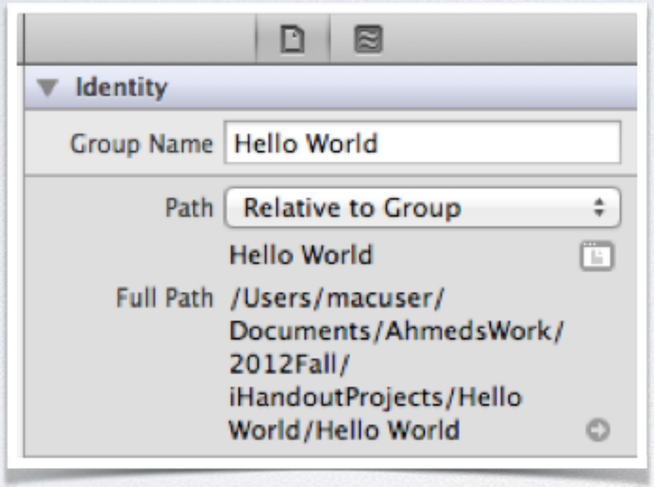


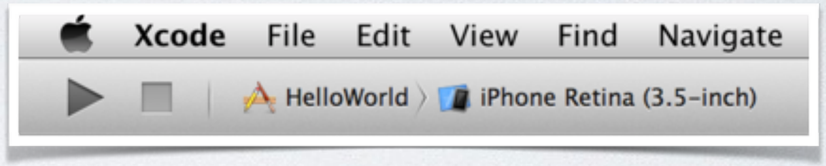
# PROJECT STRUCTURE




- Folders do not match physical layout on file system.
- Actually you will not find any sub folder named like Supporting Files in your physical project folder.
- The folder hierarchy in the XCode's navigator is simply a logical grouping of the resources.
- At this stage, we would be interested only on the Main.storyboard file.

