Using Xcode Version 4 for Absolute Beginners

Chapter 1

Zach Alexakos

2012

Table of Contents

Chapter 1

INTRODUCTION:

The software used to create iPhone and iPad applications is called Xcode- which is Apple's IDE (Integrated Development Environment). One of the reasons for Apple's success with its mobile devices has been the ability to display applications clearly regardless of the size of the device. The quality of the artwork is also very appealing and this combined with the functionality of the various applications provided has ensured global success and global recognition.

You may have also heard of Android which belongs to Google. The Android SDK (Software Development Kit) is used to create applications for Phones and Tablets that run the Android Operating System. More information can be found on this link: http://en.wikipedia.org/wiki/Android_%28operating_system%29 . Google purchased Android in 2005 and since then has developed a platform and development kit that has exceeded Apple's platform in sales for the first time in 2010.

In Apple's iOS, which is derived from Unix BSD, there are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. Android consists of a kernel based on the Linux kernel, (which is a variant of Unix), with middleware, libraries and APIs written in C.

While both platforms utilise the C programming language in various modified forms, Apple uses Objective C and Android uses C. Knowledge of the C programming language as a result is very useful and while it is not intended to make this course a purely programming set of lessons, it is nevertheless important to understand some basic programming concepts used to create the code that drives mobile applications.

The main goal of this course is to encourage the student to develop applications for either platform however the primary focus here will be to learn to develop applications for the iOS Apple platform.

Zach Alexakos September 2011

CREATING AN APPLICATION:

We could just start pressing buttons and creating any old application but the truth is it probably won't work and will end up becoming an unpleasant experience. In order to create an application that is both useful and satisfying for both the developer and end user we need to follow a number of steps to ensure, (i) the creation of a high quality final product and (ii) achieving the goals and objectives set out in our design.

To start with let's look at our Problem Solving Methodology:

Table 1: Problem-Solving Methodology			
1) Analysis	**2) Design**	**3) Development**	**4) Evaluation**
Solution Requirements	Solution Design	Coding	Strategy
Solution Constraints	Evaluation Criteria	Validation	Report
Scope of Solution		Testing	
		Documentation	

[1]

The Problem Solving Methodology provides a useful starting point for developing an application where the scope and recognition of 'usefuleness' is apparent to both developer and end user. Apple states that a good starting point is to "Create an application definition statement early in your development effort to help you turn an idea and a list of features into a coherent product that people want to own".[2]

This is where our analysis begins. What is the problem to be solved? What do we need to solve the problem? What will prevent us from achieving a solution to our problem? Ultimately what is the scope of our solution?

In order to get the "ball rolling" we need to investigate and write down some ideas to see if they will work. Using a template that includes the sub-steps of analysis will help.

1 VCAA, VCE Study Design, Information Technology, p 16, 2011-2014. http://www.vcaa.vic.edu.au/vce/studies/infotech/softwaredevel3-4.html

2 Apple, iOS Human Interface Guidelines, Chapter 2, p 25, 2011

Template 1: Anaysis Template Detail Sample

Analysis Template

Problem to be Solved	(One sentence statement)		
Application Definition Statement (Required by Apple)	What is the main purpose of the application?		
Scope of the Solution	What are the benefits of the Application? (Efficiency, Effectiveness, Capabilities of Application?) What can the Application do?		
Constraints of the Solution	What can't the Application do? (Consider hardware, software, people, data/informatin constraints)		
Ideas and Features of the Application	For example, if the Application was to be about Stock Market fluctuations then first page to contain list of World Markets to choose from- ie: DOW, ASX, FTSE etc	If the Application is about Fuel Prices then a list of Fuel Types could be appropriate on the first page- ie: PULP, ULP, DIESEL, LPG	If the Application is about Groceries the list page may contain a list of Stores, Prices, locations and so on.
Who is Your Target Audience?	If your Application is about Fuel Prices are you targetting Heavy Vehicle Drivers? Diesel Car Drivers? All drivers? High Mileage Drivers?	Once you've worked out your target audience, choose the top 3 characteristics of this group.	