# FILE INSPECTOR & SCHEMA SETTINGS

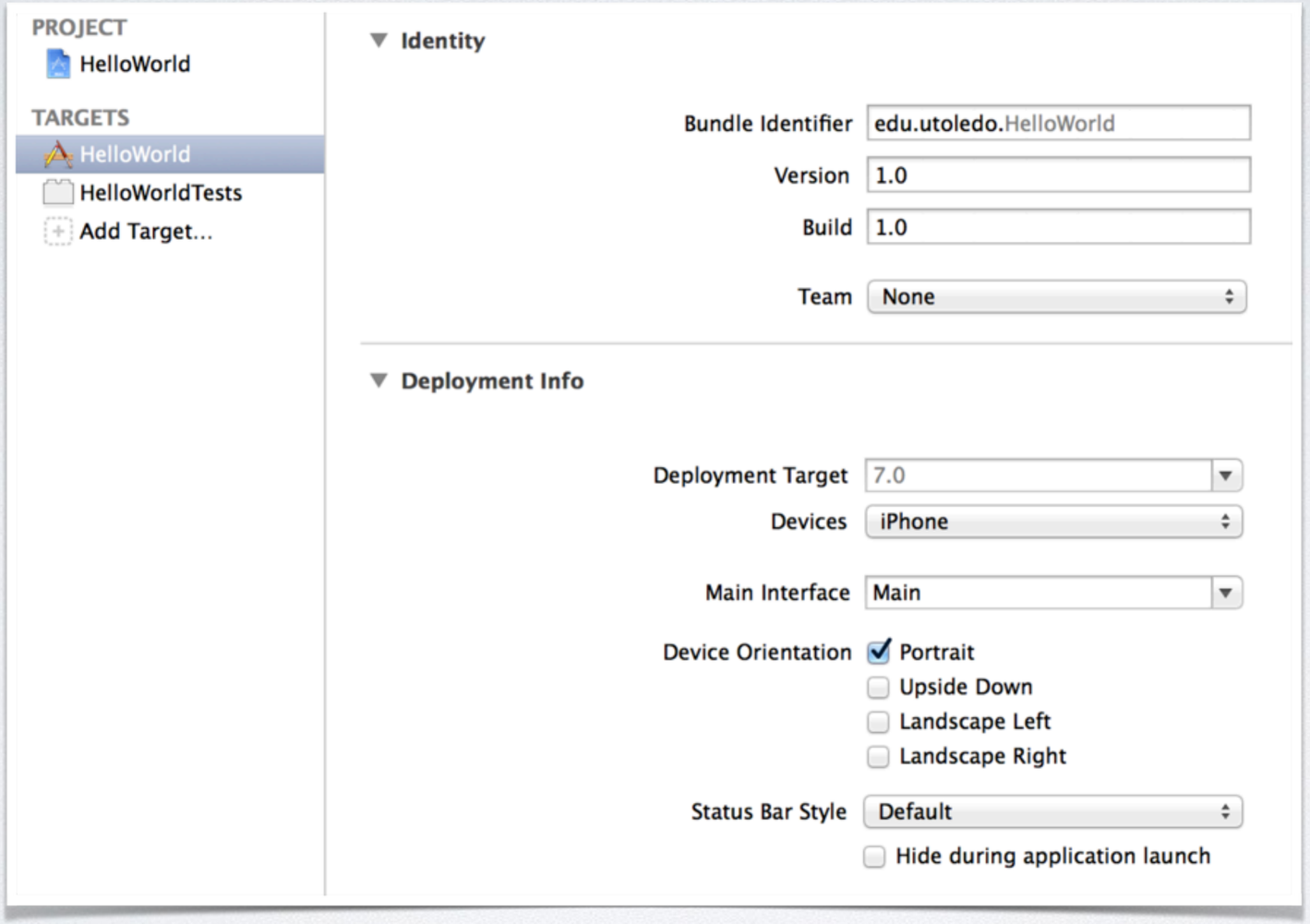


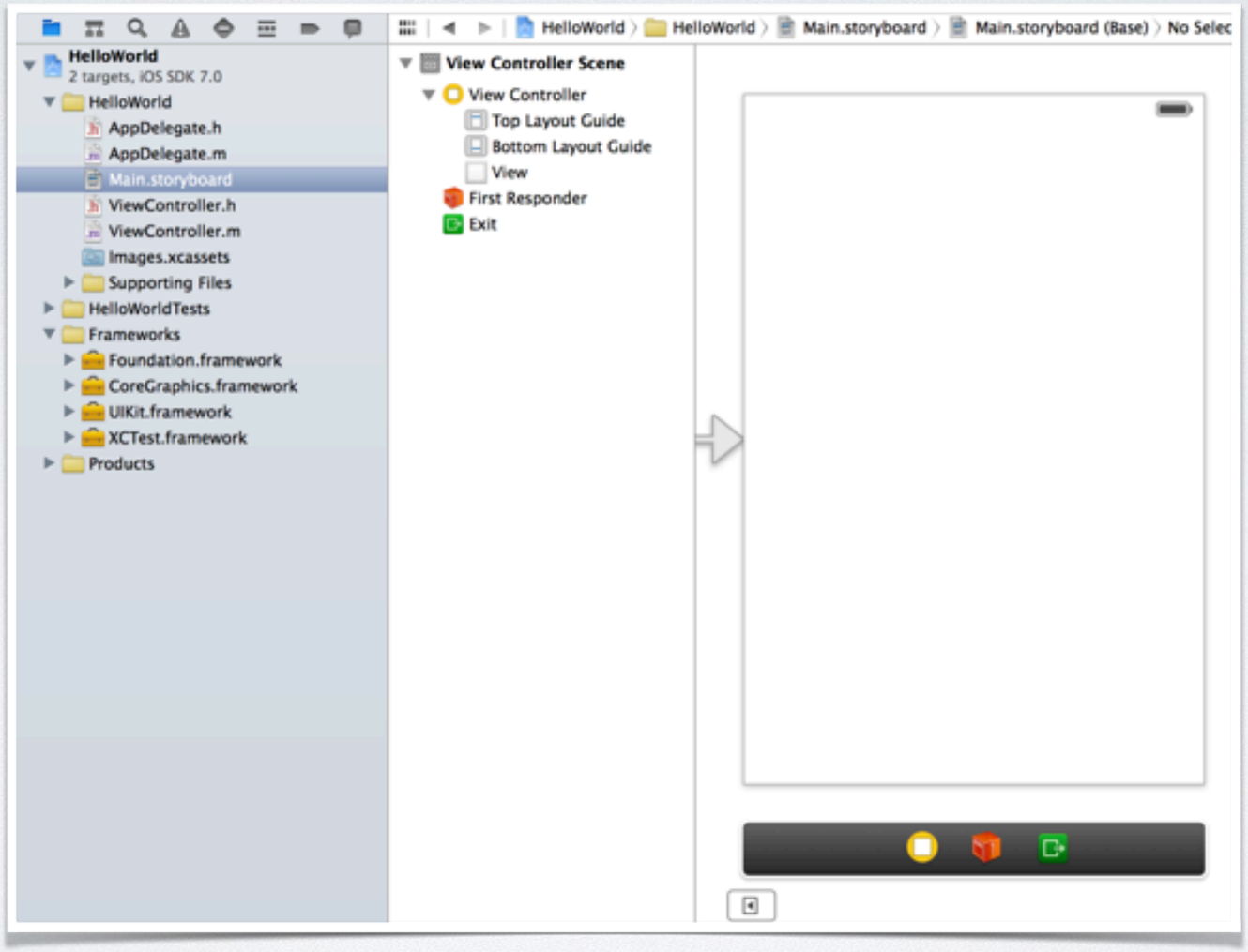


GroundSpeed™
rapid web + mobile software

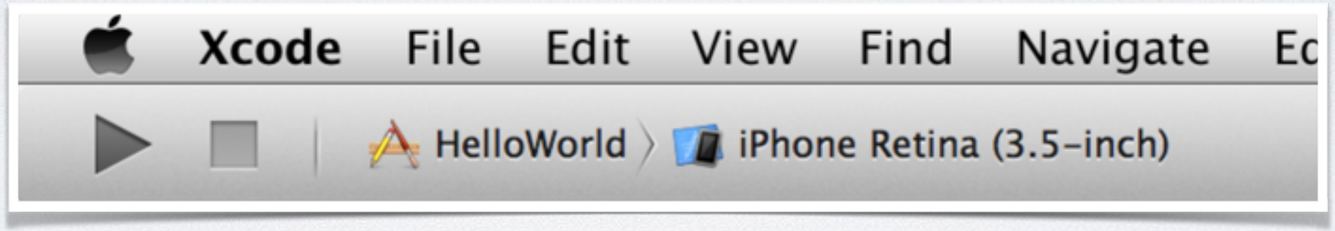
# DEVICES & TARGETS



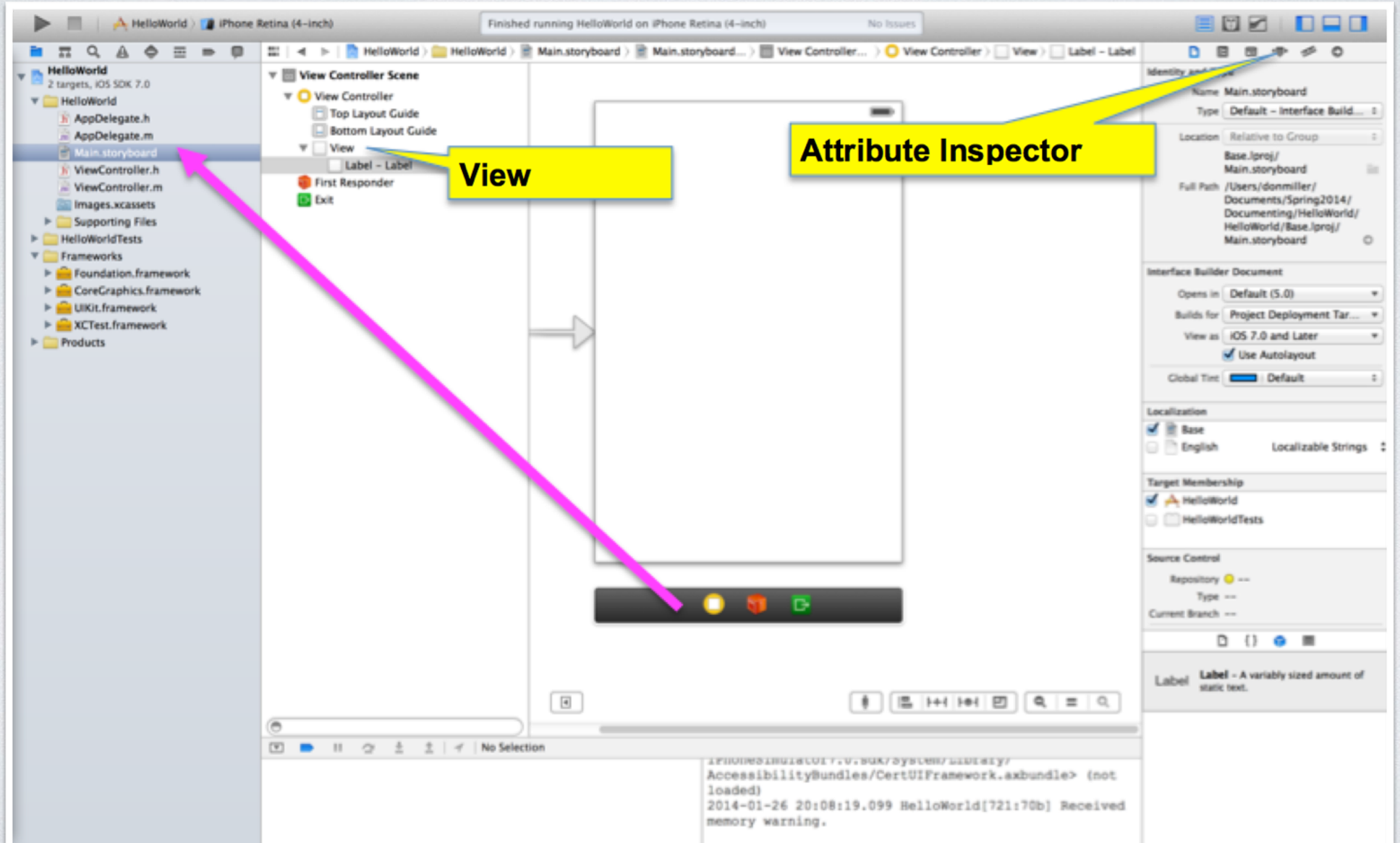
# FILE OWNER, VIEW, & FIRST RESPONDER



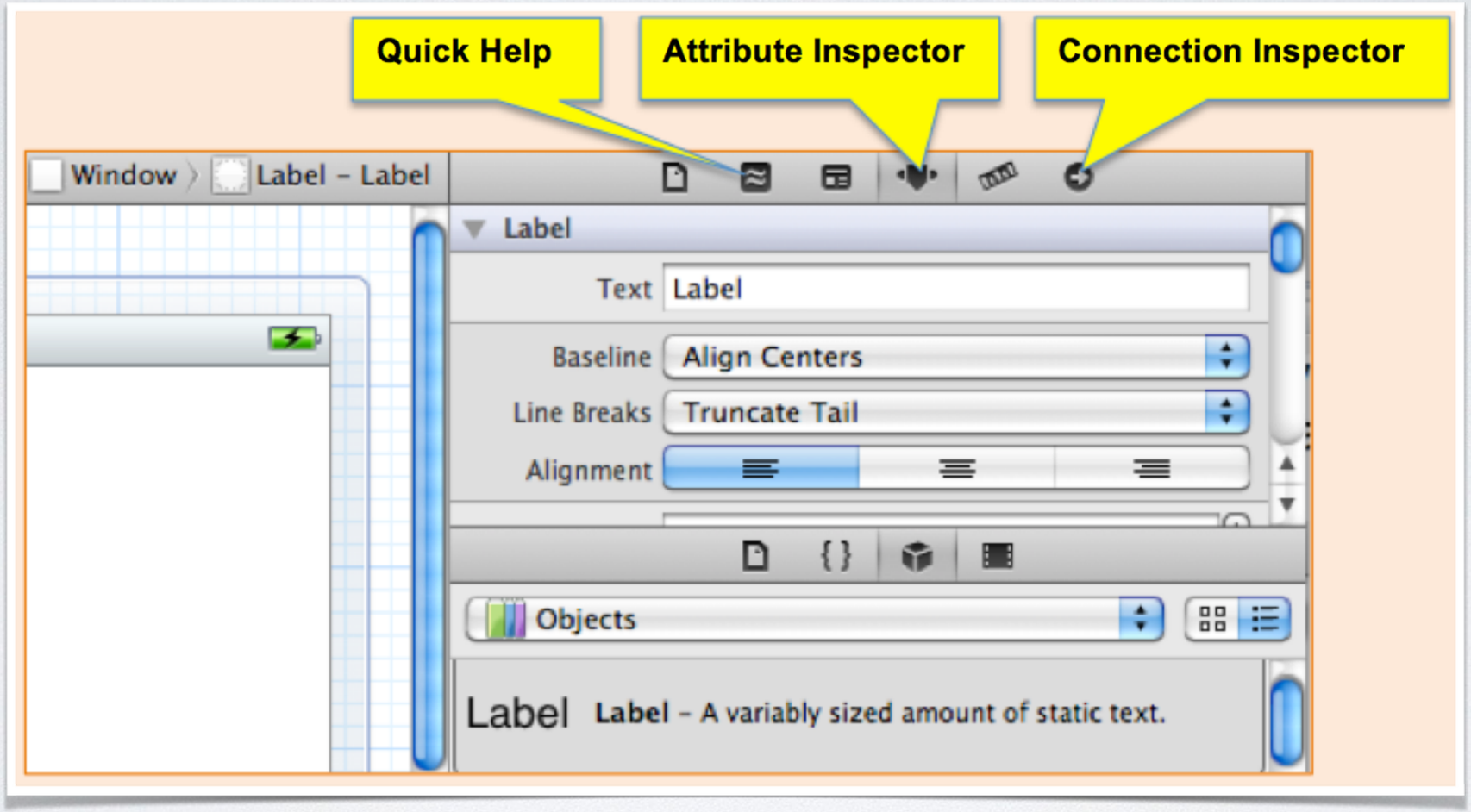
# RUN THE APPLICATION BY PRESSING PLAY



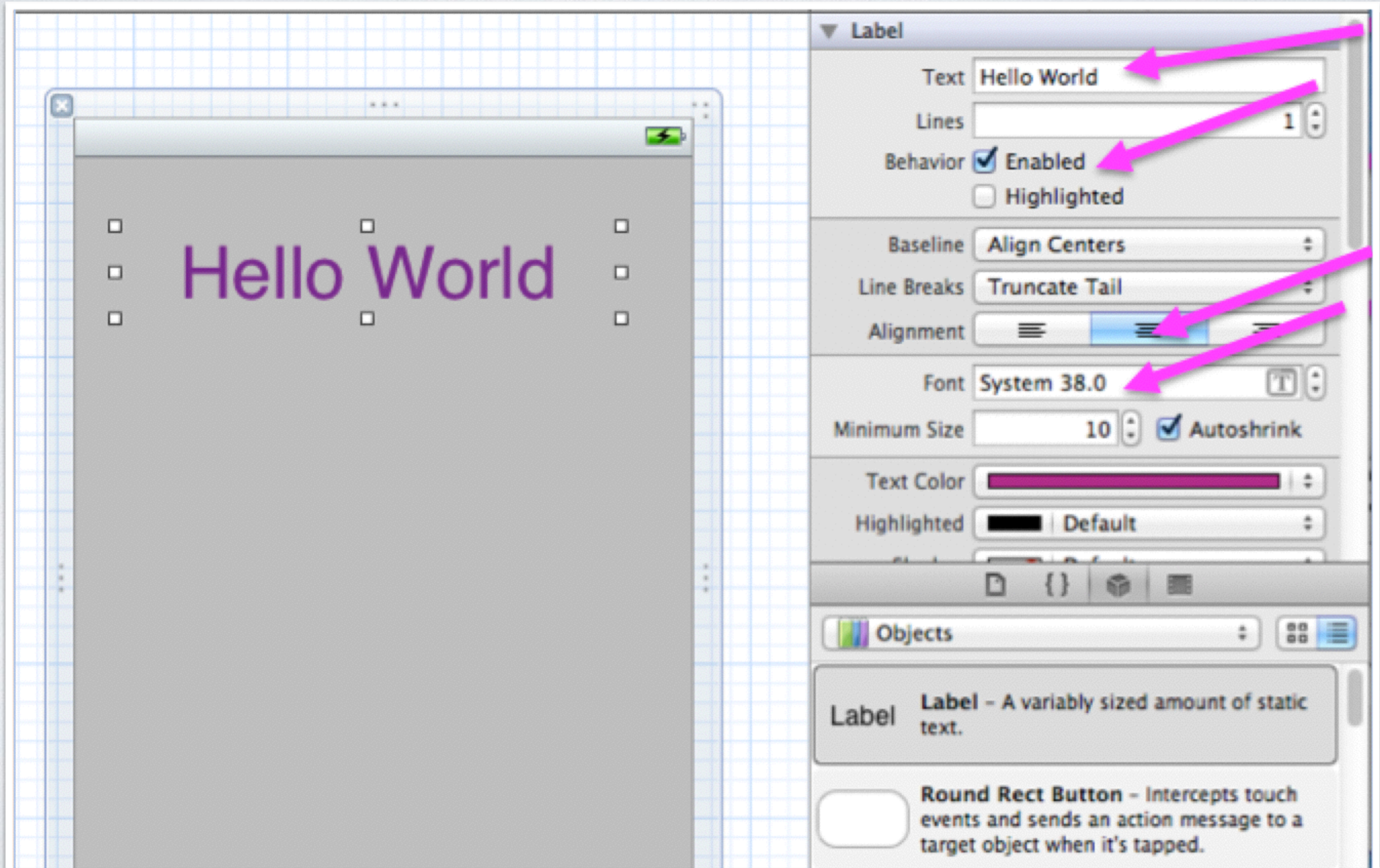
# ATTRIBUTE INSPECTOR



# INSPECTORS



# CREATE "HELLO WORLD!" LABEL

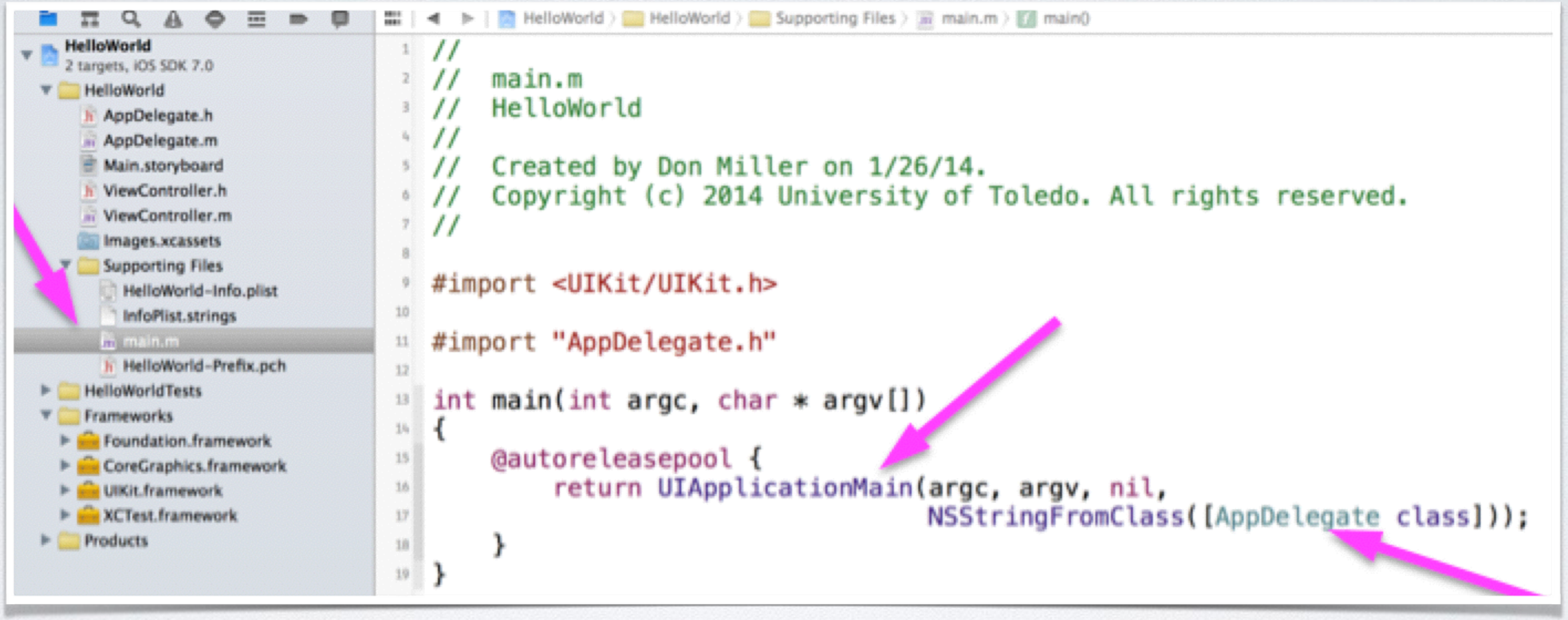


# Hello World! Demo

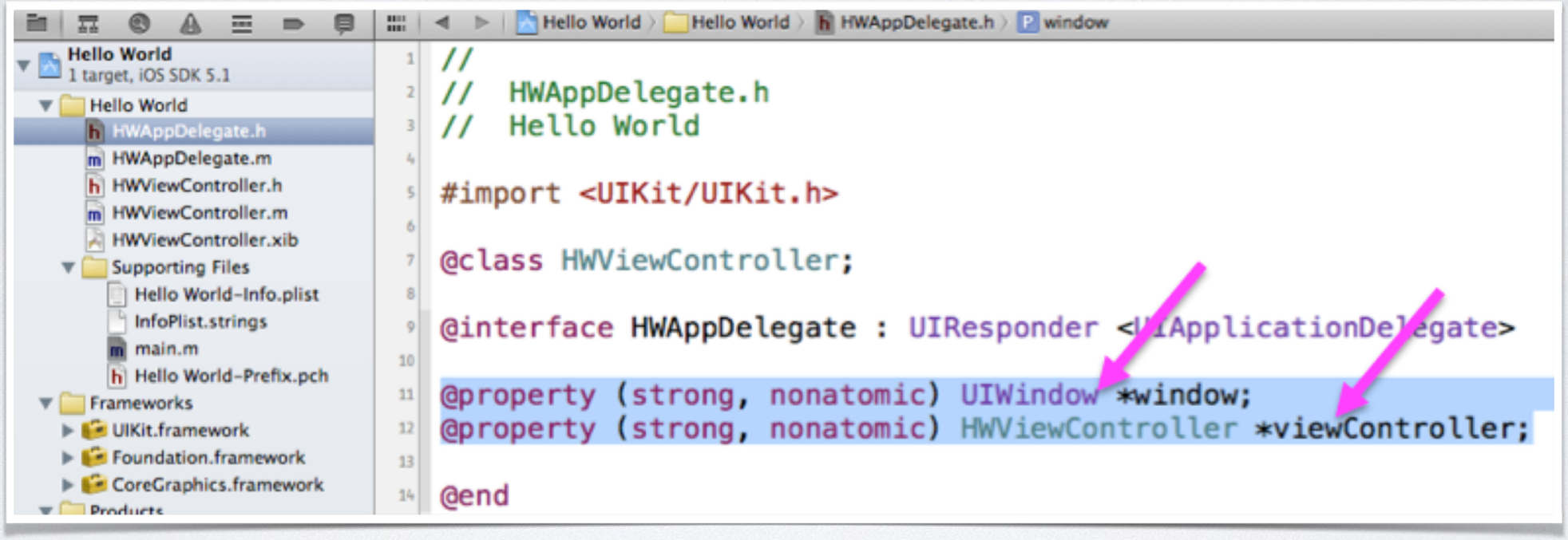
GroundSpeed™

rapid web + mobile software

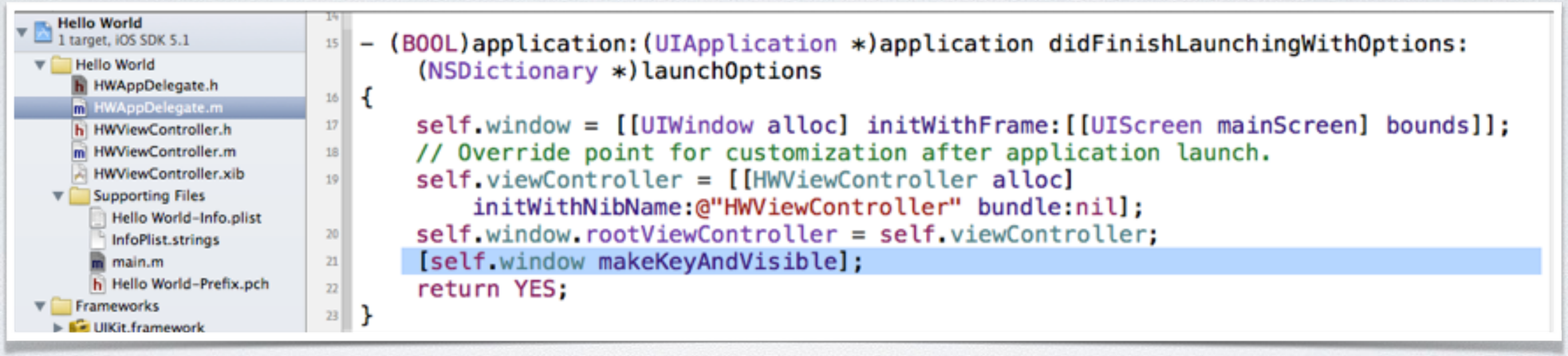
# Q. HOW DOES THE SYSTEM KNOW THAT IT NEED TO RUN THE VIEW VIA THE MAIN WINDOW?



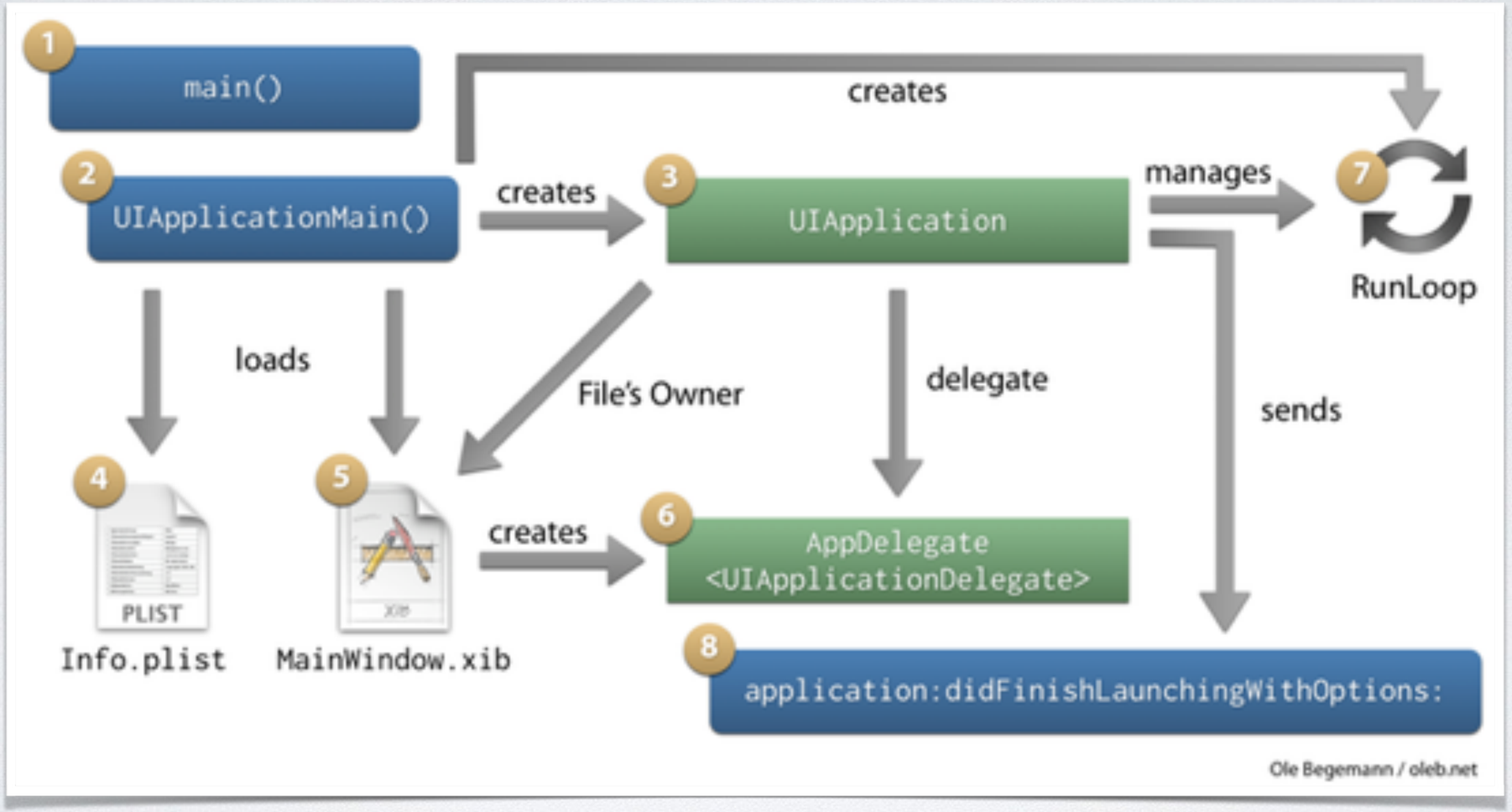
# APPDELEGATE.H



# APPDELEGATE.M

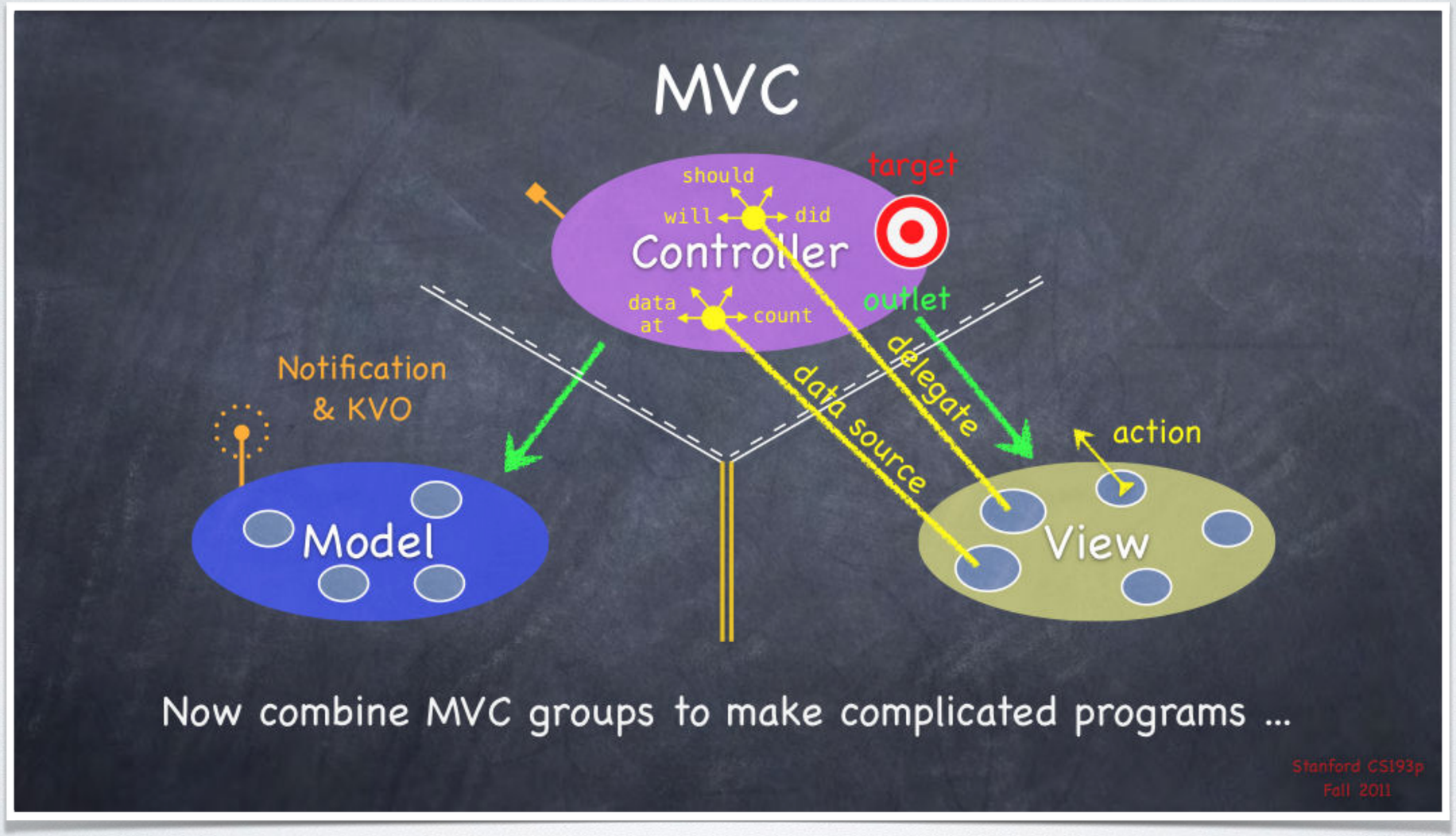


# IOS APPLICATION RUNTIME CYCLE



# MODEL - VIEW - CONTROLLER

Stanford iTunes University 2011



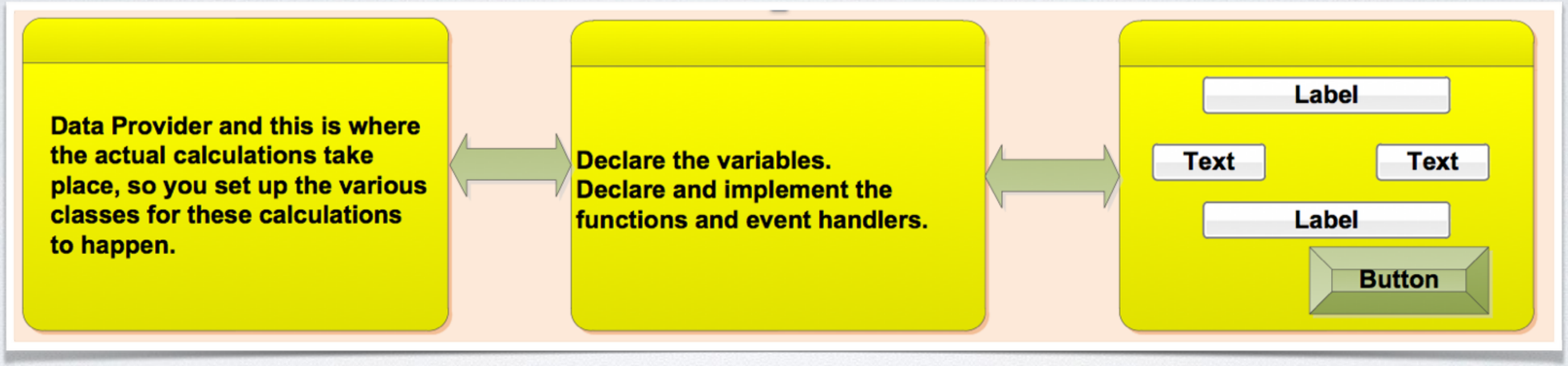
# MODEL - VIEW - CONTROLLER

- **The Model**: The M in MVC stands for the Model:  For the Model, we will need to develop an independent class (or a set of classes) that will serve the data requirements and standard business procedures.

- **The View**: In this context the View is the UI screen's layout (the canvas). Here, we configure many User Interface items (also known as UI controls on the screen).

- **The Controller**: This is a class that connects the view with the model. In its simplest form, it contains the name of the UIControls (that would be used in the app), and it also includes the signatures and the detail implementations of event handlers.

# MODEL - VIEW - CONTROLLER

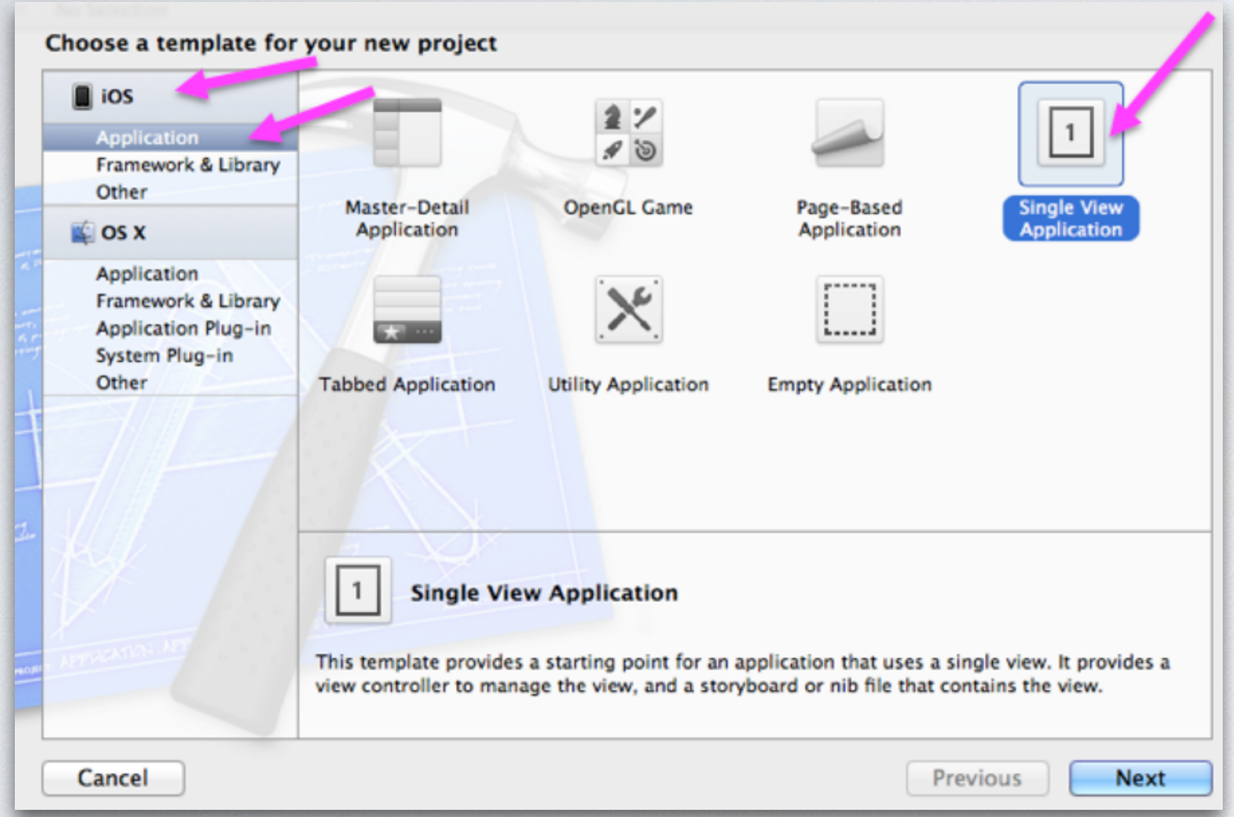


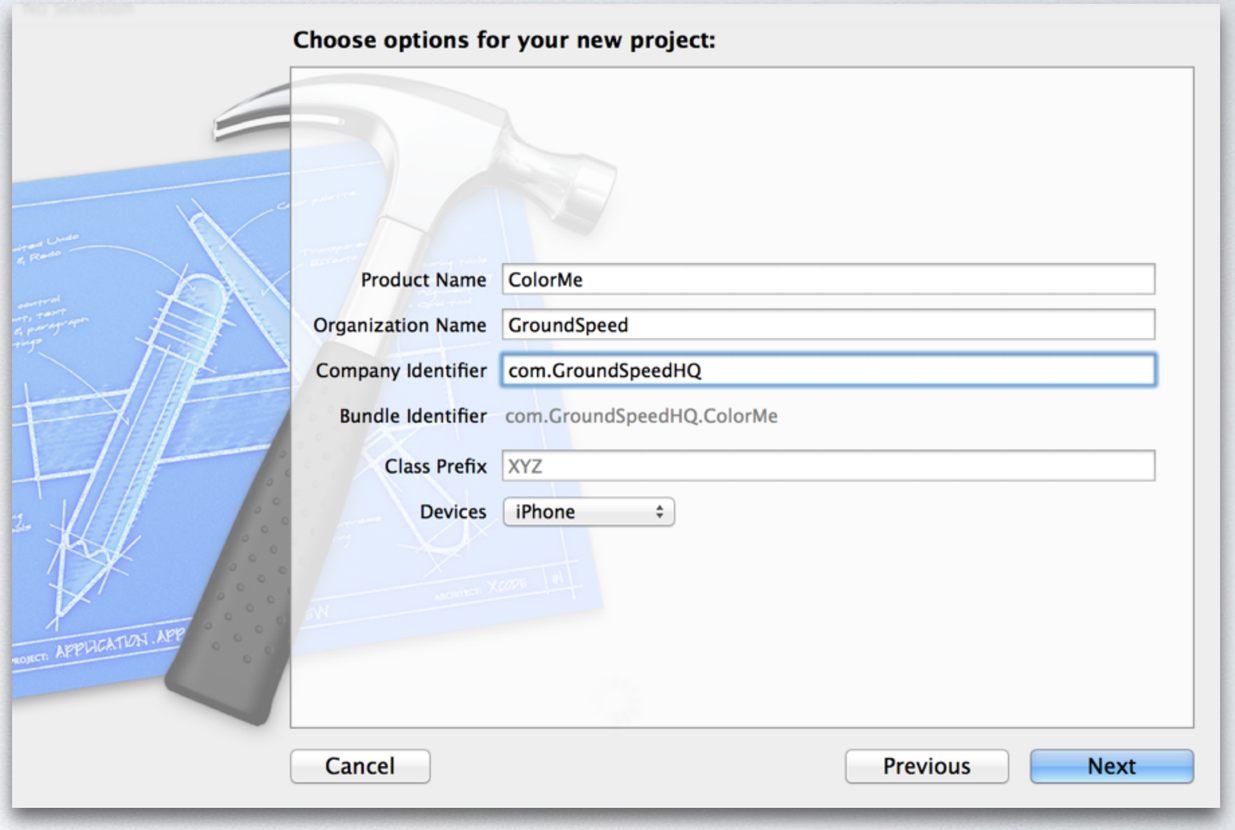
# Break Time

GroundSpeed™
rapid web + mobile software

# ANOTHER QUICK DEMO - [COLOR ME]

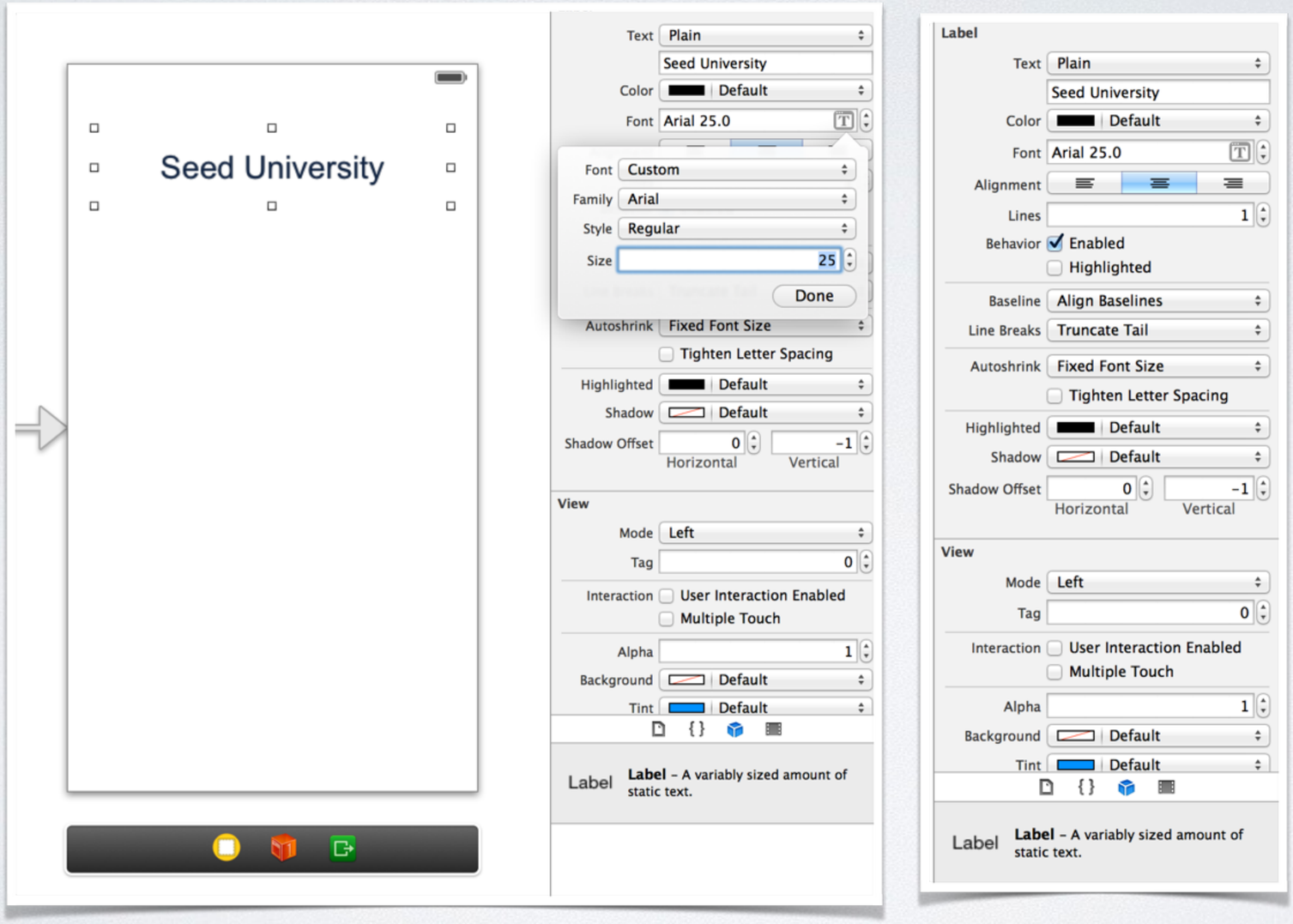
# CREATE A SINGLE VIEW APPLICATION



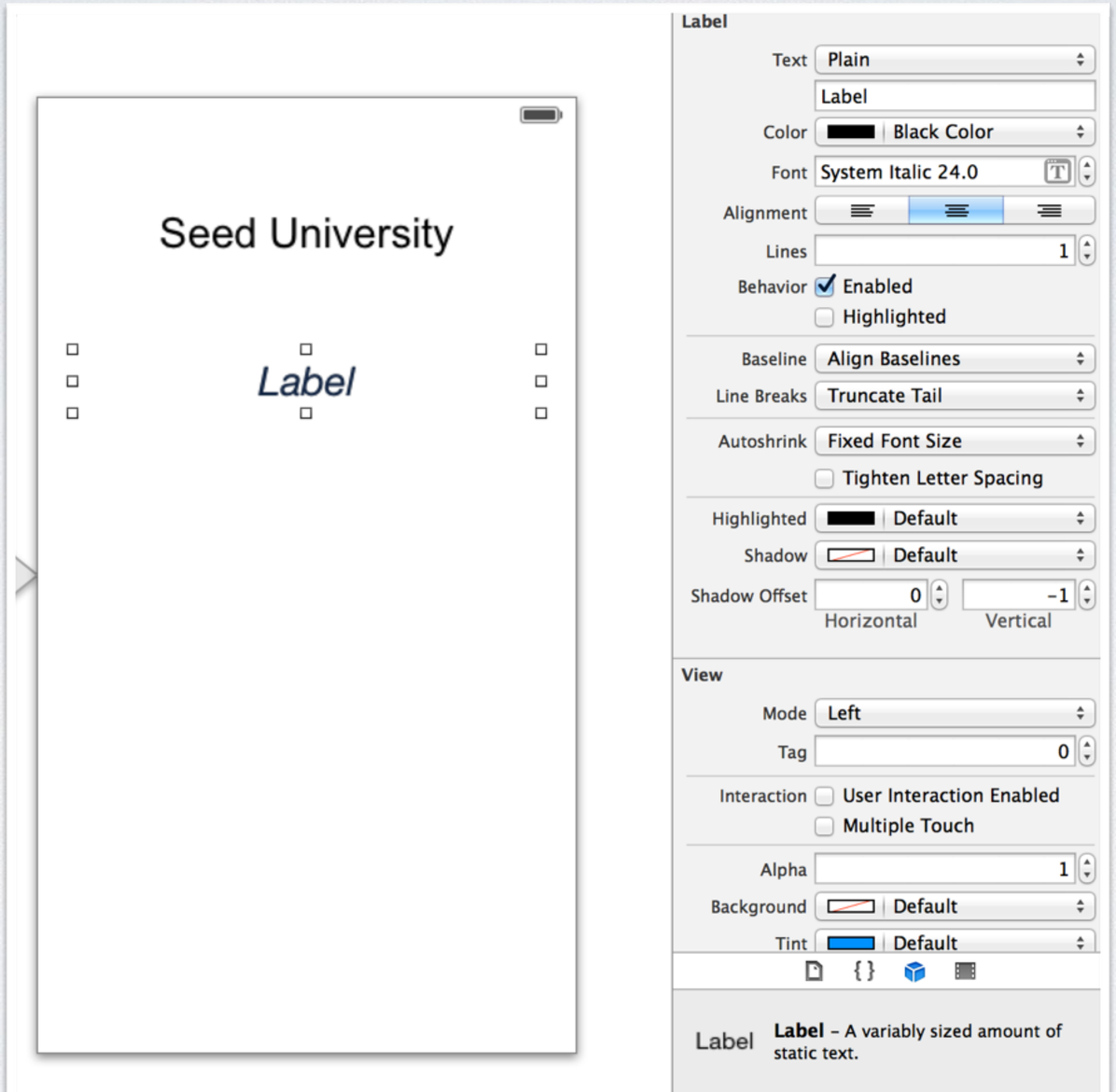

GroundSpeed™
rapid web + mobile software

# DRAG A LABEL TO THE STORYBOARD



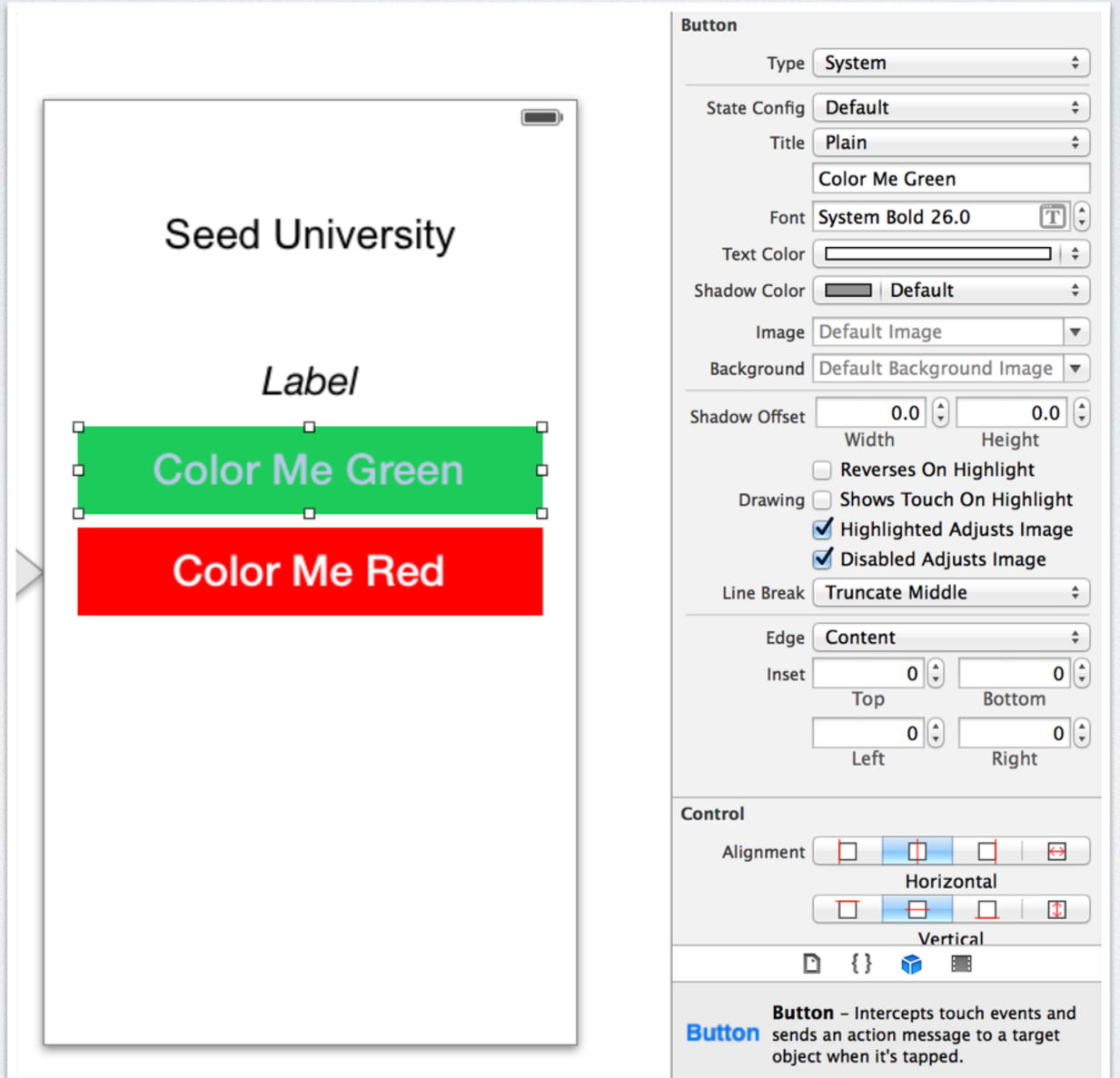

GroundSpeed™
rapid web + mobile software

# CREATE A BLANK LABEL FOR OUTPUT




GroundSpeed™
rapid web + mobile software

# CREATE TWO BUTTONS

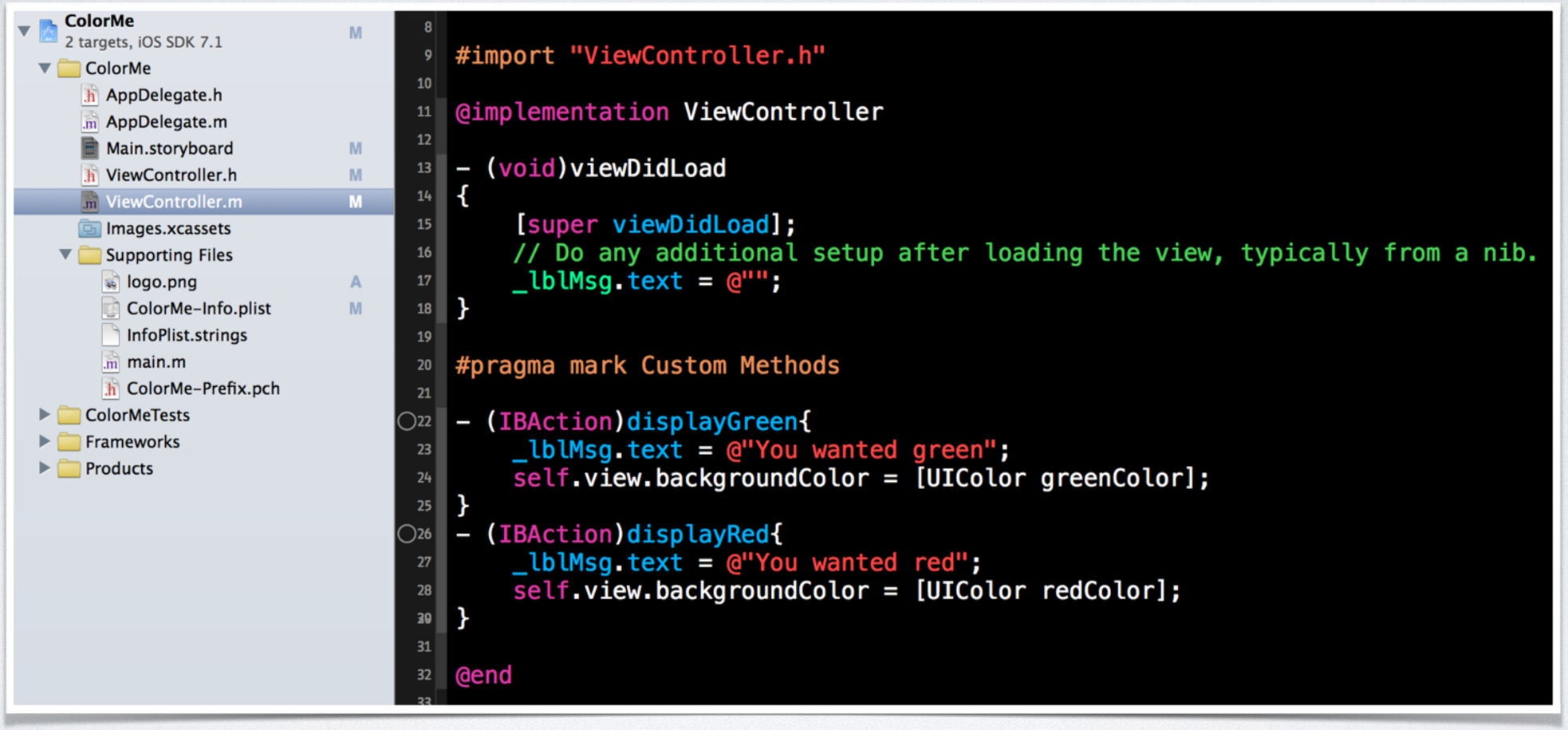


# UPDATE THE VIEWCONTROLLER.H FILE


Link to Gist

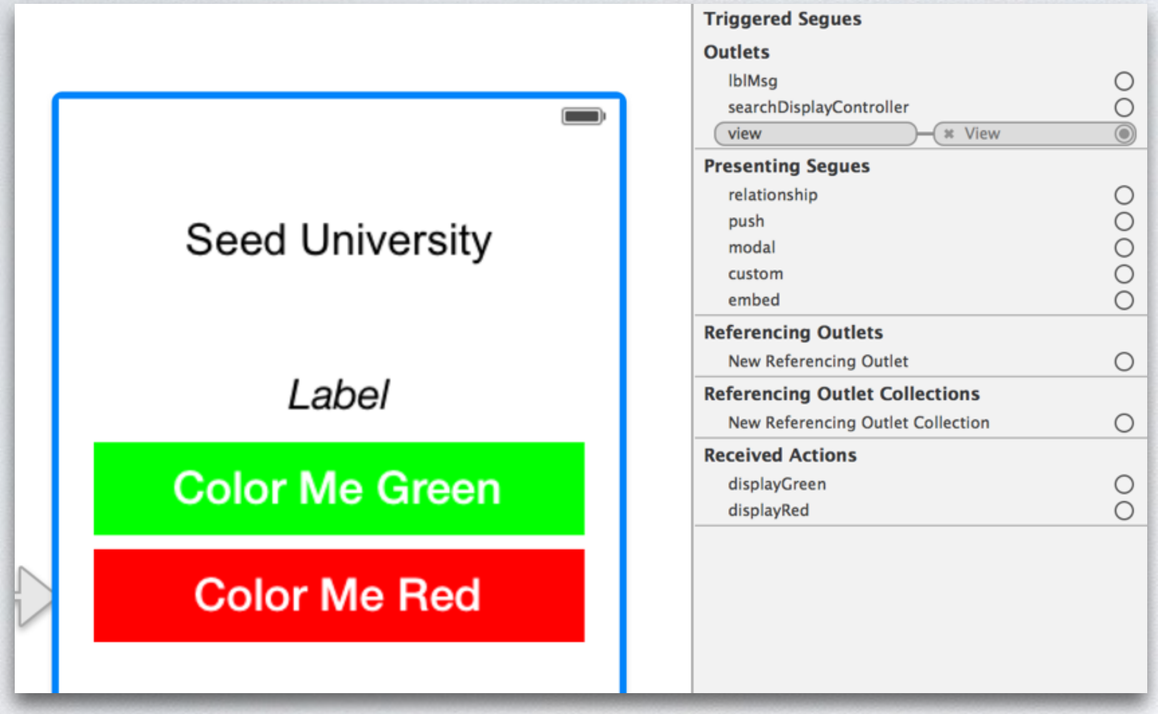
GroundSpeed™
rapid web + mobile software

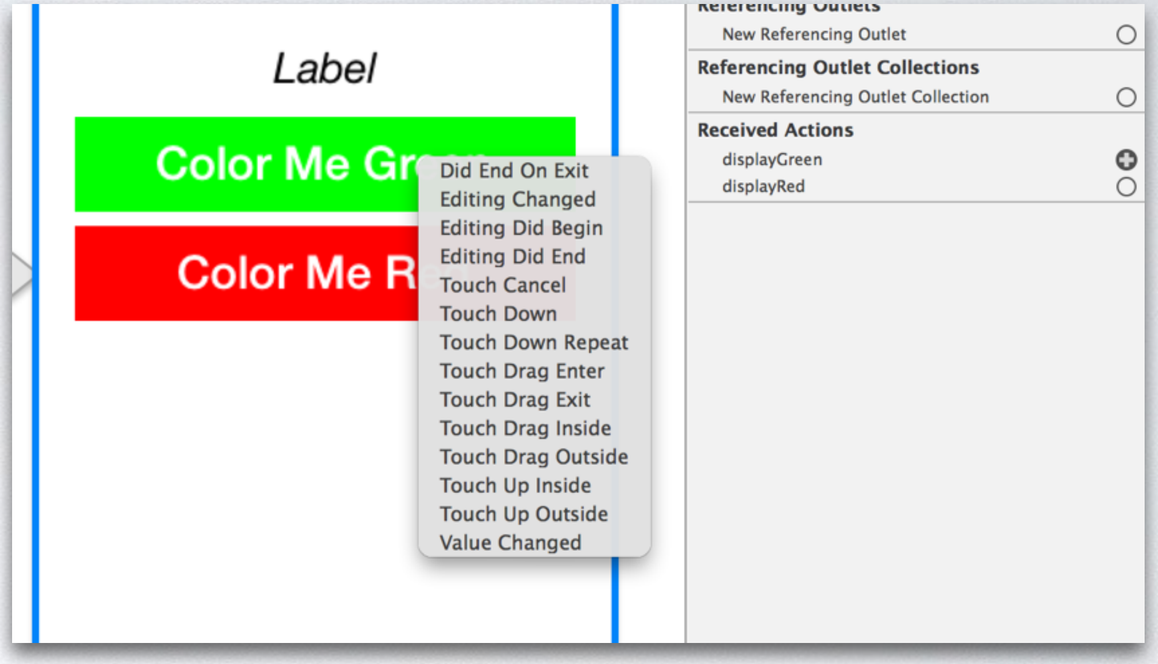
# UPDATE THE VIEWCONTROLLER.M FILE




Link to Gist

# CONNECTING THE OUTLETS AND ACTIONS

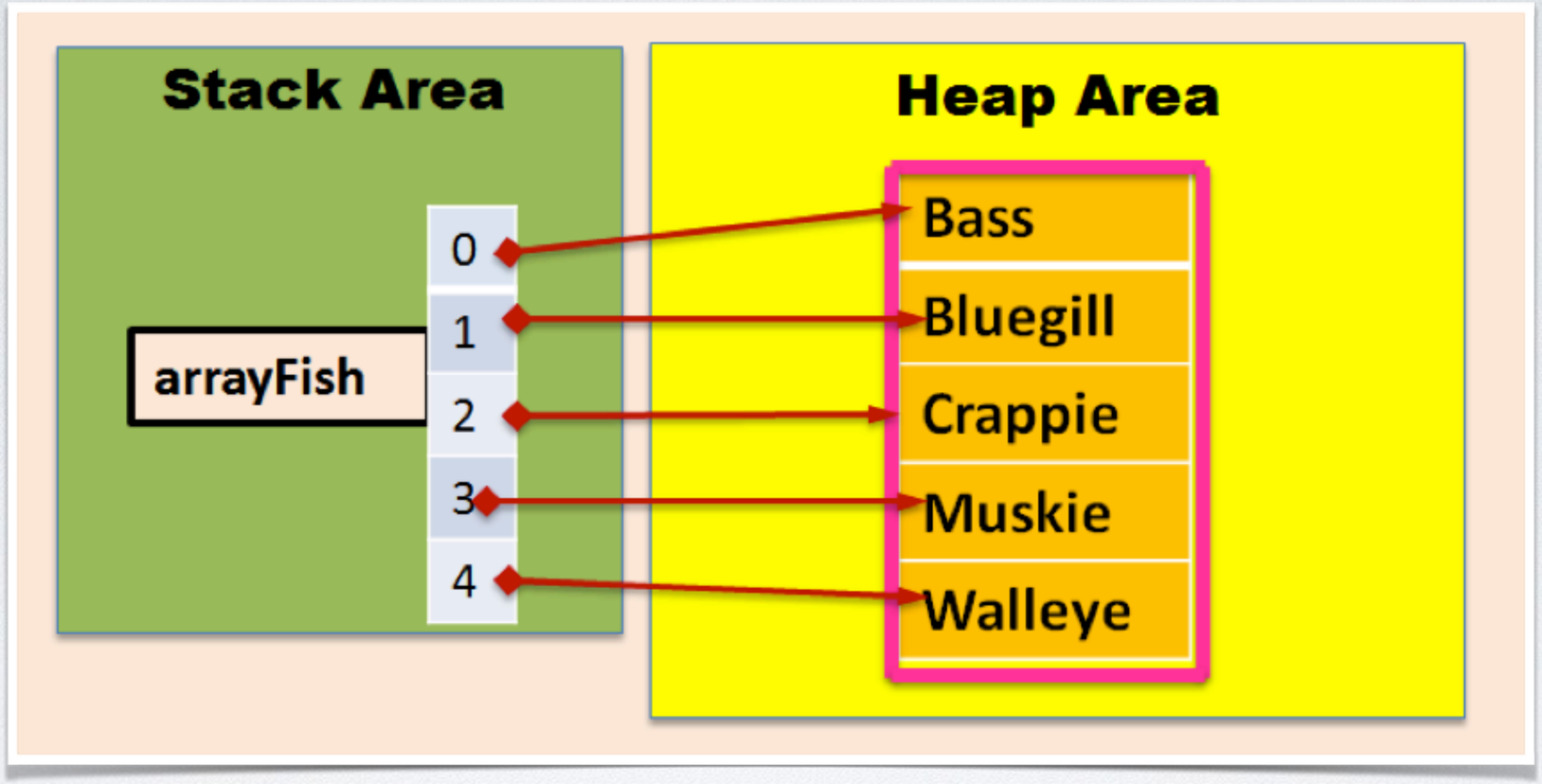


# Color Me Demo

GroundSpeed™
rapid web + mobile software

# ARRAYS

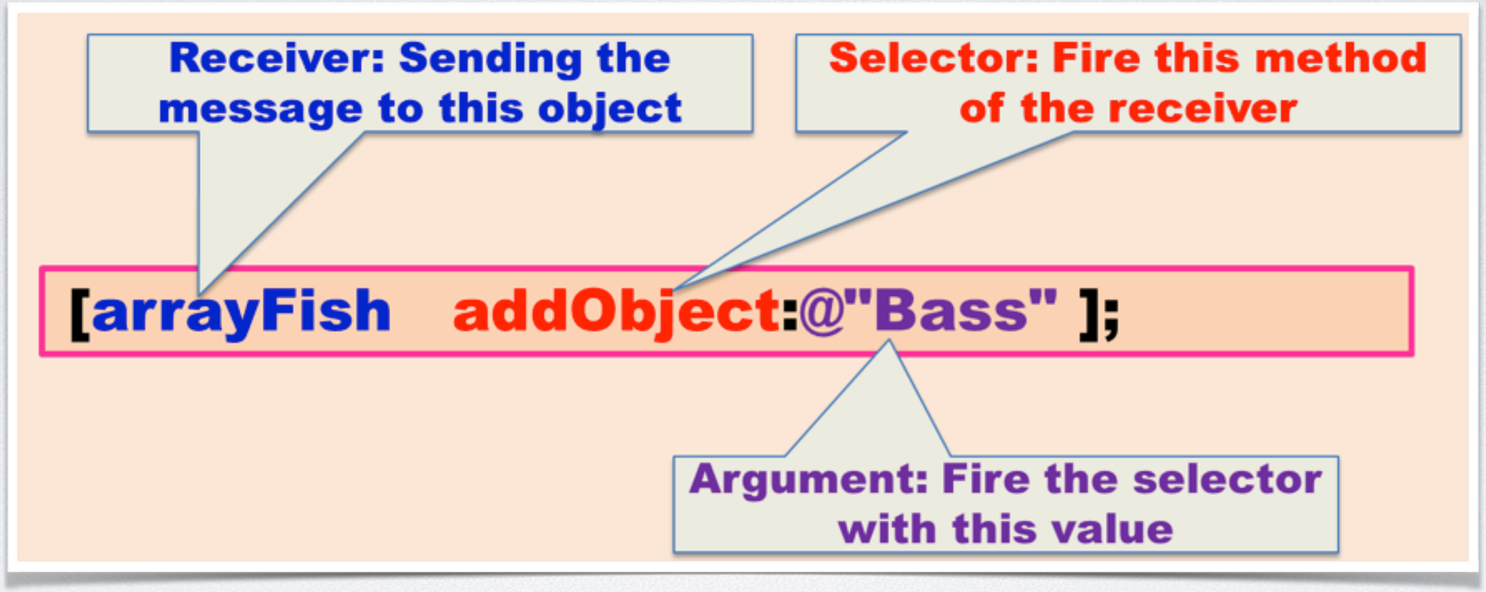



- [arrayFish count] will return the integer 5.
- [arrayFish objectAtIndex:3] will return the string Muskie
- [arrayFish objectAtIndex:5] will raise an Error