Template 2: Developer's Analysis Template

Analysis Template

Problem to be Solved			
Application Definition Statement (Required by Apple)			
Scope of the Solution			
Constraints of the Solution			
Ideas and Features of the Application	1) 2) 3)	4) 5) 6)	7) 8) 9)
Who is Your Target Audience?	A) B) C)	1) 2) 3)	

Apple recomends that developers of Applications also include some planning of "features, controls and terminology used" to make the process more efficient. Will the users of the Application want a simple background or a thematic artistic background to their application? Will the Application be jammed full of information or will the content displayed be selective and spread out across the screen?

One of the key features of the iOS devices, including Android and other platforms is the 'uniformity and cohesion of appearance'. In order to achieve this there are a number of "Platform Characteristics" that must be adhered to:

1) Controls should look tappable. Using buttons, sliders, pickers that are contoured and graded to invite touching.
2) Ease of Navigation. Use the Navigation Bar for hierarchical content and the Tab Bar for displaying peer groups of content and functionality.
3) The User Interface or UI should look the part regardless of the device used. iPhone Apps don't look good when simply upscaled to fit an iPad screen. The Application must be created specifically for the device.
4) Make sure the application does what it is advertised to do. Leading users 'off the beaten track' is a great way to receive global negative feedback and low downloads. Make sure your Applications does what it was intended to do.
5) Beware of customisation and stick with the standard control designs. Jailbroken iPhones and iTouches might look good to the Jailbreaker but often they look tacky and are difficult to navigate for other users. The same applies to customised Applications. People are reluctant to learn new processes for dealing with information so if it doesn't look familiar, people are less likely to use it, let alone download it.
6) Let users scroll. Don't squeeze information onto a screen in size 8 font which users will not read. Allow use of the scroll function or create a link page. Remember 3-5 more 'hops' to different pages in an Application and the user will be annoyed and eventually give up on the App.
7) Test your design, Test the UI elements and test the functionality of your Application. Observe your testers or end users to gauge their responses, body language and feedback. Remember the first answer is not always the most truthfull response, you will need to pry and ask, using tact of course, what your end user thinks of your Application.

Table 2: Developer Icon and Image Recommendations and Requirements

Description	**Size for iPhone and iPod Touch (in pixels)**	**Size for iPad (in pixels)**	**Guidelines**
Application Icon **(required)**	57 x 57 114 x 114 (high resolution)	72 x 72	1) -Make sure icon has 90 degree corners. Does not have shine or gloss. -Does not use Alpha Transparency. -Make sure icons have 'visible' backgrounds.
App Store icon **(required)**	512 x 512	512 x 512	
Small icon for Spotlight search results and Settings **(recommended)**	29 x29 58 x 58	50 x 50 for Spotlight Search results 29 x 29 for Settings	
Document icon **(recommended for custom document types)**	22 x 29 44 x 58	64 x 64 320 x 320	iOS can create a Document Icon by default otherwise you can make your own.
Web clip icon **(recommended for web applications and websites)**	57 x 57 114 x 114	72 x 72	As for 1)
Toolbar and Navigation Bar icon **(optional)**	Approximately 20 x 20 Approximately 40 x 40	Approximately 20 x 20	-Use pure white with appropriate alpha transparency. -No drop shadow -Use anti-aliasing
Tab bar Icon	Approximately 30 x 30 Approximately 60 x 60 (high resolution)	Approximately 30 x 30	
Launch Image **(required)**	320 x 480 640 x 960 (high resolution)	For Portrait 768 x 1004 For Landscape 1024 x 768	Supply an image for both Portrait and Landscape orientations.

*PNG Format is recommended for all icons and images. Standard depth bit for icons and images is 24 bits (8 bits for red, green and blue each) and an 8 bit alpha channel.
*Alpha transparency can be used in icons created for navigation bars, toolbars and tab bars it is not to be used in application icons.