# CREATING AN ARRAY WITH OBJECTIVE-C

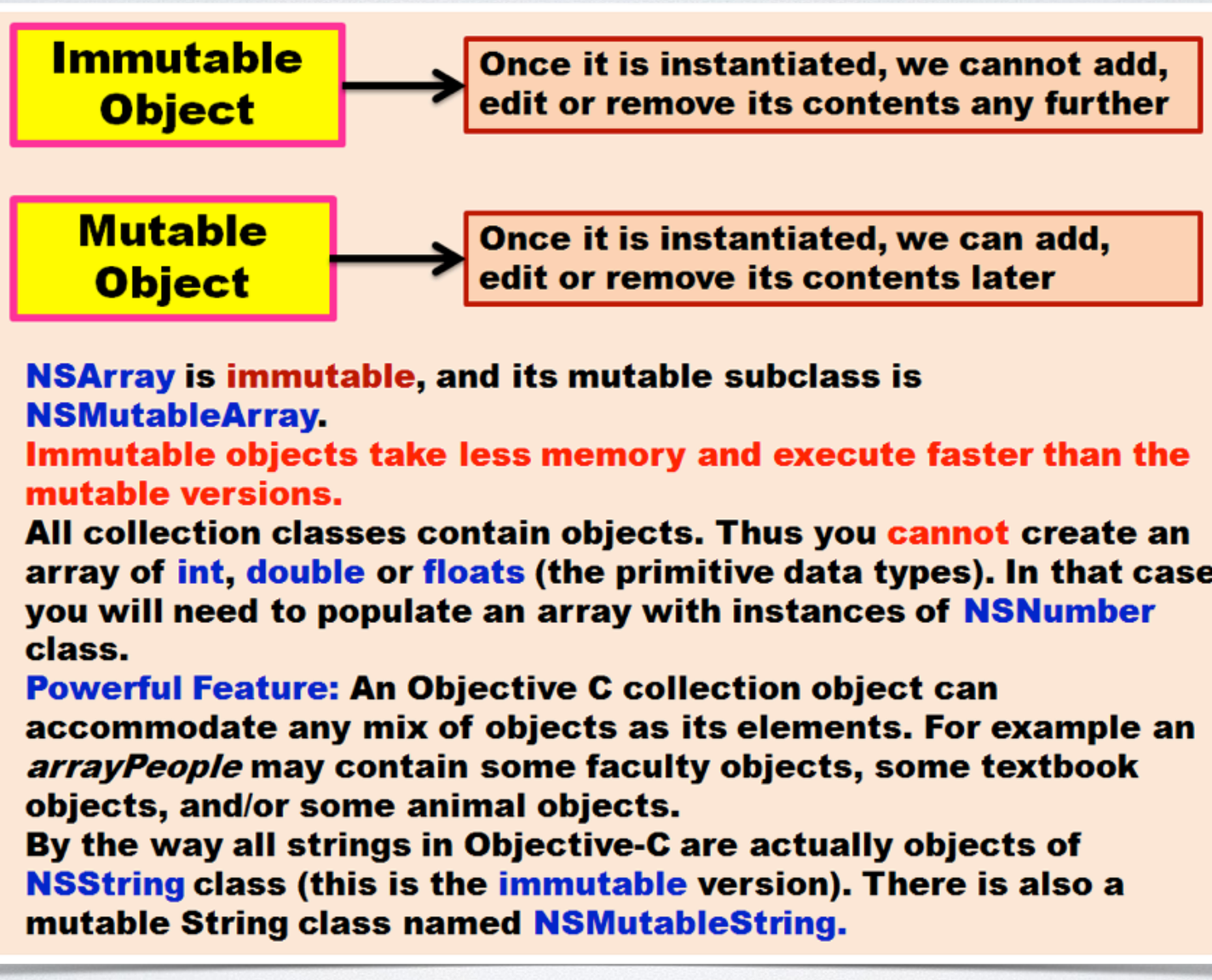
```
1  // allocate and initialize an NSMutableArray object
2  NSMutableArray  *arrayFish = [[NSMutableArray alloc] init];
3
4  [arrayFish addObject:@"Bass" ];
5  [arrayFish addObject:@"Bluegill" ];
6  [arrayFish addObject:@"Crappie" ];
7  [arrayFish addObject:@"Muskie" ];
8  [arrayFish addObject:@"Walleye" ];
```

# SENDING MESSAGES

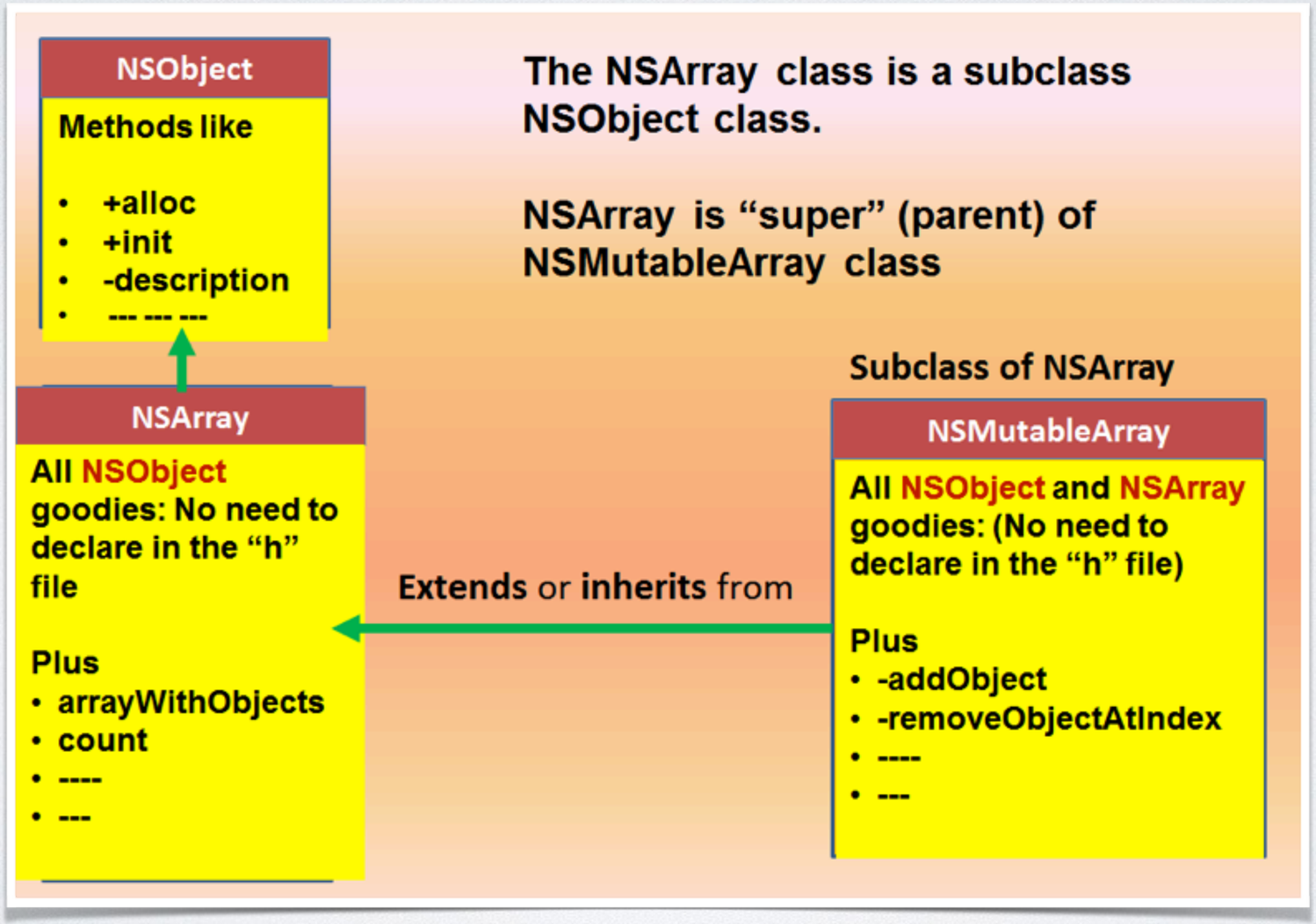



- In this context, let us get introduced to method firing traits in Objective-C. The method firing phenomena is also known as Sending Messages.
  - In Objective –C, usually a method is called in a pair of square brackets.
  - A message has three parts:
    - receiver: to whom are you sending the message
    - selector: which method of the receiver would you want to fire
    - arguments: the values required by the selector, if any

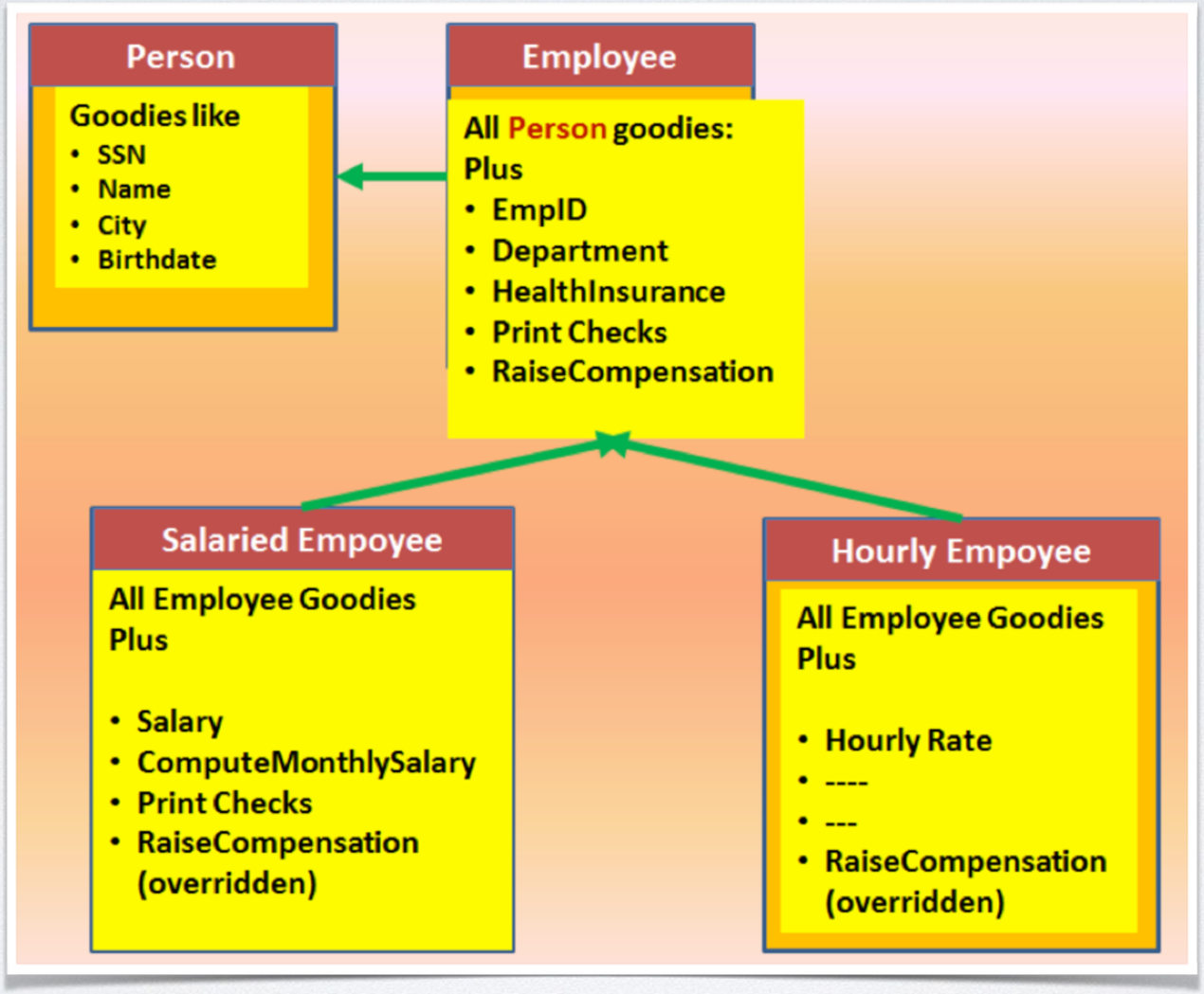
# ABOUT MUTABLE AND IMMUTABLE OBJECTS



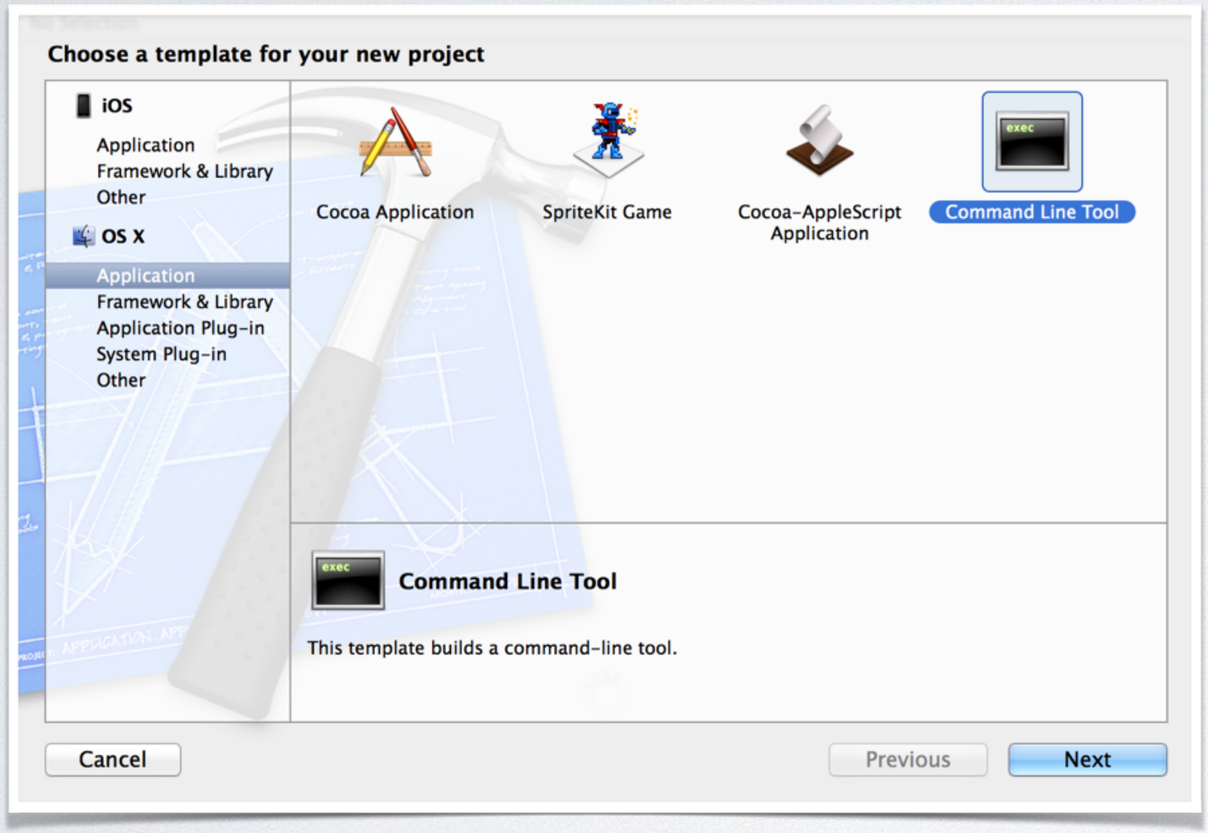
# SUBCLASSES AND NSMUTABLEARRAY GENEALOGY

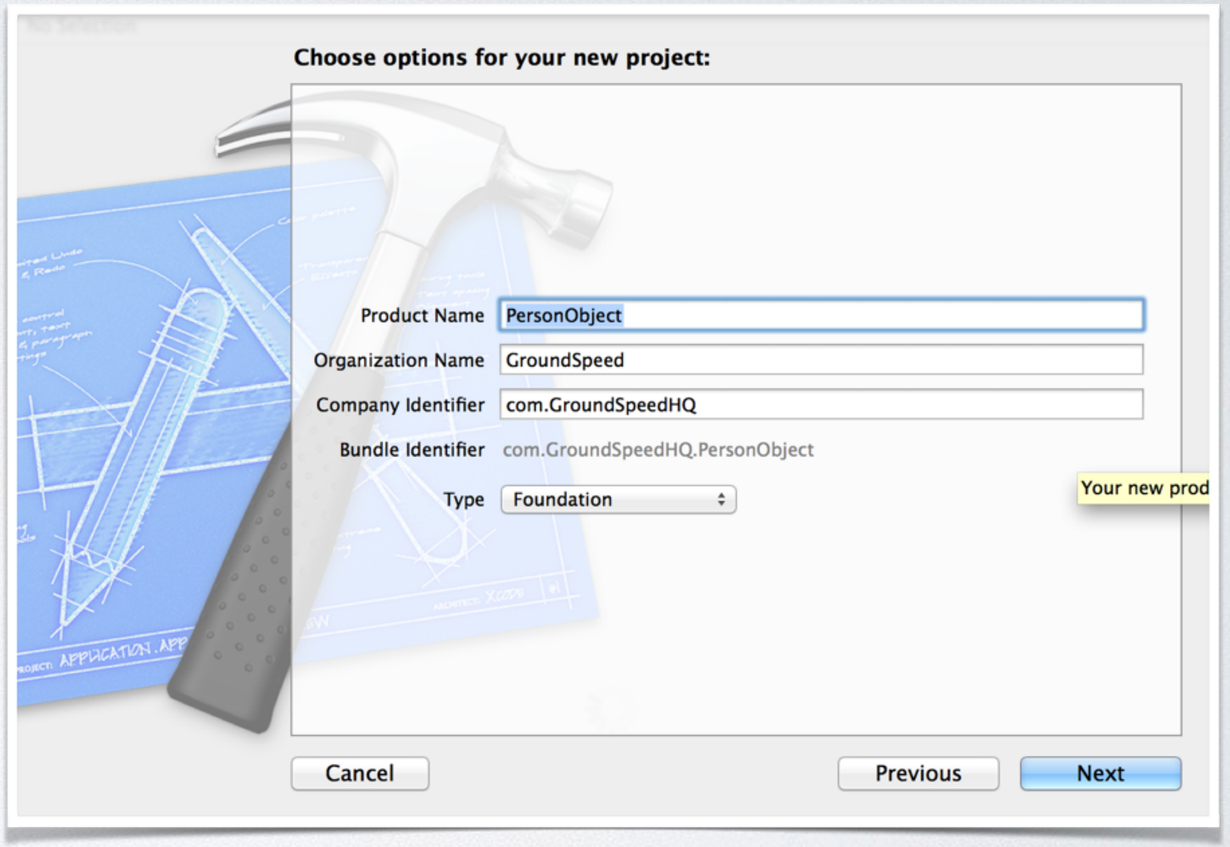


# AN EXAMPLE



# EXAMPLE: CREATE A COMMAND LINE PROGRAM




GroundSpeed™
rapid web + mobile software

# CLASS ILLUSTRATIONS

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

@interface Person : NSObject

@property (nonatomic, strong) NSString *ssn;
@property (nonatomic, strong) NSString *name;
@property (nonatomic, strong) NSString *city;
@property (nonatomic, strong) NSString *state;

@end
```

```objc
#import <Foundation/Foundation.h>
#import "Person.h"

@interface Employee : Person

@property (nonatomic, strong) NSString *empId;
@property (nonatomic, strong) NSString *department;
@property (nonatomic, strong) NSString *healthInsurance;

-(void)printChecks;
-(void)raiseCompetition;

@end
```

```objc
#import <Foundation/Foundation.h>
#import "Employee.h"

@interface SalariedEmployee : Employee

@property (nonatomic, strong) NSString *salary;

-(void)computeMonthlySalary;
-(void)printChecks;
-(void)raiseCompetition;

@end
```

```objc
#import <Foundation/Foundation.h>
#import "Employee.h"

@interface HourlyEmployee : Employee

@property (nonatomic, strong) NSString *hourlyRate;

-(void)raiseCompetition;

@end
```

# CLASS CONSUMPTION

```
9  #import <Foundation/Foundation.h>
10 #import "HourlyEmployee.h"
11
12 int main(int argc, const char * argv[])
13 {
14
15     @autoreleasepool {
16
17         // insert code here...
18         HourlyEmployee *hourlyEmployee = [[HourlyEmployee alloc] init];
19
20         hourlyEmployee.name = @"Don Miller";
21         hourlyEmployee.empId = @"1234";
22         hourlyEmployee.salary = @"9,999";   ! Property 'salary' not found on object of type 'HourlyEmployee *'
23         hourlyEmployee.hourlyRate = @"9.99";
24         [hourlyEmployee raiseCompetition];
25     }
26     return 0;
27 }
```

# STORYBOARD OVERVIEW

- Storyboards are graphic organizers displayed in sequence for the purpose of pre-visualizing a motion picture or animation. The storyboarding process, in the form it is known today, is used by developers to give the designer, developer, and end user an opportunity to see what the application or web site will look like at a high level.

- In iOS, a Storyboard is a container of screens that may have embedded view controllers, navigation controllers, and tab bar controllers. It allows the user to graphically connect screens to one another through embedded objects on the screen. These connections are called segues (pronounced seg-wey). The storyboard eliminates the need for xib files and has one file that contains all of your screens and transitions. It is possible to connect two storyboard files with one another; however, unlikely with smaller applications.

# PROS OF STORYBOARDS

**Pros of using Storyboards:**

- With a storyboard you have a better conceptual overview of all the screens in your app and the connections between them. It's easier to keep track of everything because the entire design is in a single file, rather than spread out over many separate nibs.

- The storyboard describes the transitions between the various screens. These transitions are called "segues" and you create them by simply ctrl-dragging from one view controller to the next. Thanks to segues you need less code to take care of your UI.

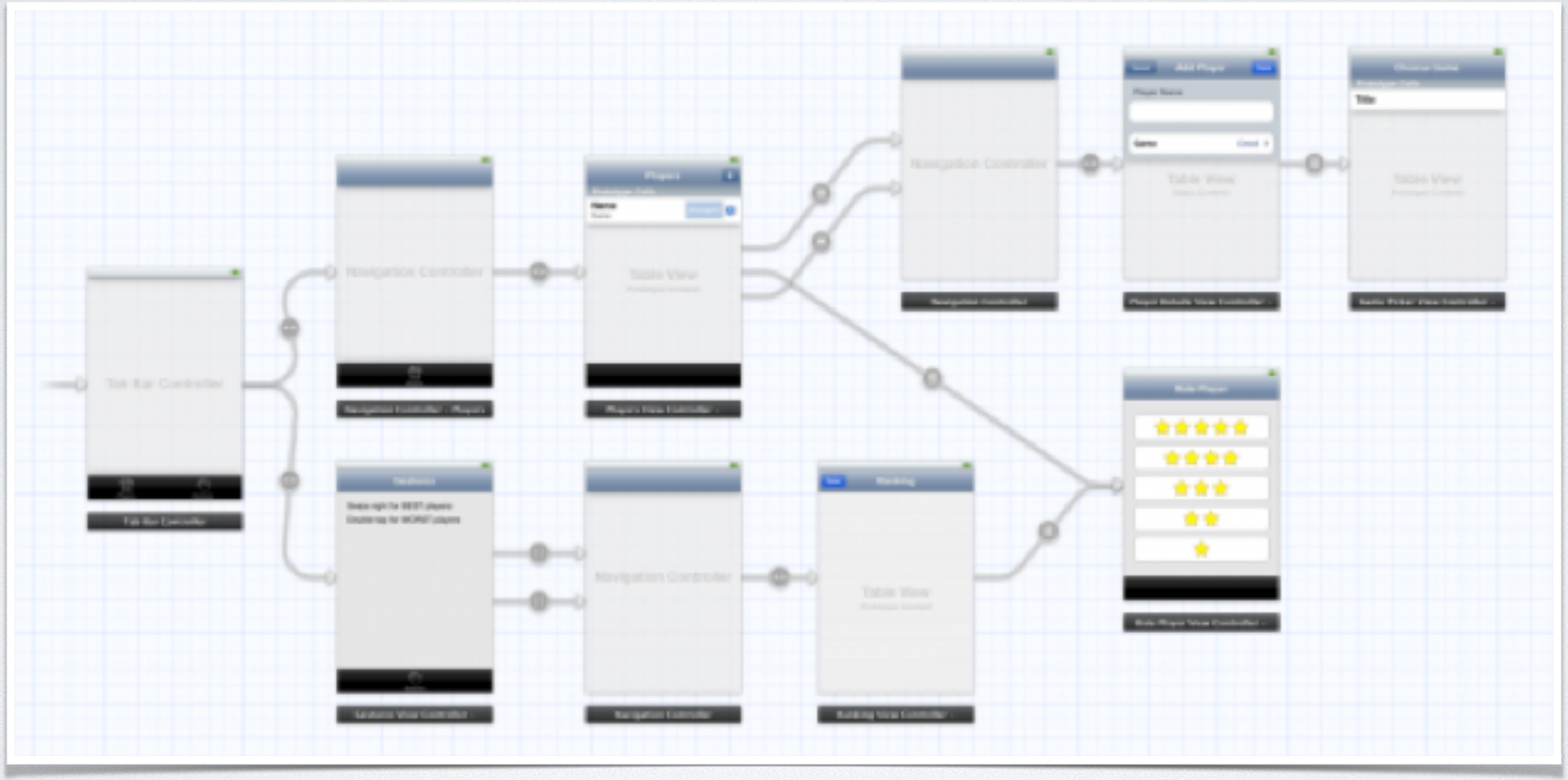
- Storyboards make working with table views a lot easier with the new prototype cells and static cells features. You can design your table views almost completely in the storyboard editor, something else that cuts down on the amount of code you have to write.

# CONS OF STORYBOARDS

**Cons of using Storyboards:**

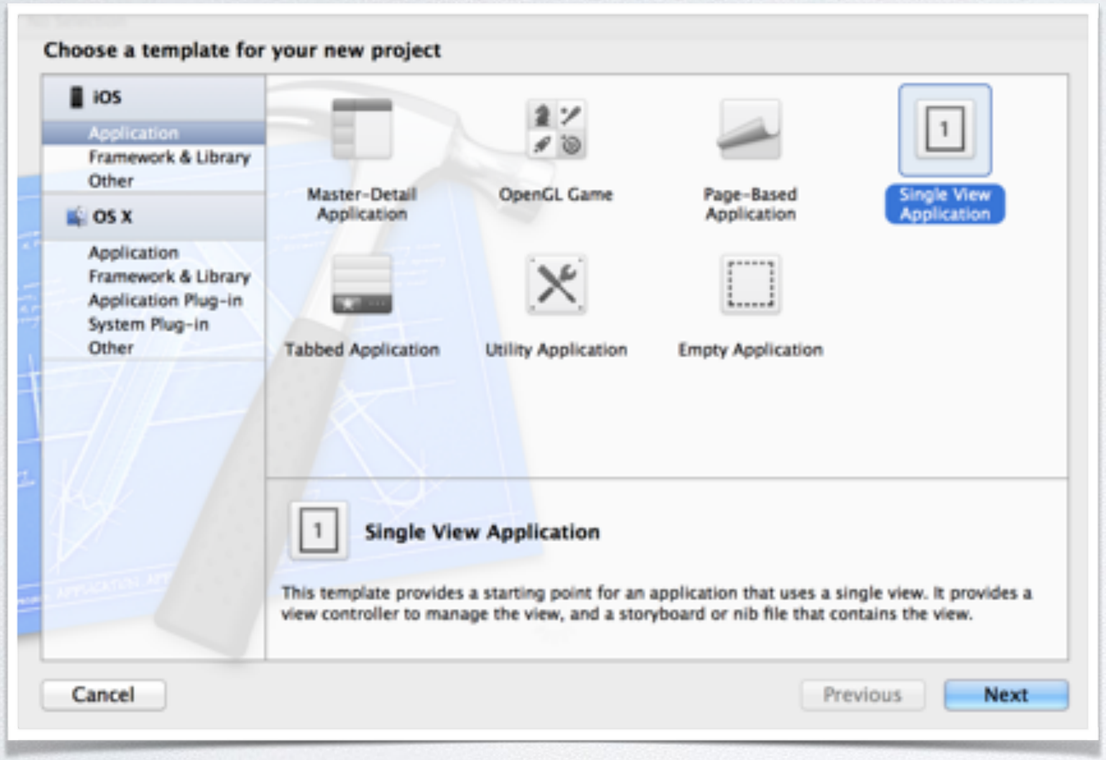
- It's not easy to work with storyboards in a team, since only one participant can work on the storyboard at once (because it's one binary file).
- If you need to do things storyboard doesn't offer, it's not quite easy to get storyboard mixed with programmatically created views)