THE DESIGN STAGE
CREATING A PROTOTYPE:

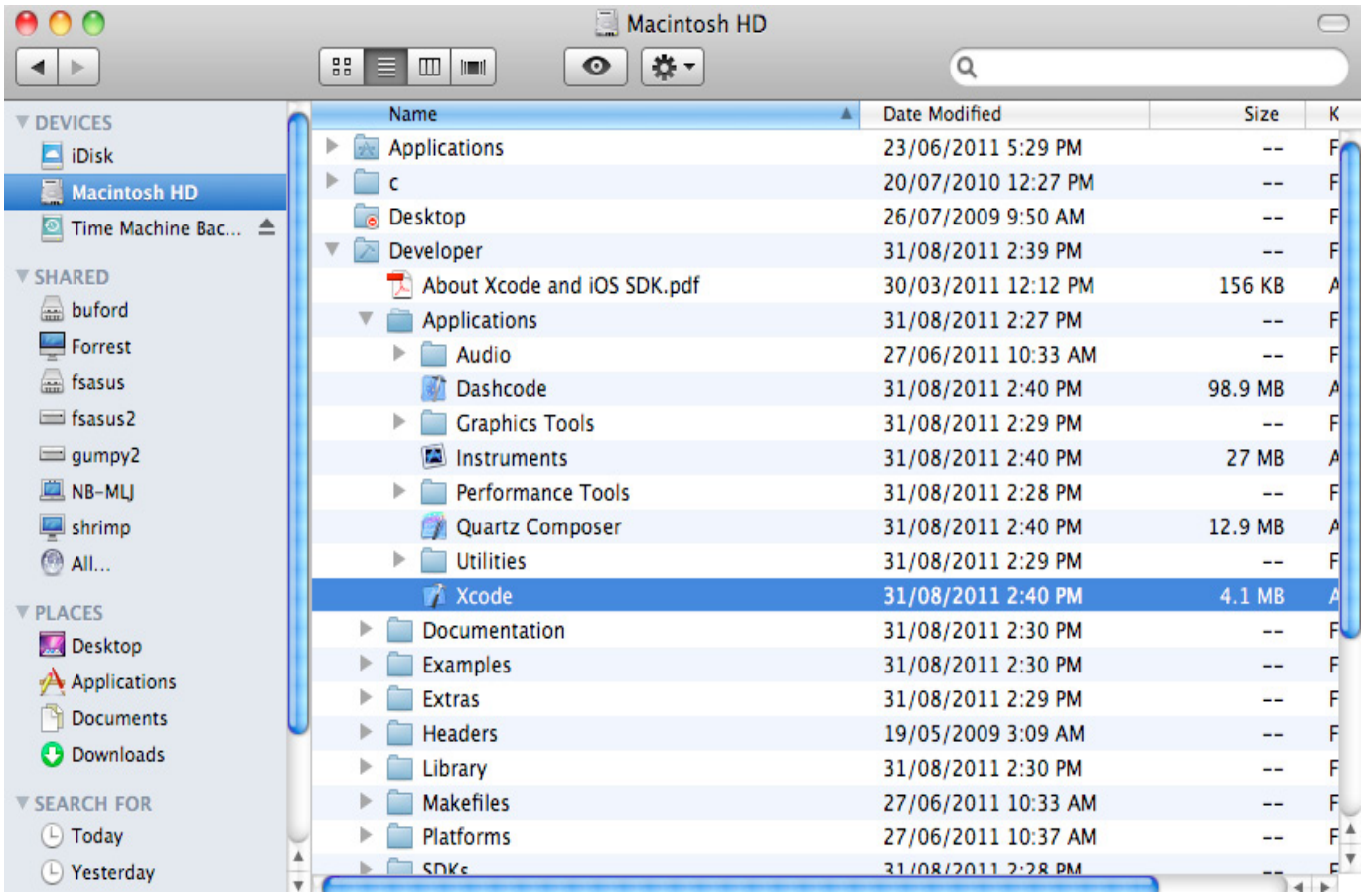
Once you've worked out what it is that you will be creating it is usually a good idea to create a Prototype using the Xcode Templates to build a basic version of your Application. Ask other students or developers to use your 'template application' to gain a better understanding of what works, what doesn't and how to improve the application's look and feel. The useful aspect of design for an iPhone or iPad Application is the ability to design using the existing device and Xcode software simulator.