- Since all objects are in one file, a larger project could cause the storyboard to be very slow and unresponsive.  It loads the entire thing into memory. It loads the entire item into memory.

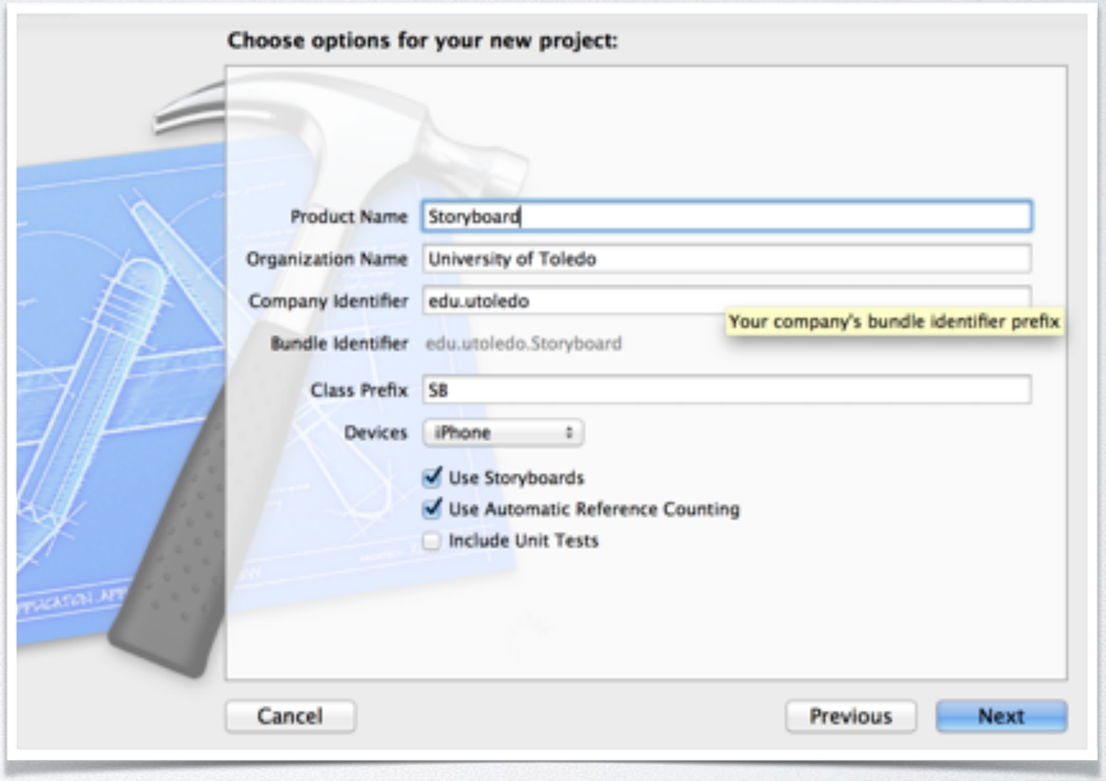
# STORYBOARD OVERVIEW




GroundSpeed™
rapid web + mobile software

# STORYBOARD DEMO
# STEP 1: CREATE A NEW SINGLE VIEW APPLICATION

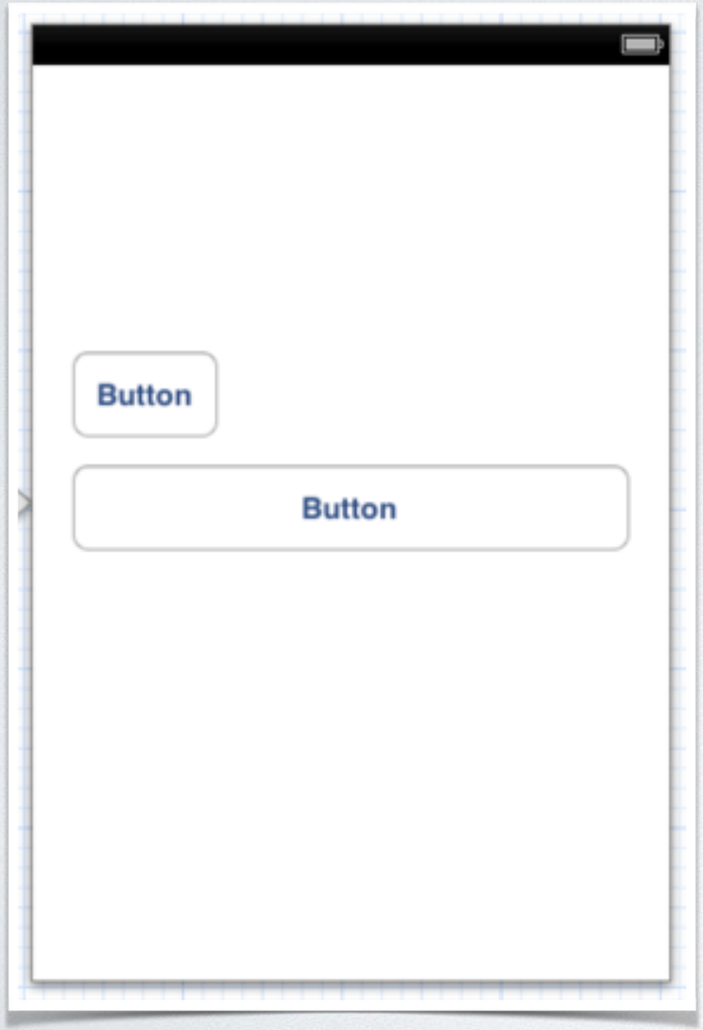


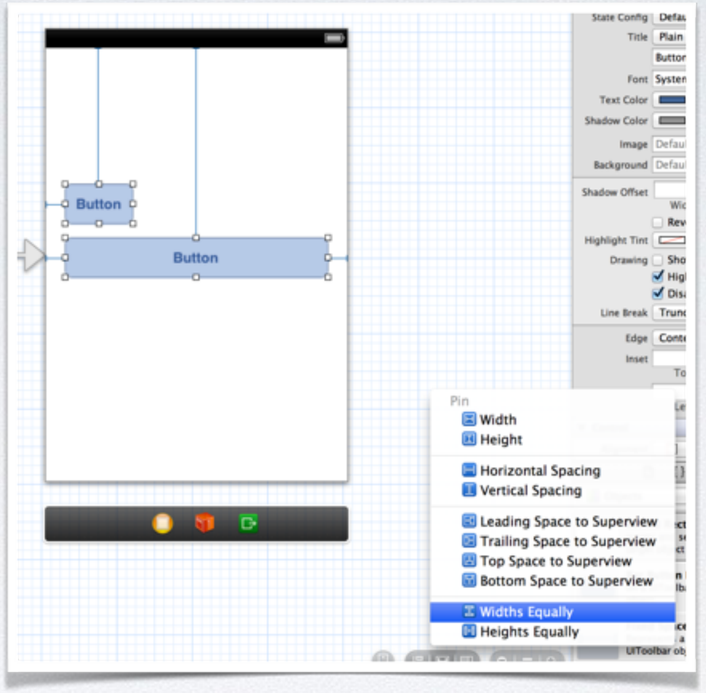



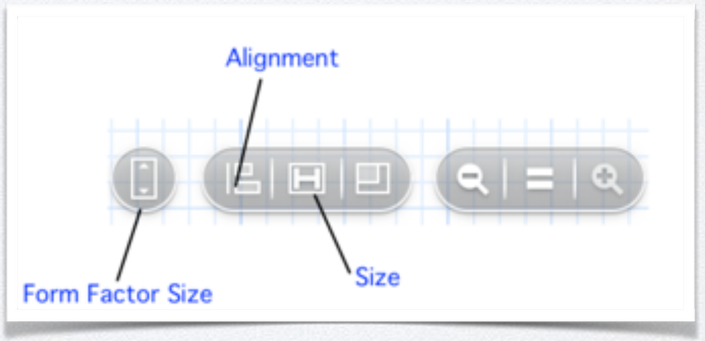
# STEP 2: OBSERVE THE SUMMARY PAGE

The Main Storyboard will default to the newly created storyboard.
We can change it here instead of coding in the AppDelegate.

# STEP 3: ADD 2 BUTTONS TO THE VIEW CONTROLLER
# STEP 4: LET'S MAKE THEM THE SAME WIDTH

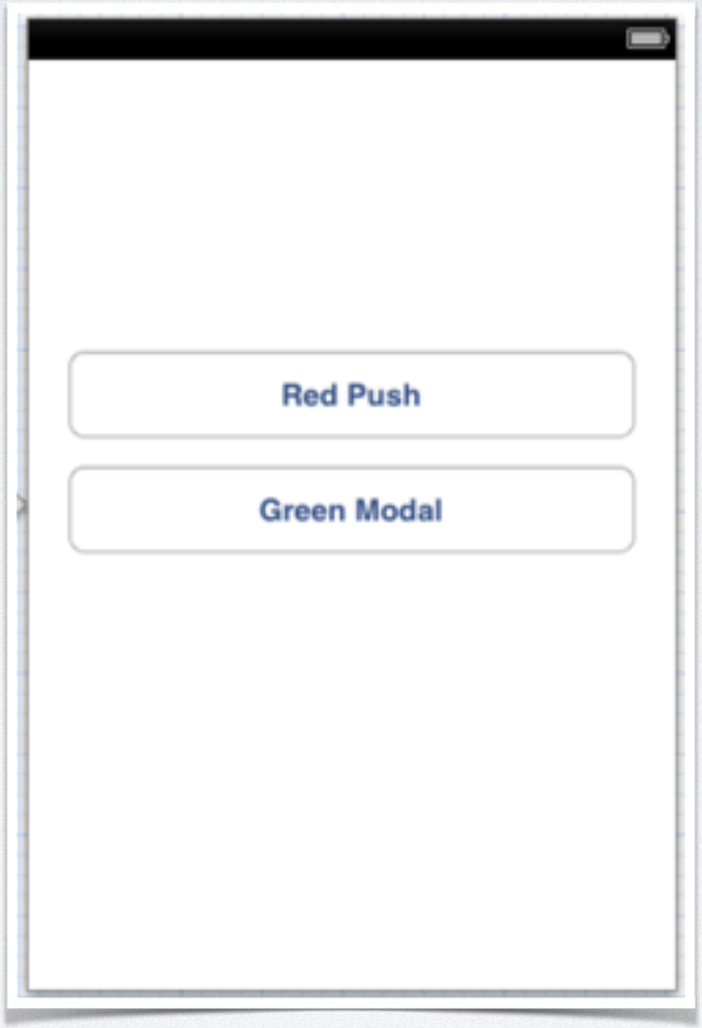


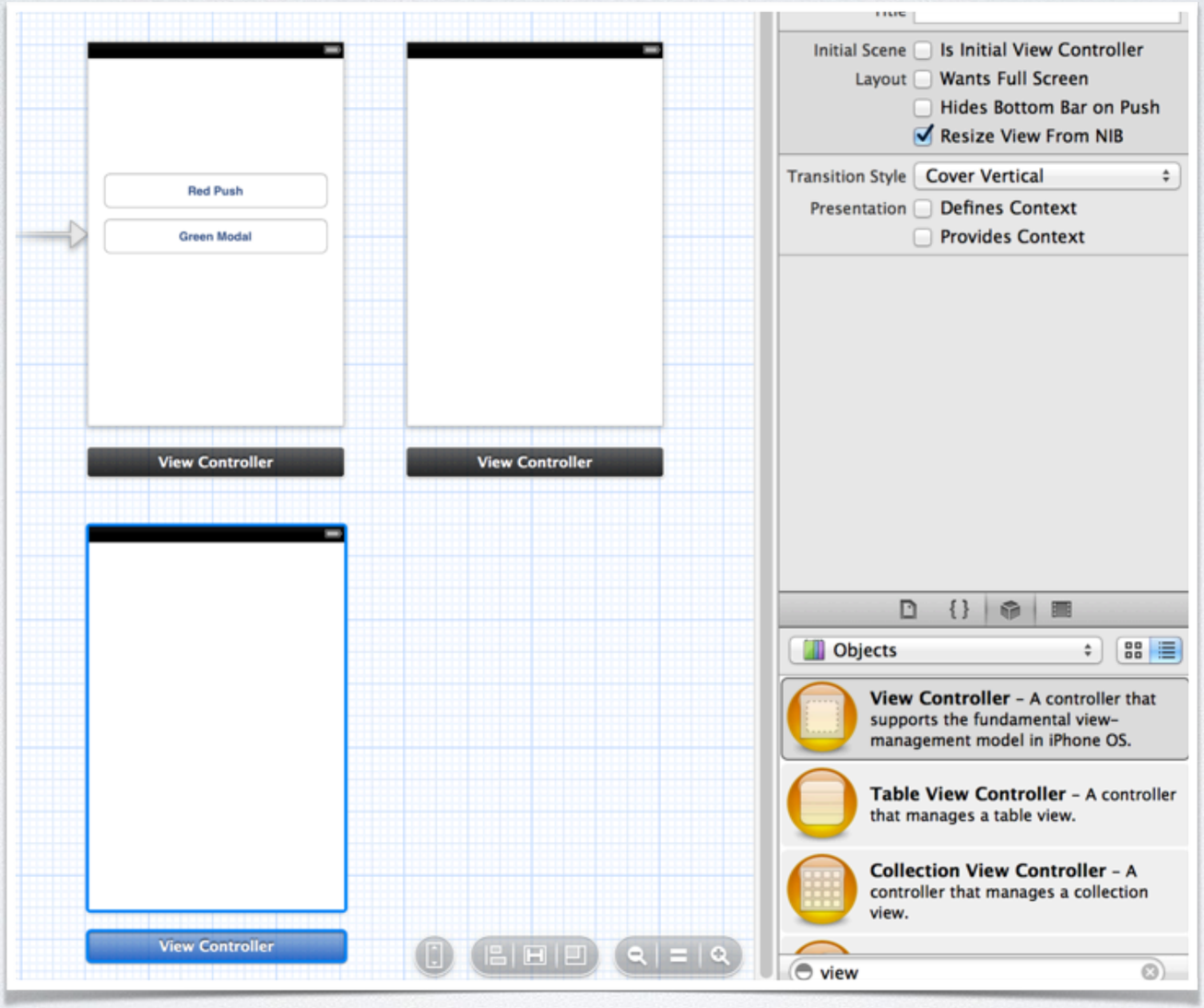




GroundSpeed™
rapid web + mobile software

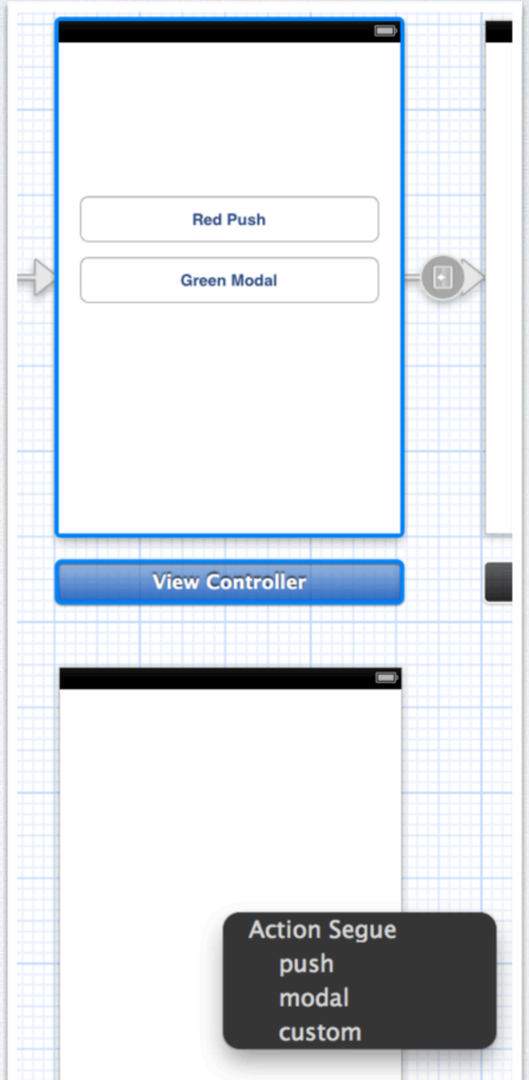
# STEP 5: CHANGE THE BUTTON LABELS TO RED AND GREEN



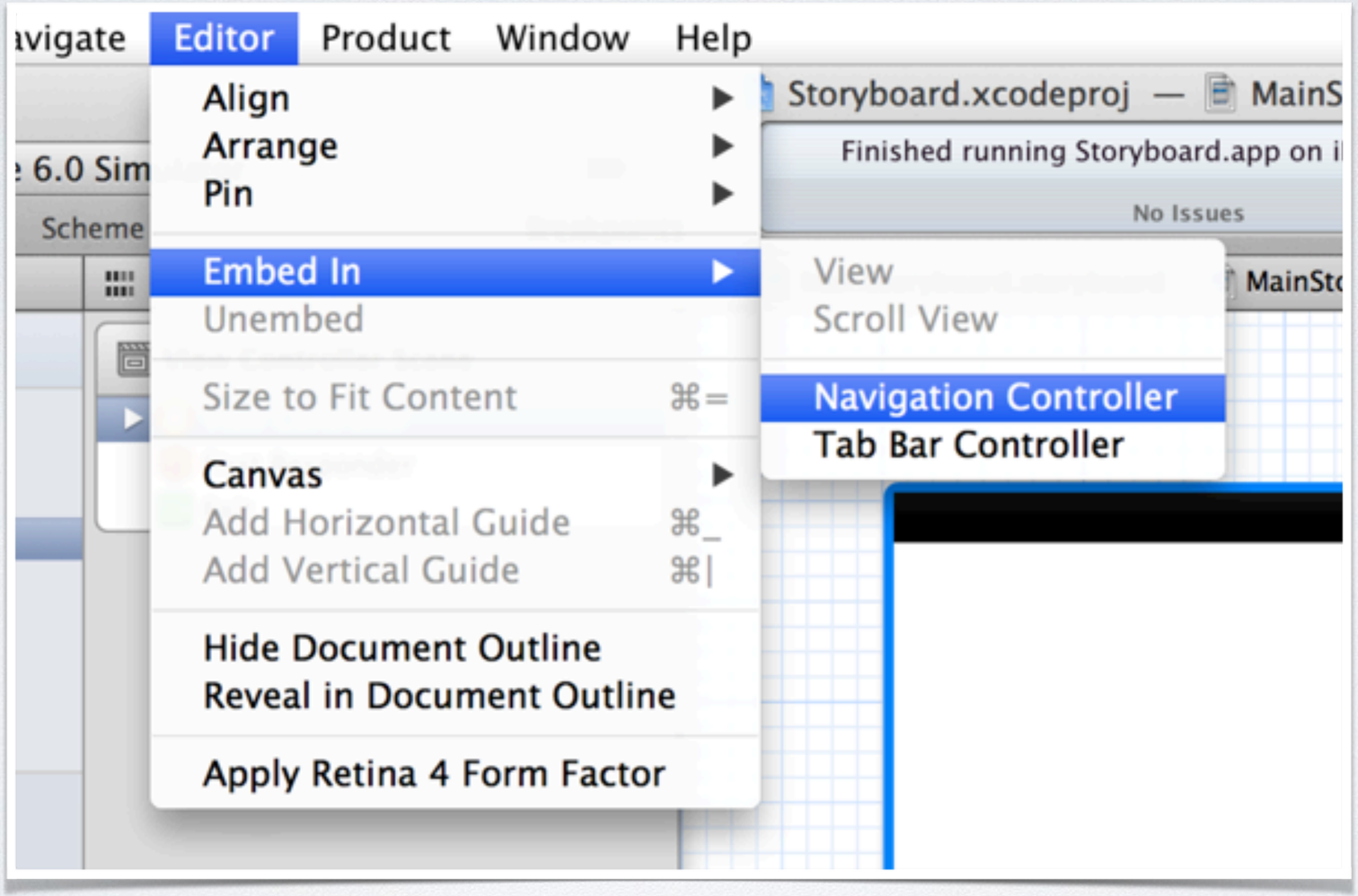
# STEP 6: ADD TWO VIEW CONTROLLERS TO THE STORYBOARD BY DRAGGING THEM OVER



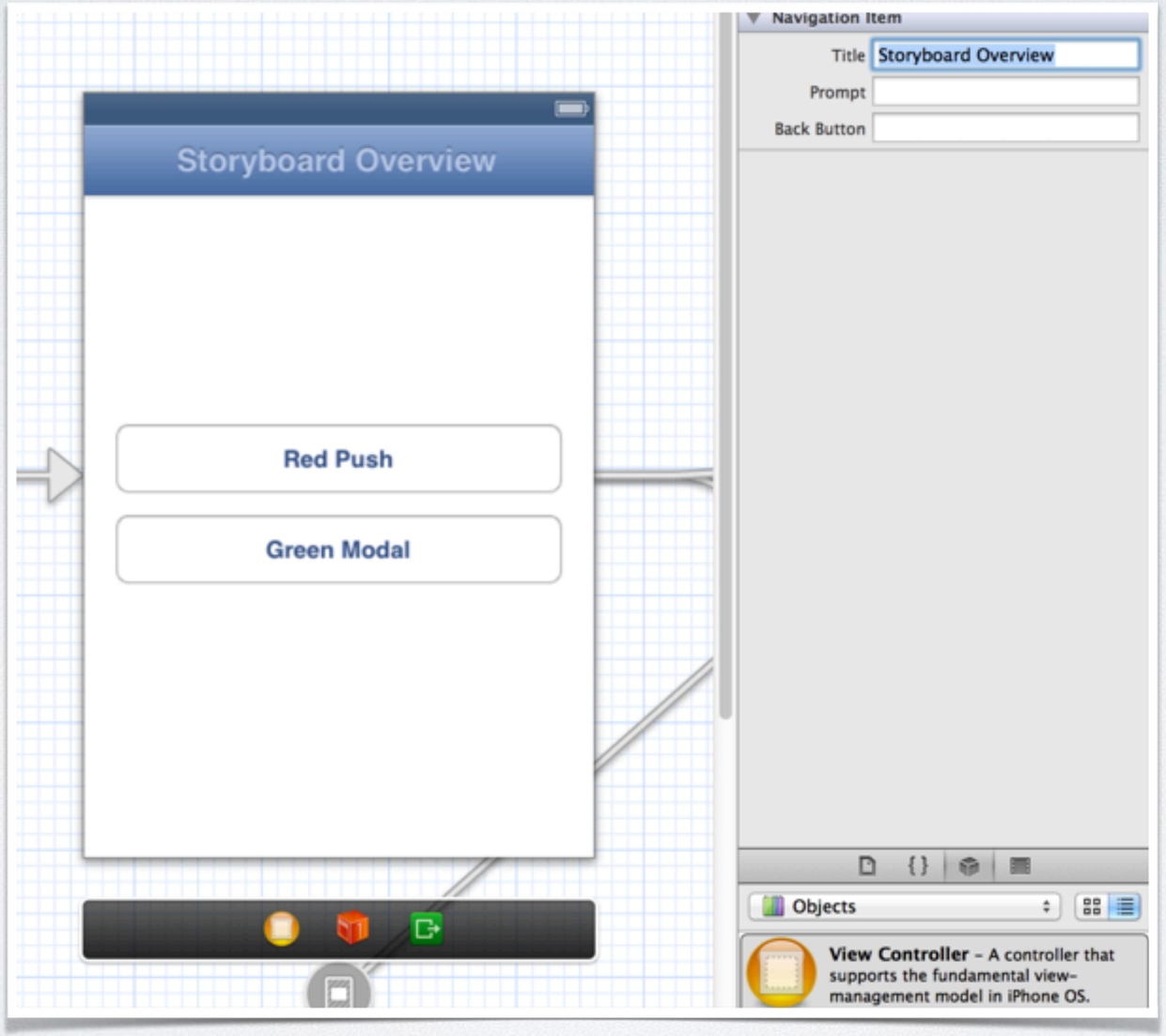
# STEP 7: CONNECT THE BUTTONS TO THE NEW VIEW CONTROLLERS BY HOLDING DOWN THE [CONTROL] KEY AND DRAGGING FROM THE BUTTON TO THE NEW VIEW CONTROLLER

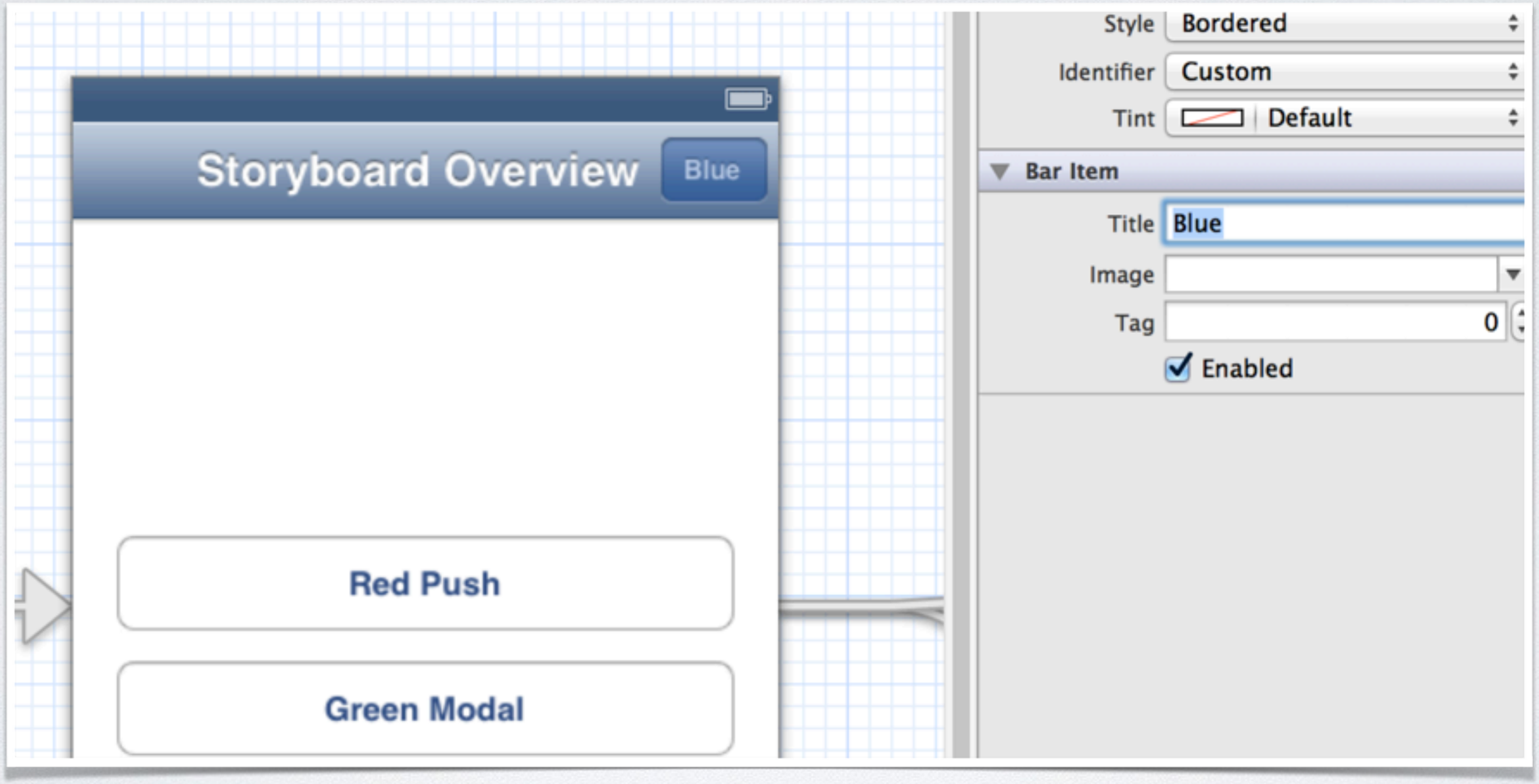
# STEP 8: EMBED A NAVIGATION CONTROLLER ON YOUR PRIMARY VIEW



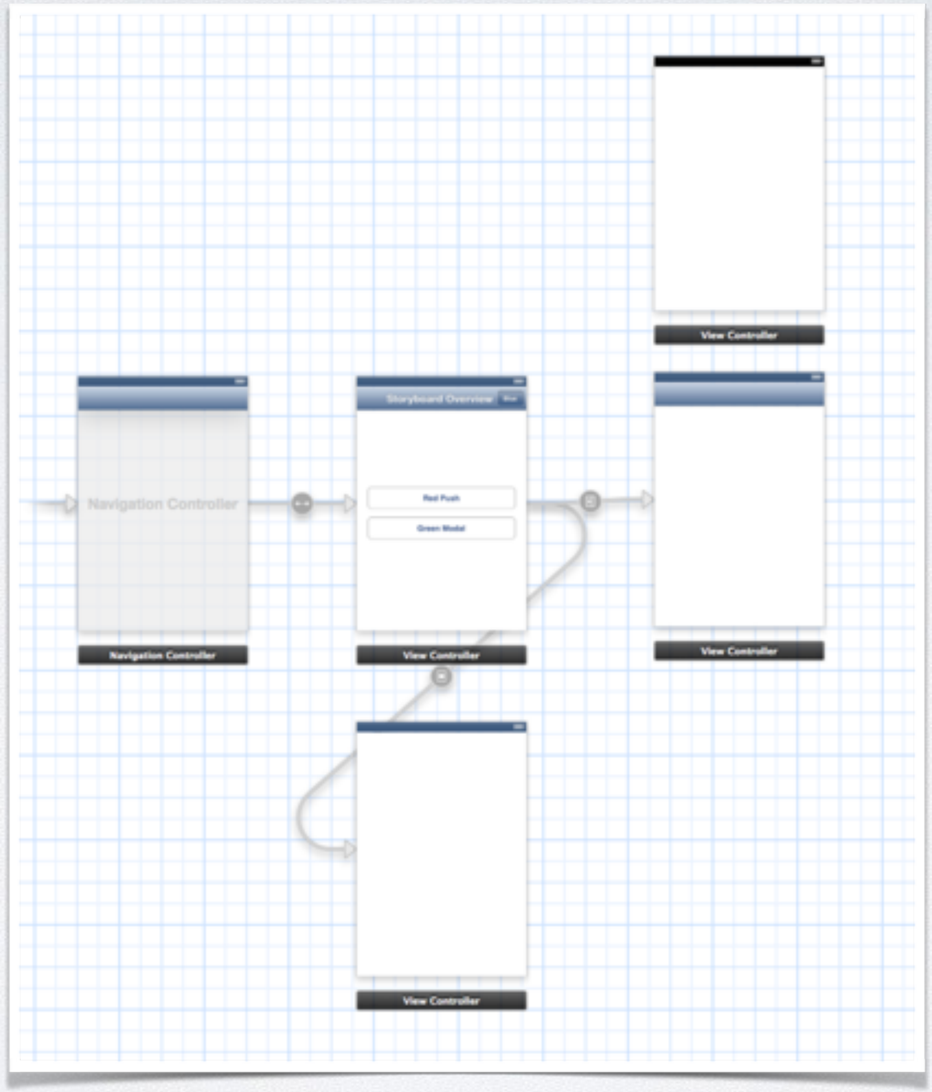
# STEP 9: CHANGE THE TITLE OF THE NAVIGATION BAR



# STEP 10: ADD A BAR BUTTON ITEM TO THE NAVIGATION BAR AND CHANGE THE TITLE TO BLUE

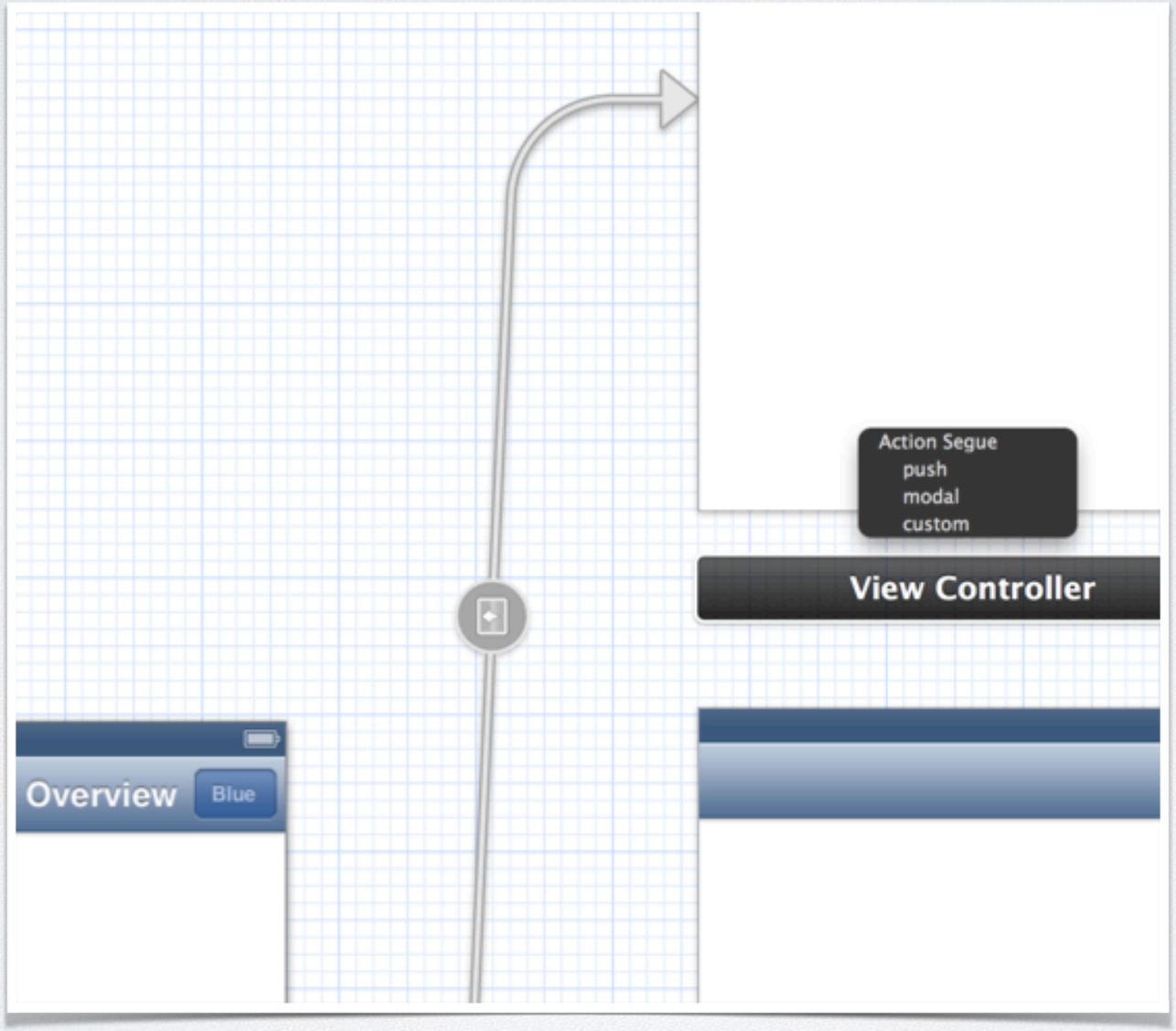


# STEP 11: ADD OUR THIRD AND FINAL VIEW CONTROLLER TO THE STORYBOARD

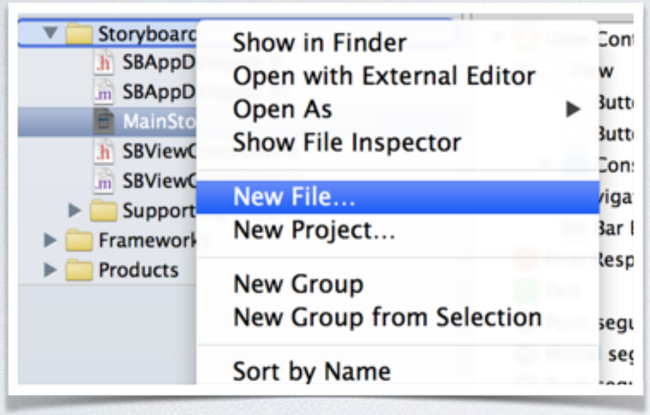


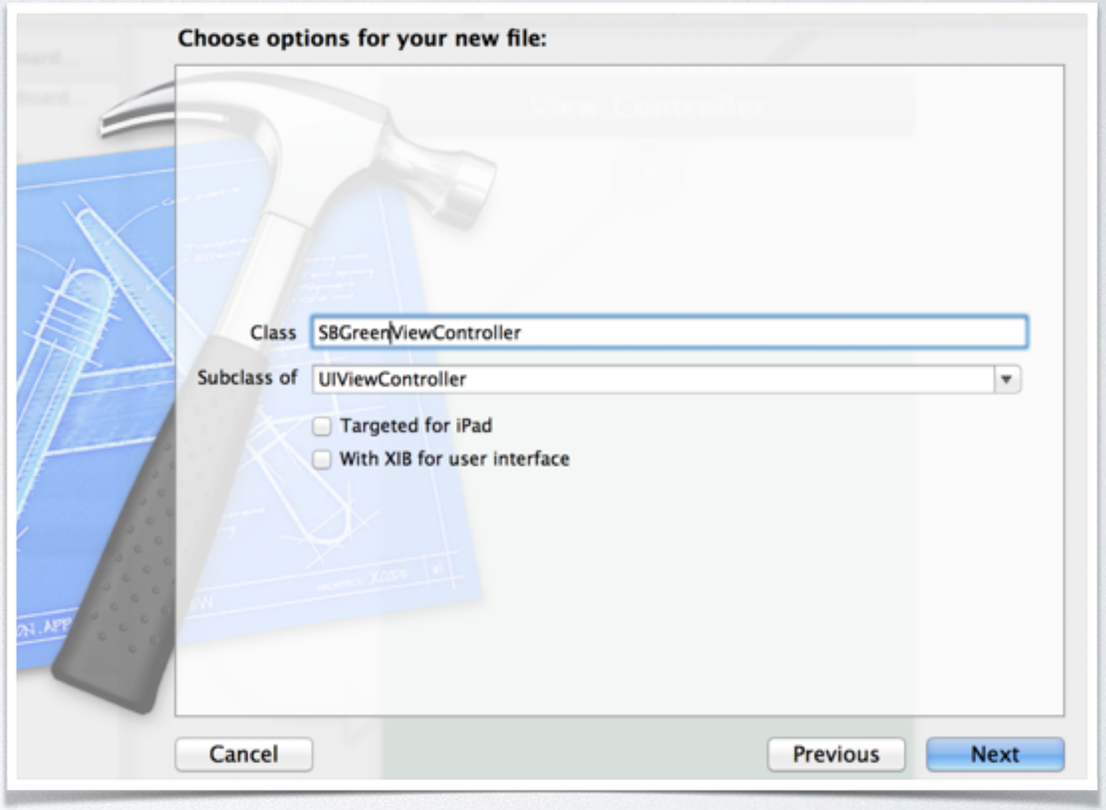

GroundSpeed™
rapid web + mobile software

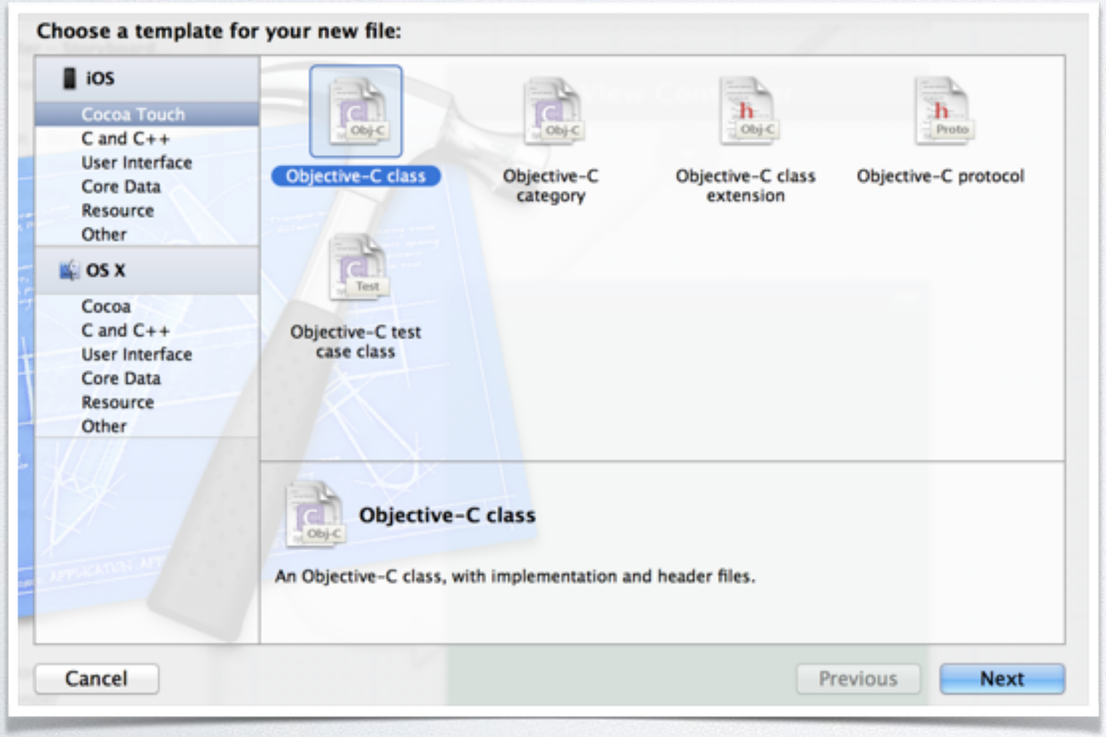
# STEP 12: CONNECT OUR BAR BUTTON ITEM TO THE NEW VIEW CONTROLLER AND SELECT PUSH. REMEMBER TO HOLD DOWN THE [CONTROL] BUTTON WHILE CLICKING AND DRAGGING



# STEP 13: NOW WE NEED TO DO A LITTLE PROGRAMMING. ADD A CONTROLLER OBJECT AND NAME IT "SBGREENVIEWCONTROLLER" AS SHOWN BELOW

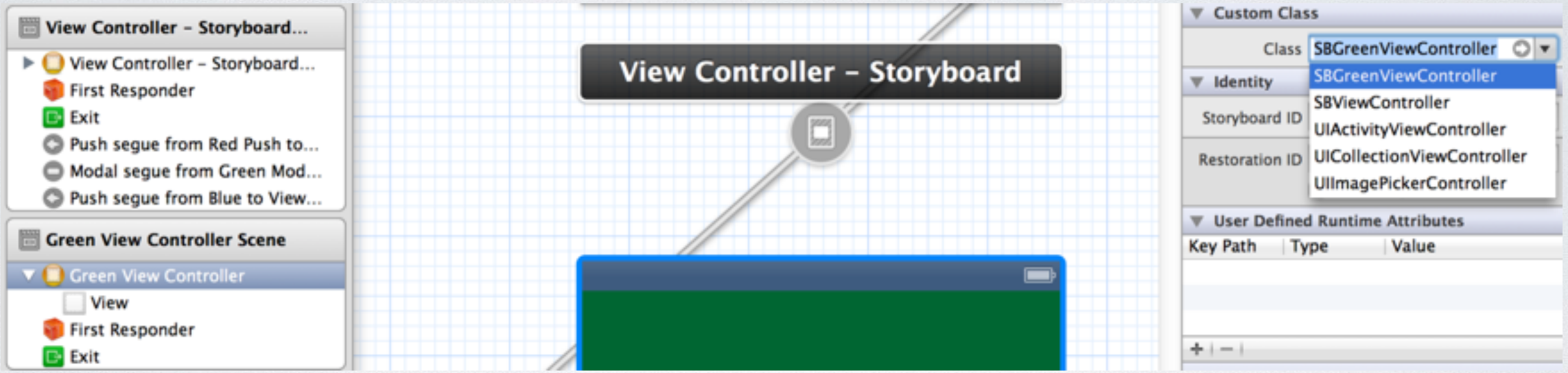


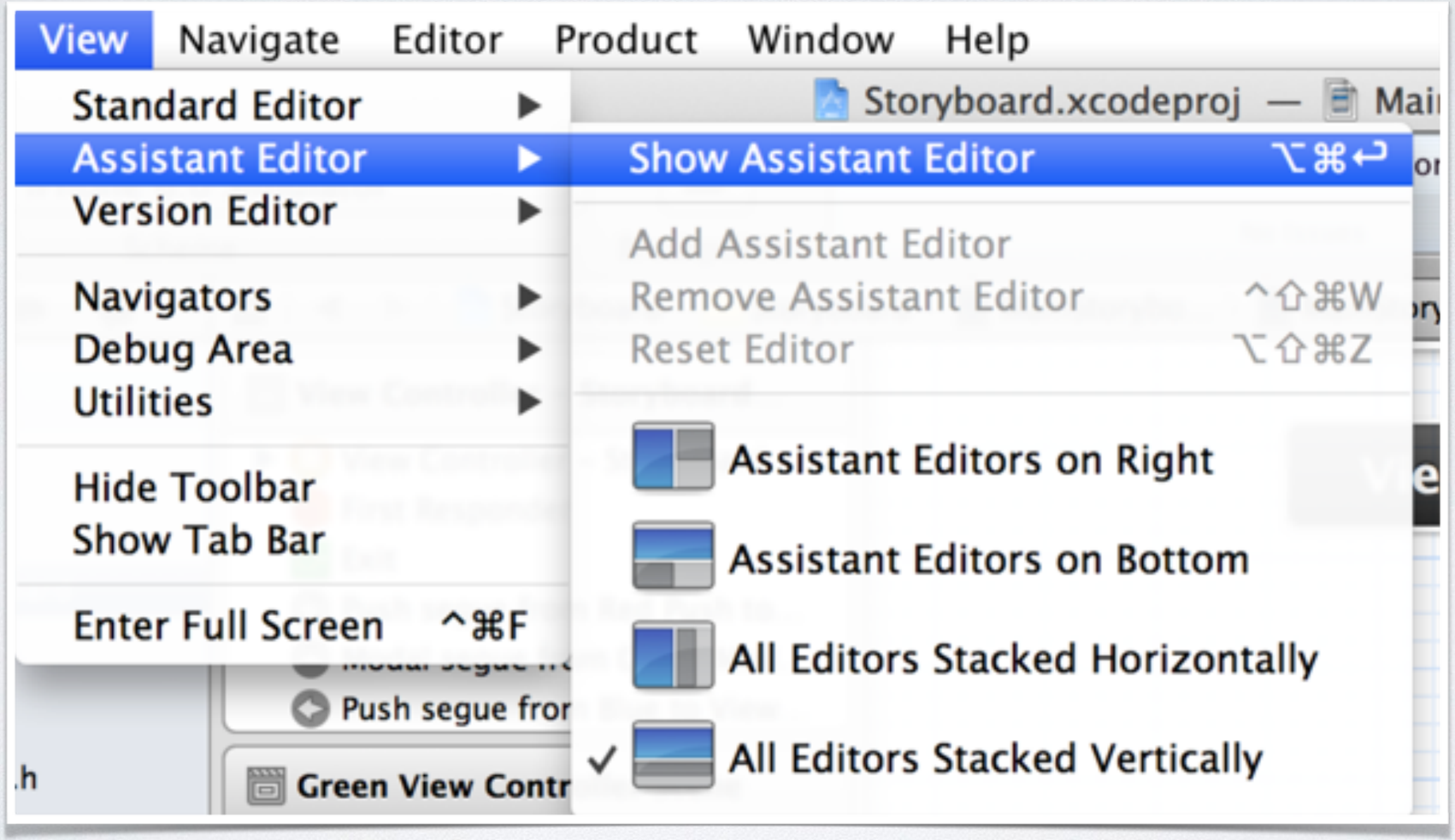

GroundSpeed™
rapid web + mobile software

# STEP 14: ASSIGN YOUR VIEW IN THE STORYBOARD TO THE NEW OBJECT VIEW CONTROLLER YOU JUST CREATED

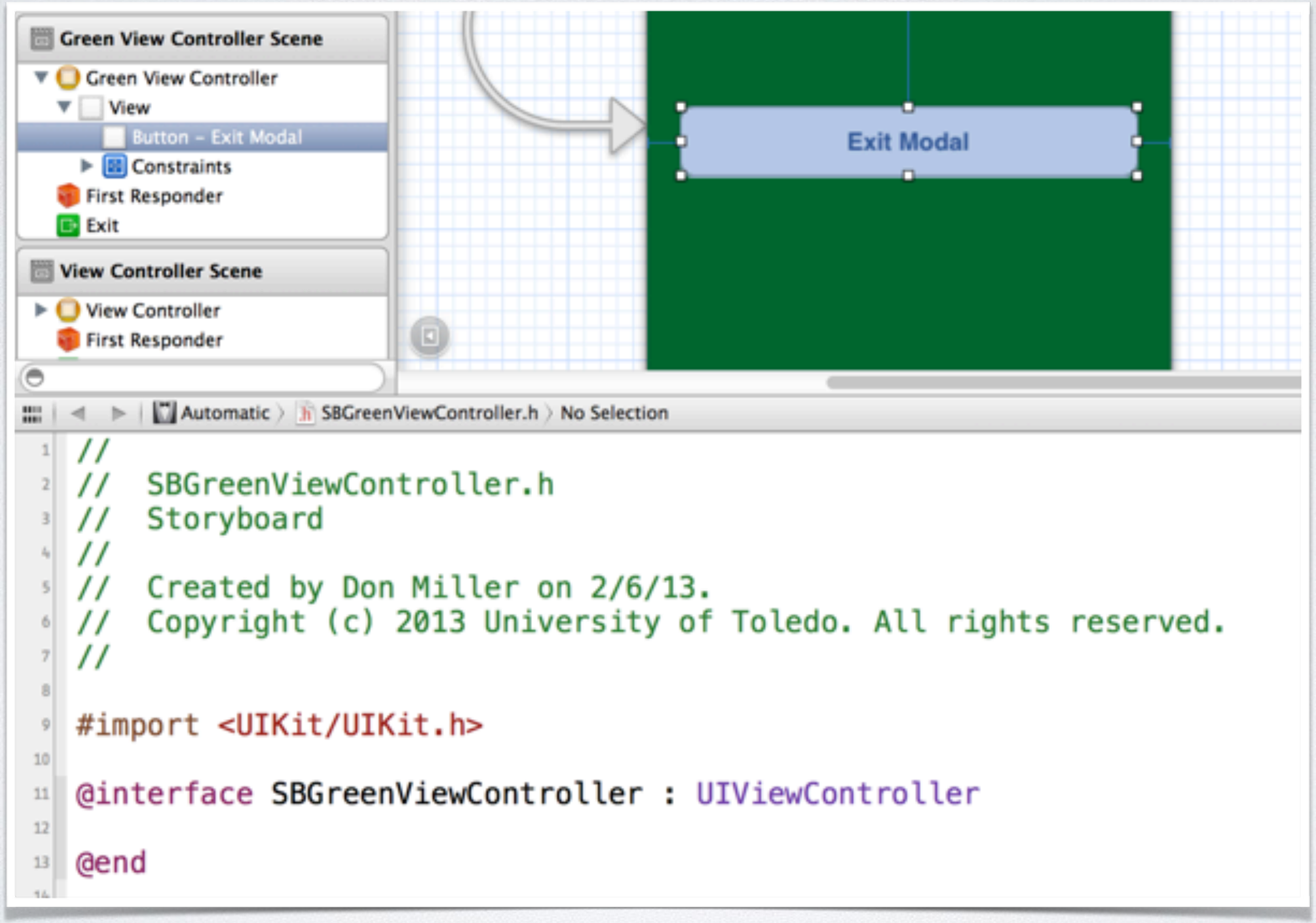



GroundSpeed™
rapid web + mobile software

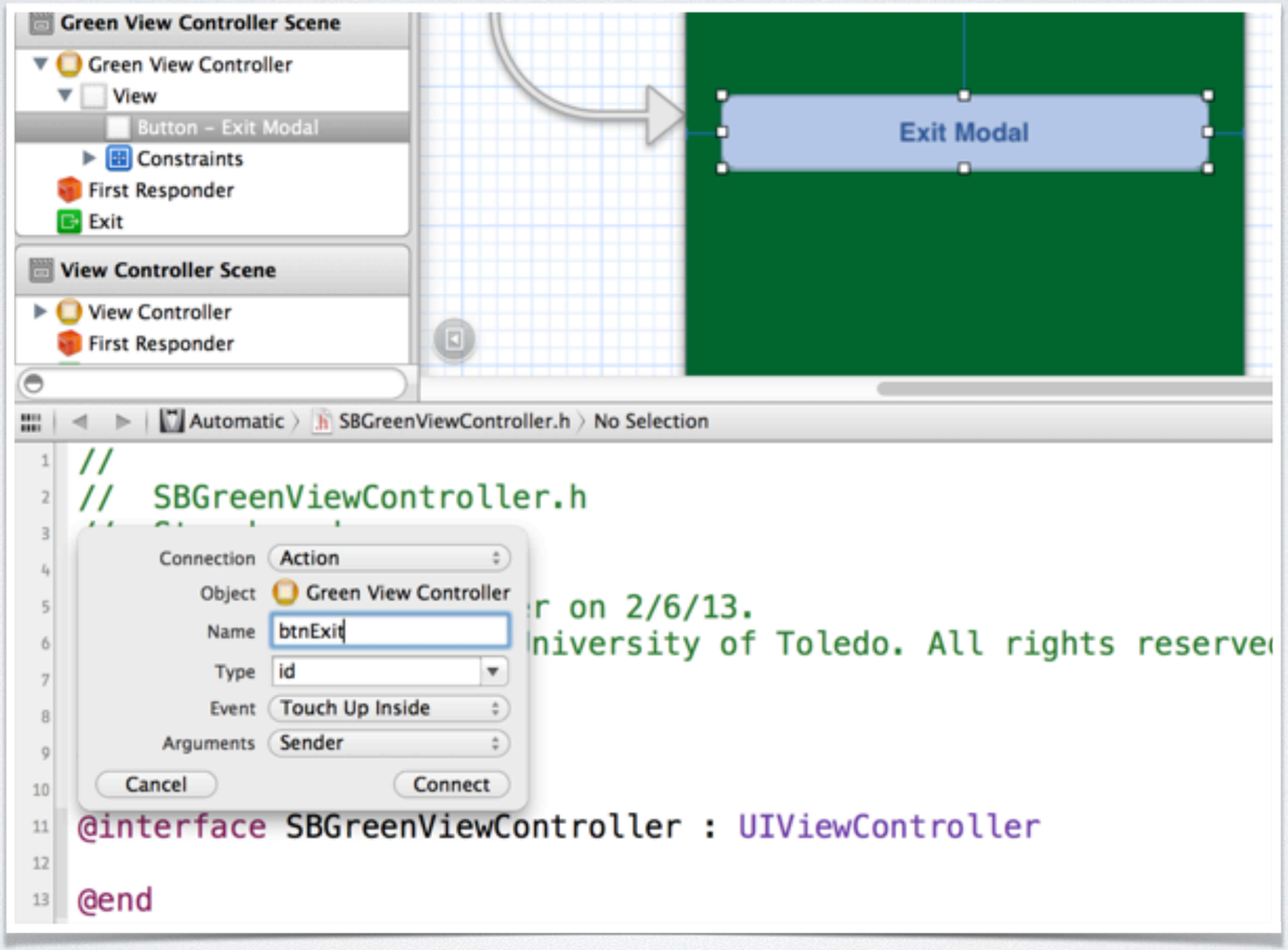
# STEP 15: SHOW THE ASSISTANT EDITOR TO HAVE OUR CODE GENERATED FOR US BY CLICKING



# STEP 16: ADD A BUTTON TO THE GREEN VIEW

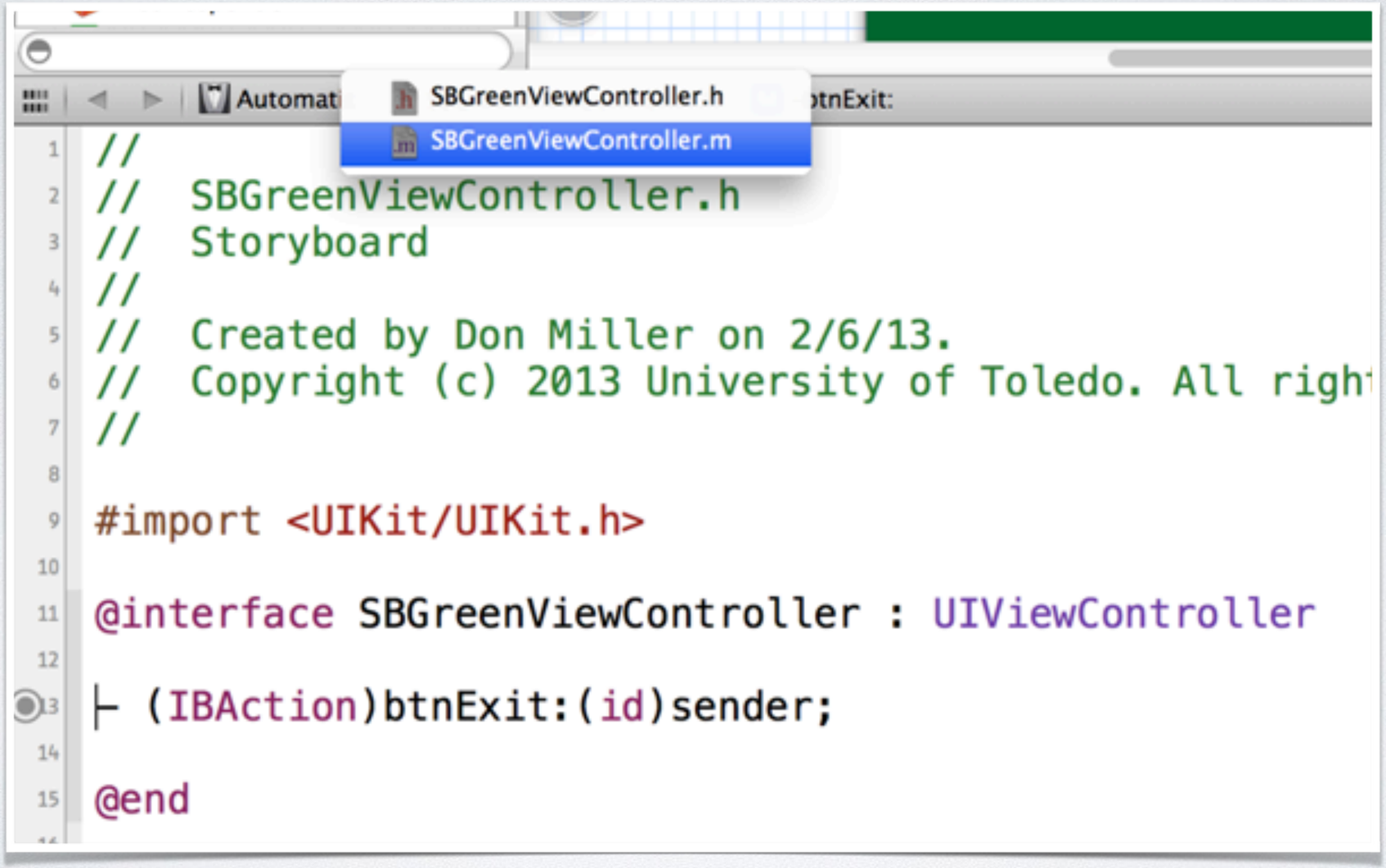


# STEP 17: HOLD DOWN THE CONTROL WHEN CLICKING FROM THE BUTTON TO YOU HEADER FILE AND DRAG IT UNDER THE INTERFACE LINE. CHANGE THE CONNECTION TO ACTION AND THE NAME TO BTNEXIT




GroundSpeed™
rapid web + mobile software

# STEP 18: CHANGE TO YOUR IMPLEMENTATION FILE BY CLICKING THE SBGREENVIEWCONTROLLER.H TITLE AT THE TOP OF THE ASSISTANT EDITOR

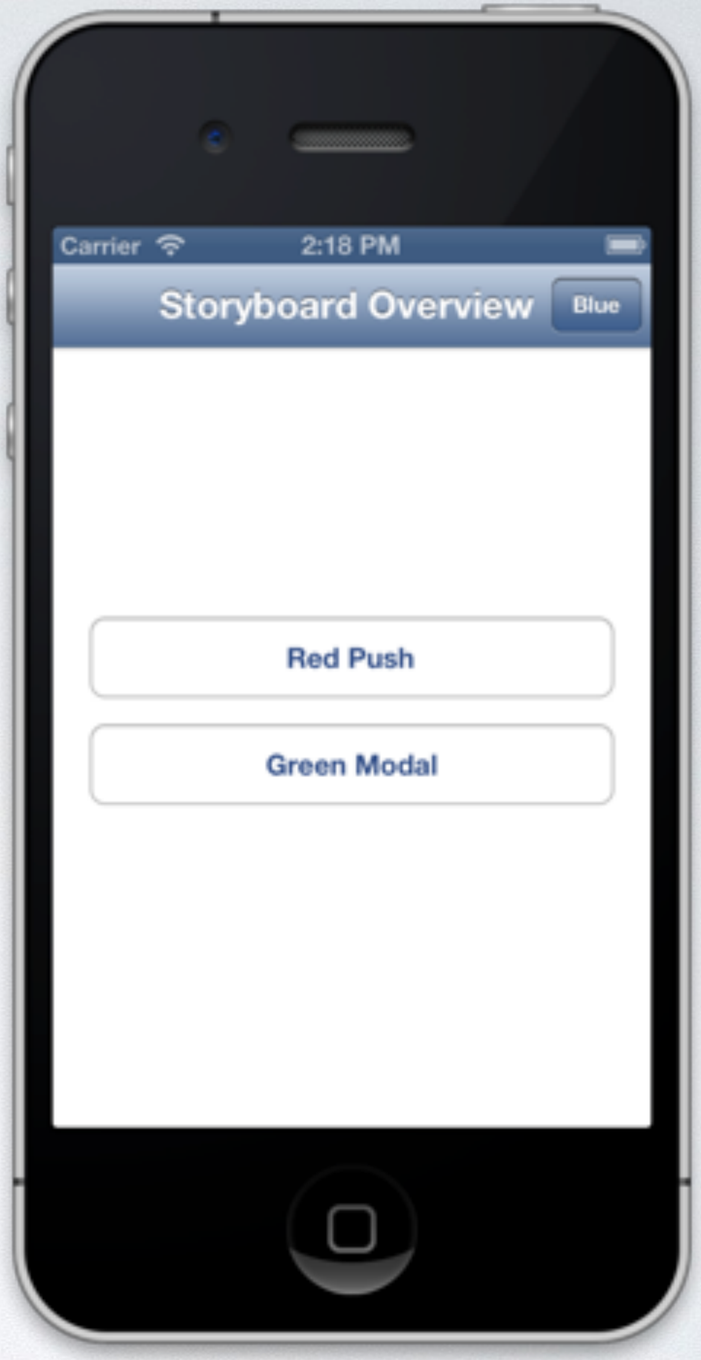


# STEP 19: ADD THE FOLLOWING LINE OF CODE IN THE BTNEXIT METHOD TO DISMISS THE MODAL WINDOW

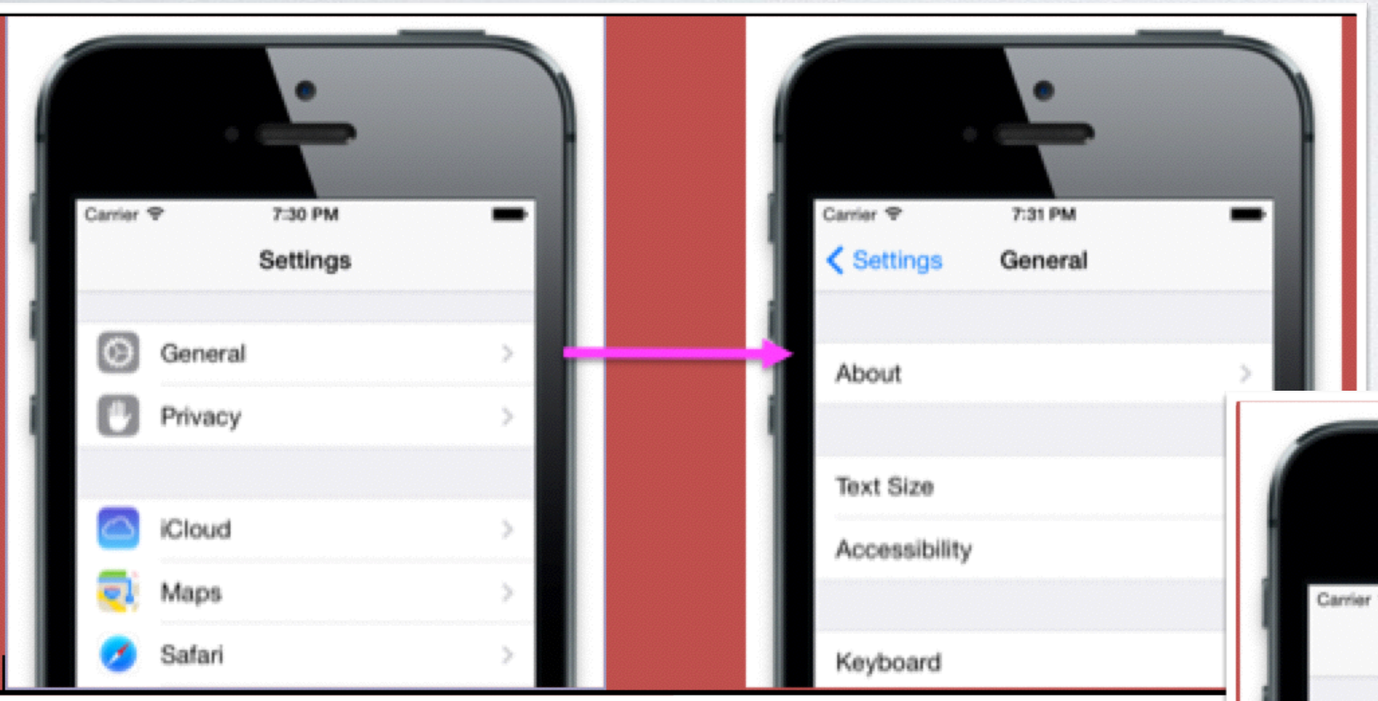
Automatic › SBGreenViewController.m › M -btnExit:

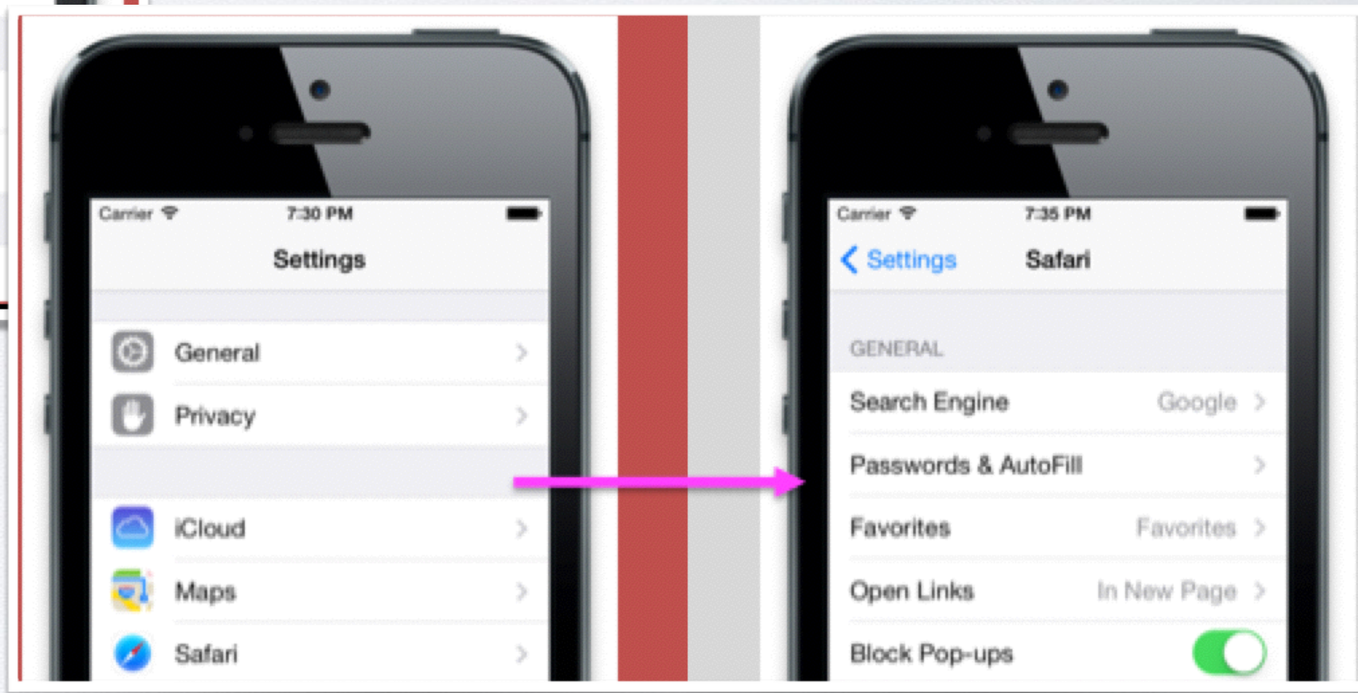
```
24  }
25
26  - (void)viewDidLoad
27  {
28      [super viewDidLoad];
29      // Do any additional setup after loading the view.
30  }
31
32  - (void)didReceiveMemoryWarning
33  {
34      [super didReceiveMemoryWarning];
35      // Dispose of any resources that can be recreated.
36  }
37
38  - (IBAction)btnExit:(id)sender {
39      [self dismissViewControllerAnimated:YES completion:nil];
40  }
41  @end
```

# STEP 20: RUN IT!



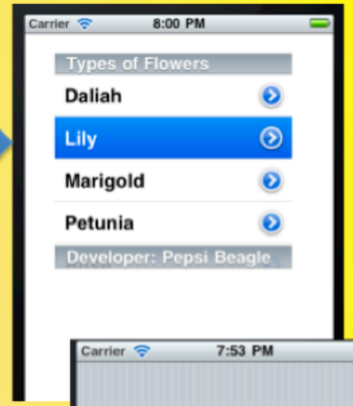
# UITABLEVIEW OVERVIEW



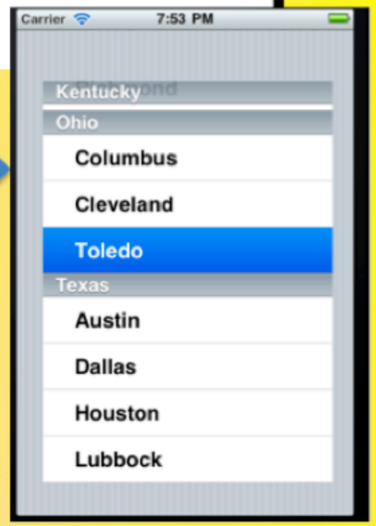



# UITABLEVIEW OVERVIEW

**A simple UI Table View with 4 rows**




**A sectioned UI Table View**



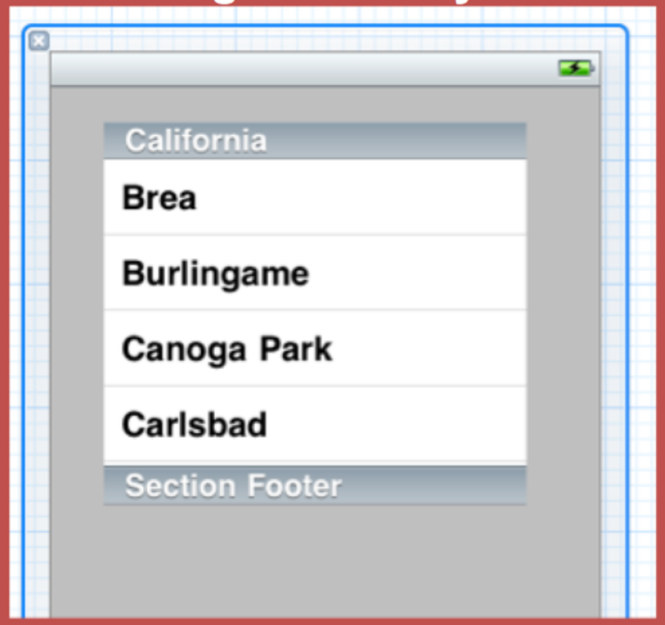

- **Programmatically, the UIKit identifies rows and sections through their index number.**
- **Sections are numbered 0 through n-1 from the top of a table view to the bottom.**
- **Rows are numbered 0 through n-1 within a section.**
- **As mentioned earlier, the table view comes in one of two basic styles: plain or grouped.**

**If you employ a table view object, it is always a good practice to declare the protocol conformance in the header file, such as:**

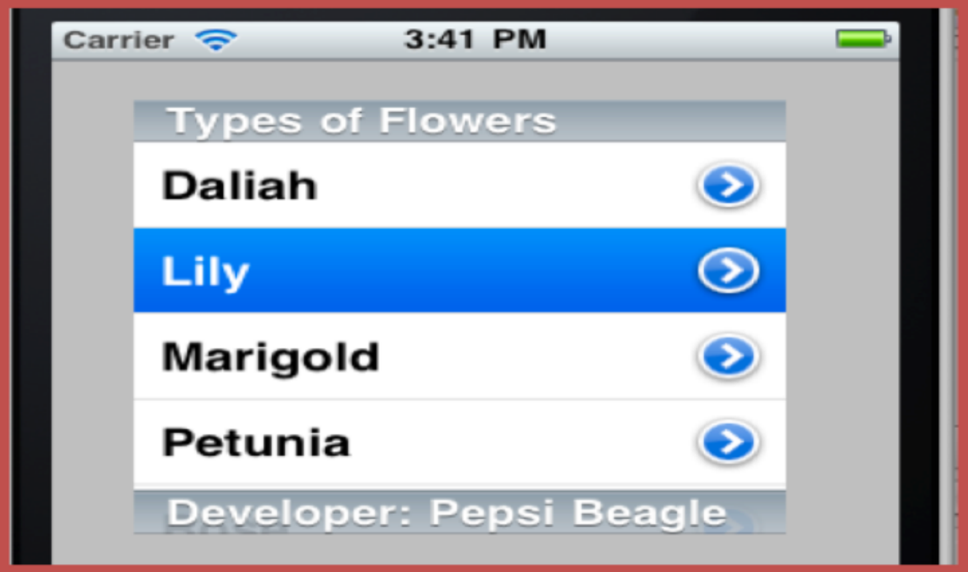
```
@interface xyzViewController : UIViewController
<UITableViewDataSource, UITableViewDelegate> {
  --- --- --- ---
  --- --- --- ---
  --- --- --- ---
}
```

# TABLE VIEW EXAMPLE

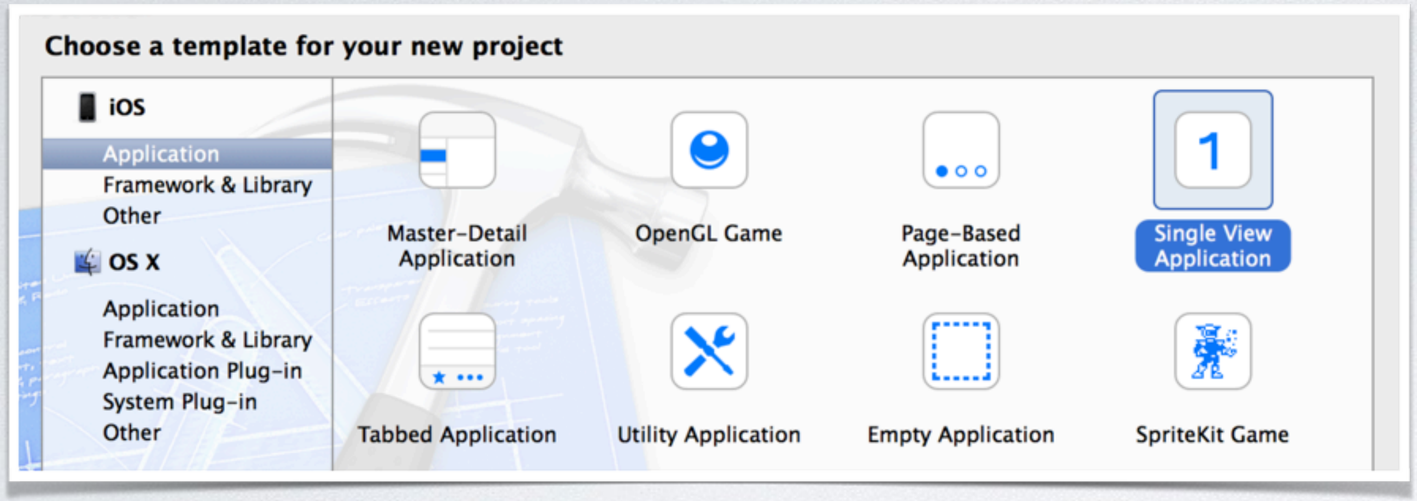
**Design Time Layout**



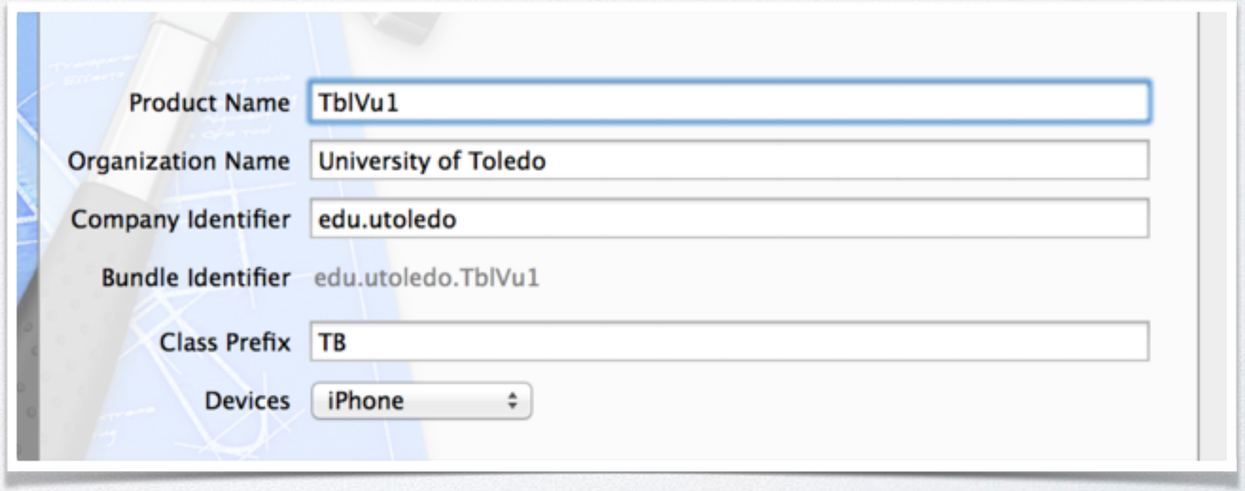

**Run Time Screen Shot**



# STEP 1: CREATE A SINGLE VIEW BASED APP.

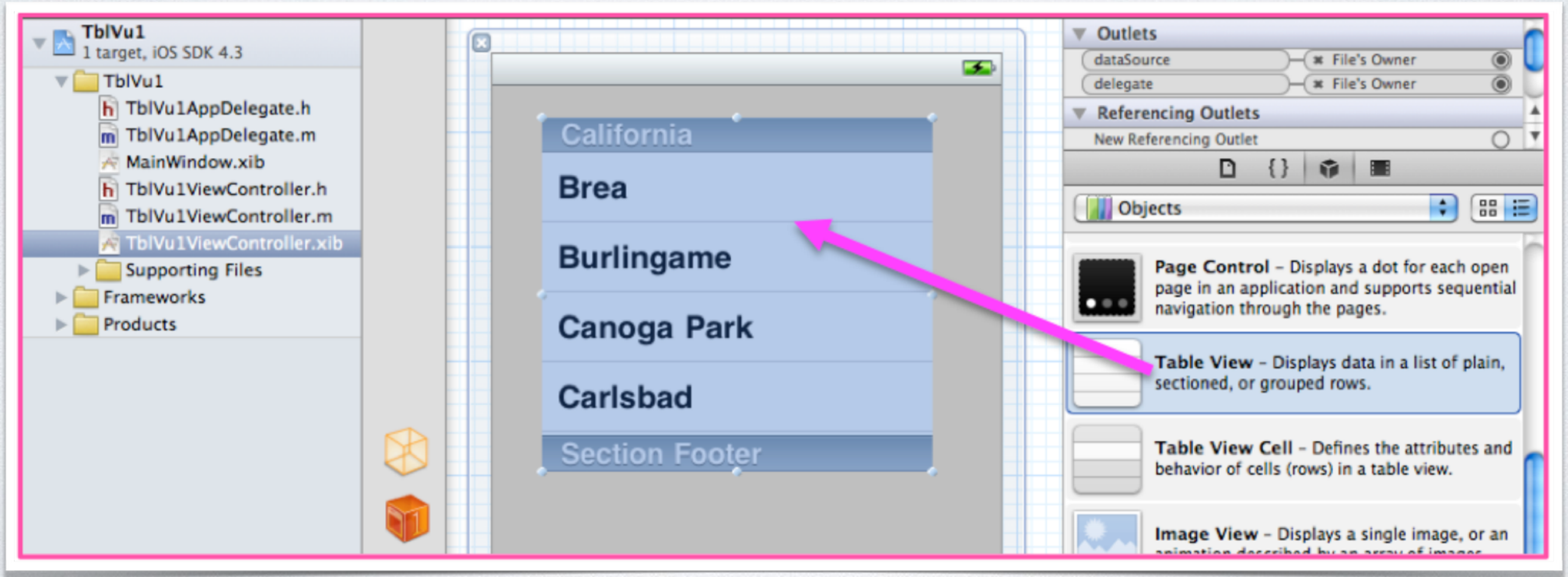



- **Let's name it: TblVu1**



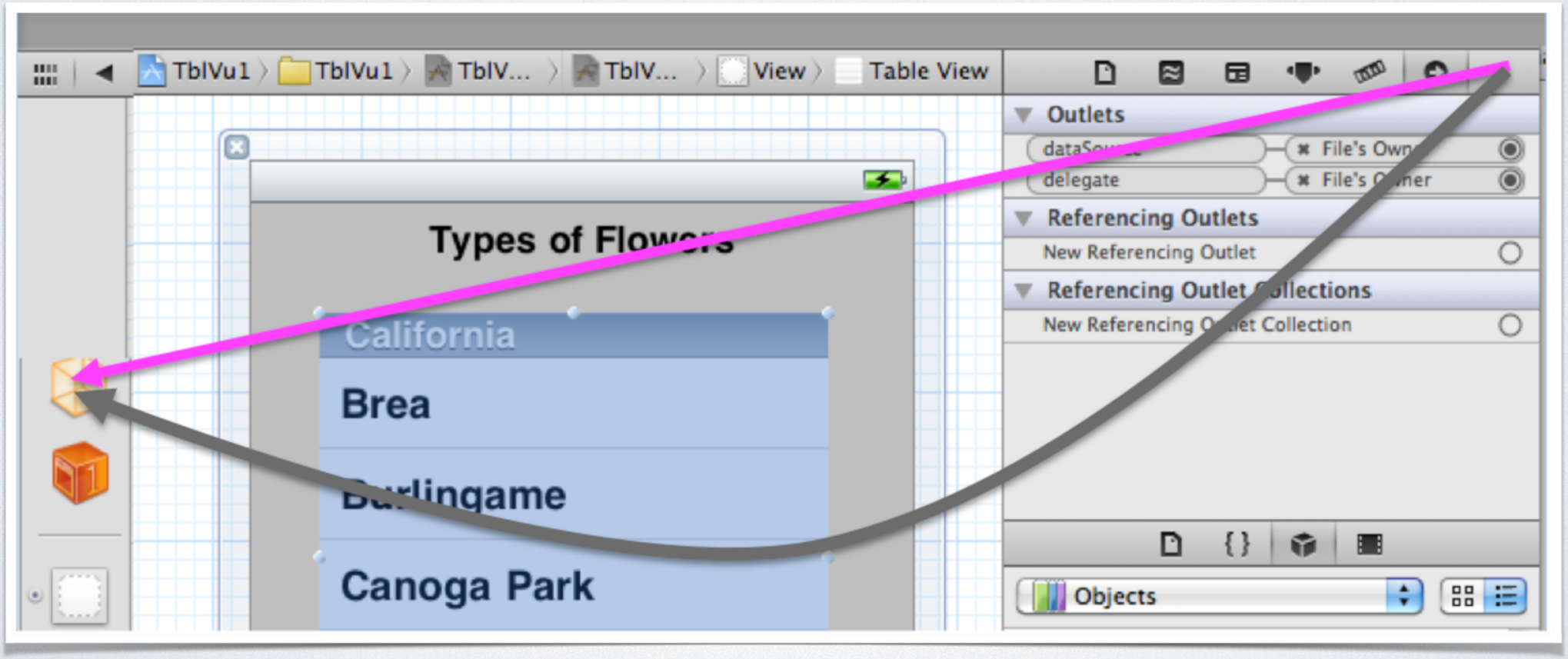
# STEP 2: DRAW TABLE VIEW

- **Open the Main.storyboard file.**
- **Draw a label and change its text to "Types of Flowers".**
- **Now draw a TableView. The system will display some dummy entries (in the design view) as shown below:**



# STEP 3: DATASOURCE & DELEGATE

- **Open the Property Inspector of the Table View, then drag its dataSource and delegate outlets and drop those on the File's Owner.**

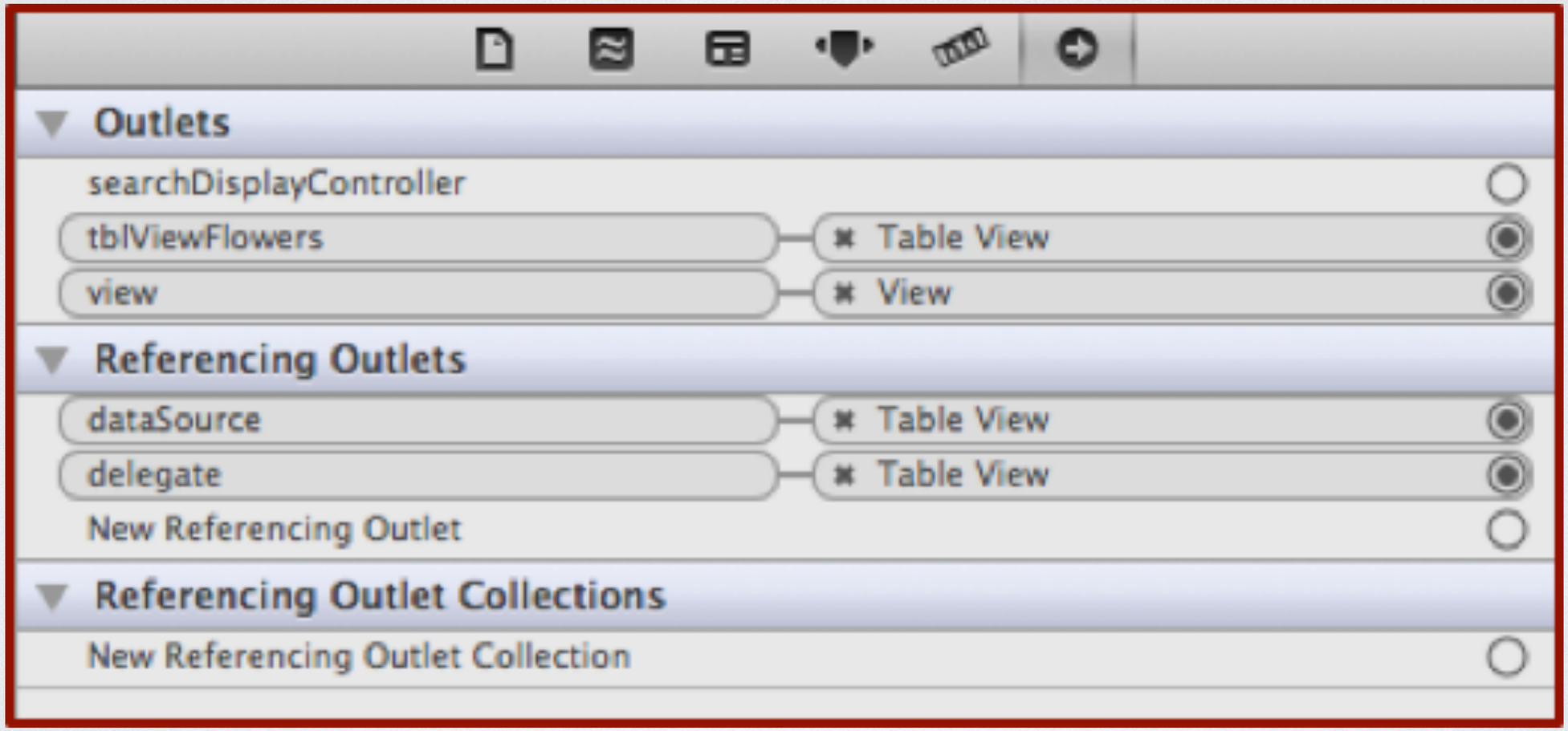



GroundSpeed™
rapid web + mobile software

# STEP 3A: OPEN THE TBLVU1VIEWCONTROLLER.H FILE AND INSERT THE FOLLOWING CODE