However before starting your own project it is important to begin with an understanding of the interface and features of Xcode. Once you have familiarised yourself with the features, layout and coding required in Xcode you will be on your way to creating high quality applications. As a result we will devote a significant amount of time with learning the basics of using Xcode, including using simple code for user inputs and screen outputs.

YOUR FIRST APPLICATION- "HelloWorld"

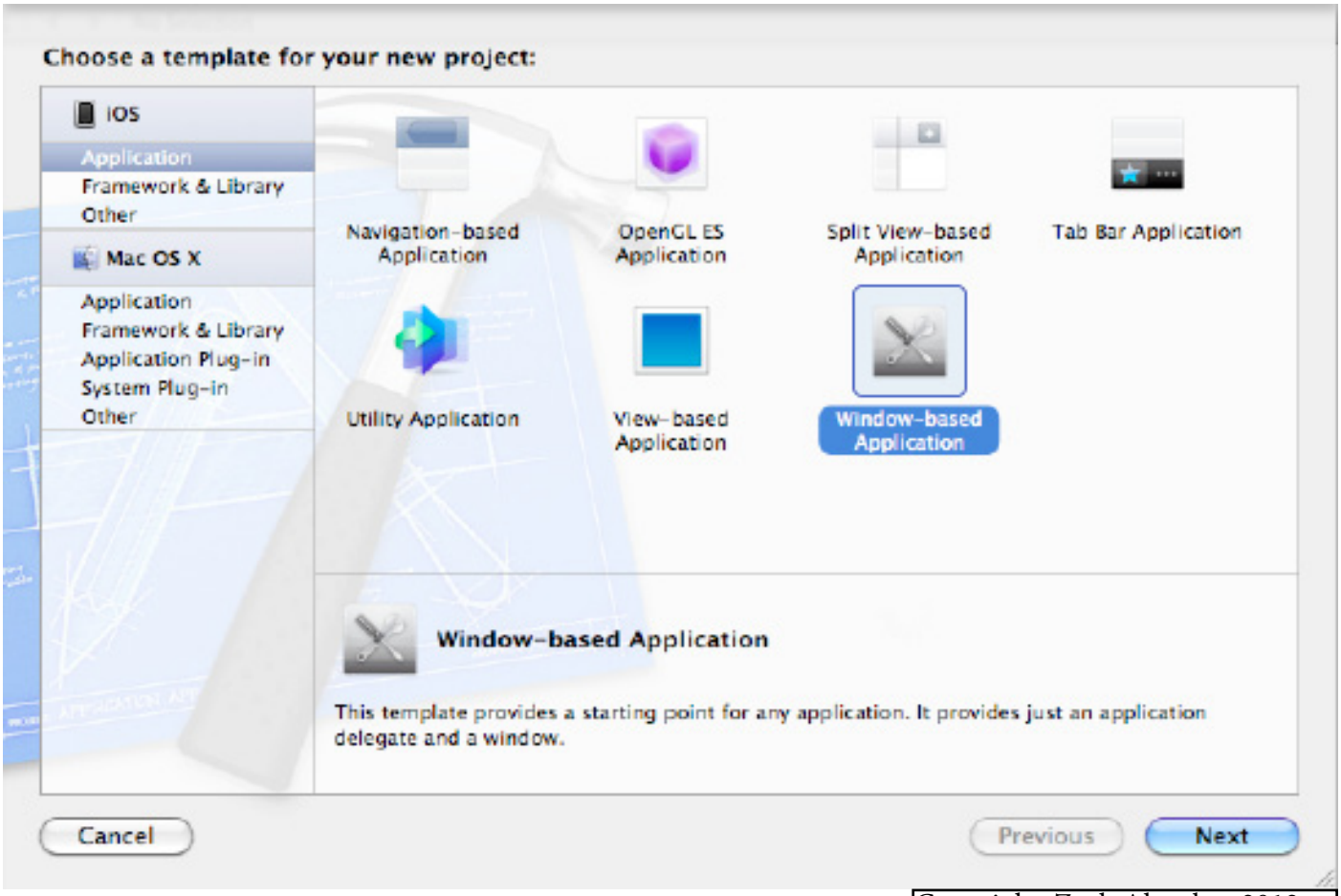
Let's start with opening the Xcode application. Follow the image steps provided:

1) Find and Click on the Xcode application on your Macintosh HD drive in Developer.