```
//  TblVu1ViewController.h
//  TblVu1

#import <UIKit/UIKit.h>

@interface TblVu1ViewController : UIViewController
<UITableViewDataSource, UITableViewDelegate>
{

}

@property (nonatomic, strong) IBOutlet  UITableView *tblViewFlowers;
@property (nonatomic, strong) NSMutableArray *arrayFlowerNames;

@end
```

# STEP 3B: GO BACK TO THE STORYBOARD FILE, AND CONNECT THE TBLVIEWFLOWERSOUTLET TO THE UI TABLE VIEW.



# STEP 4: DEVELOPING THE CODE IN THE IMPLEMENTATION FILE

Just like the UIPicker view, the TableView has numerous datasource and delegate methods. In this section, we will develop and discuss the relevant code.

**4A:** Synthesize the properties.

```
//  TblVu1ViewController.m
#import "TblVu1ViewController.h"
@implementation TblVu1ViewController
@synthesize tblViewFlowers, arrayFlowerNames;
```

**4B:** Populate the arrayFlowerNames array in the viewDidLoad method:

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

   // Populate the arrayFlowerNames array
   arrayFlowerNames = [[NSMutableArray alloc]
   initWithObjects: @"Daliah", @"Lily",  @“Marigold", @"Petunia", @"Rose", nil];
}
```

# 4C: EMPLOY SOME OF THE DATA SOURCE METHODS OF THE TABLEVIEW OBJECT TO SUPPLY THE NUMBER OF SECTIONS AND NUMBER OF ROWS IN EACH SECTION.

The task is easy for us because we have only one section.

```
// How Many sections will we have in the table view?

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1; // we will have only one section in our table entries.
}

// How many rows in each section?
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
   return [arrayFlowerNames count];
}
```

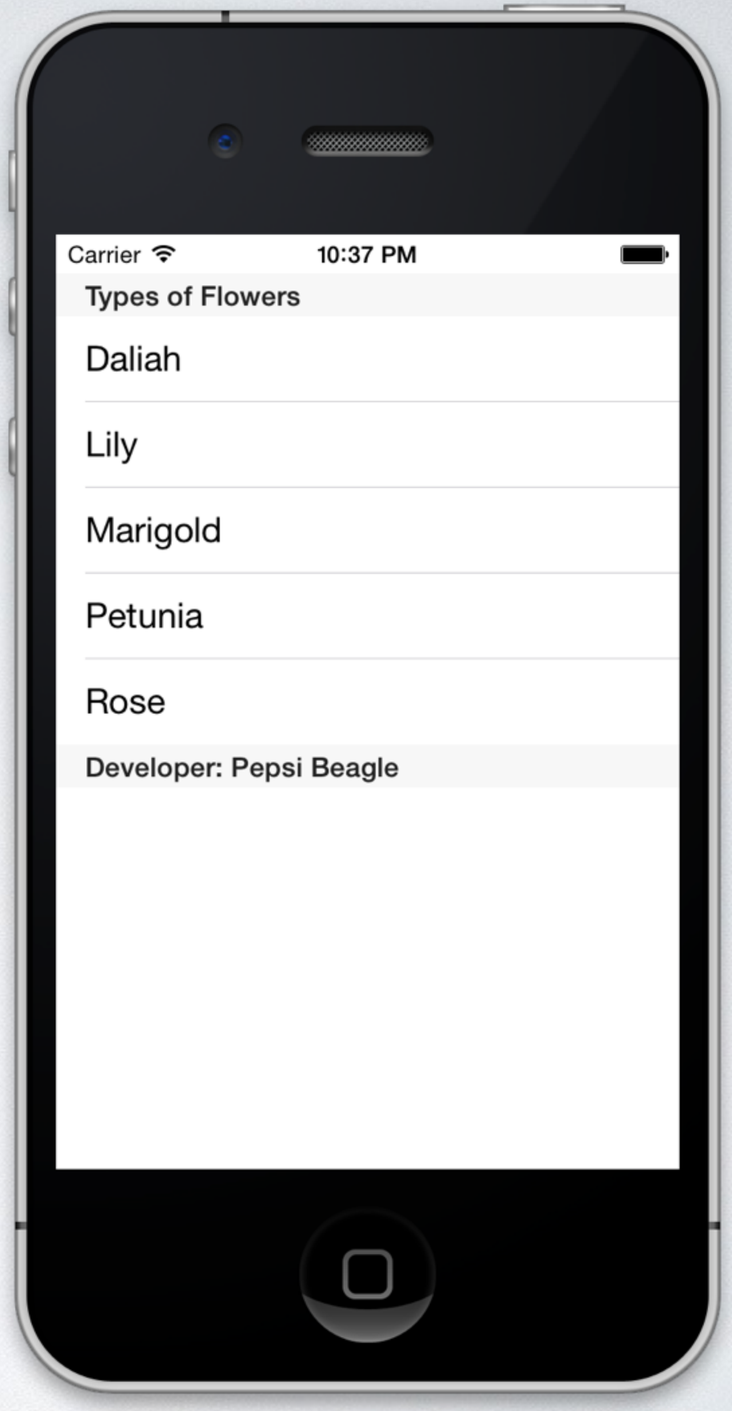
# 4D: POPULATING THE TABLE VIEW. WE CAN USE ONE OF THE DELEGATED METHODS OF THE TABLE VIEW CLASS TO POPULATE THE CELLS OF EACH ROW.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @“theCell";

    // Let us get a pointer to a new UITableViewCell for reusable purpose.

    UITableViewCell *theCell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

    // if it is nil, it must be a new cell. So, let us initialize it with default
    // style and autorelease feature. In case of a large table view, when the user scrolls
    // up or down, we would like to reuse the cell.

   if (theCell == nil) {
       theCell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
   }

    // Configure the cell.
    theCell.textLabel.text =[arrayFlowerNames objectAtIndex:indexPath.row];

    // You can assign the cell.accessory type to a Disclosure symbol, as shown below:
    theCell.accessoryType = UITableViewCellAccessoryNone;
    return theCell;
}
```

# 4E: OPTIONAL HEADER AND FOOTER TITLE OF SECTIONS: YOU MAY IMPLEMENT THE FOLLOWING METHODS FOR THE HEADER AND FOOTER TITLES

```
// Implement the following delegated method to display a header for a section.
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
        return @"Types of Flowers";
}

// Implement the following delegated method to display a footer for a section.

- (NSString *) tableView:(UITableView *)tableView
  titleForFooterInSection:(NSInteger)section
{
        return @"Developer: Pepsi Beagle";
}
```

# YOU ARE DONE!



**Don Miller**

**don@GroundSpeedHQ.com**

**@donmiller**

GroundSpeed™

rapid web + mobile software