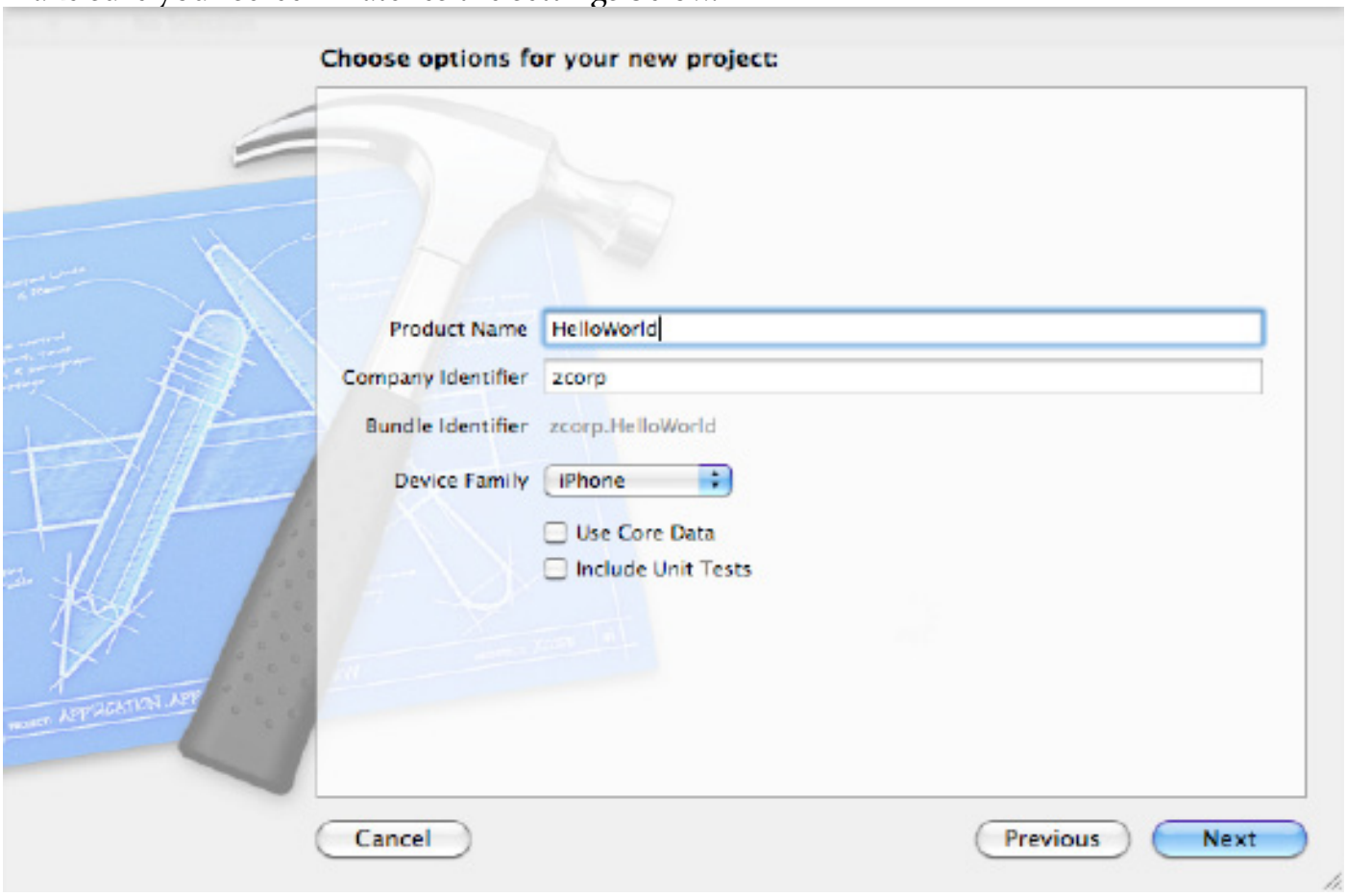


2) Once the Xcode welcome window is open click on 'Create a new Xcode project'.

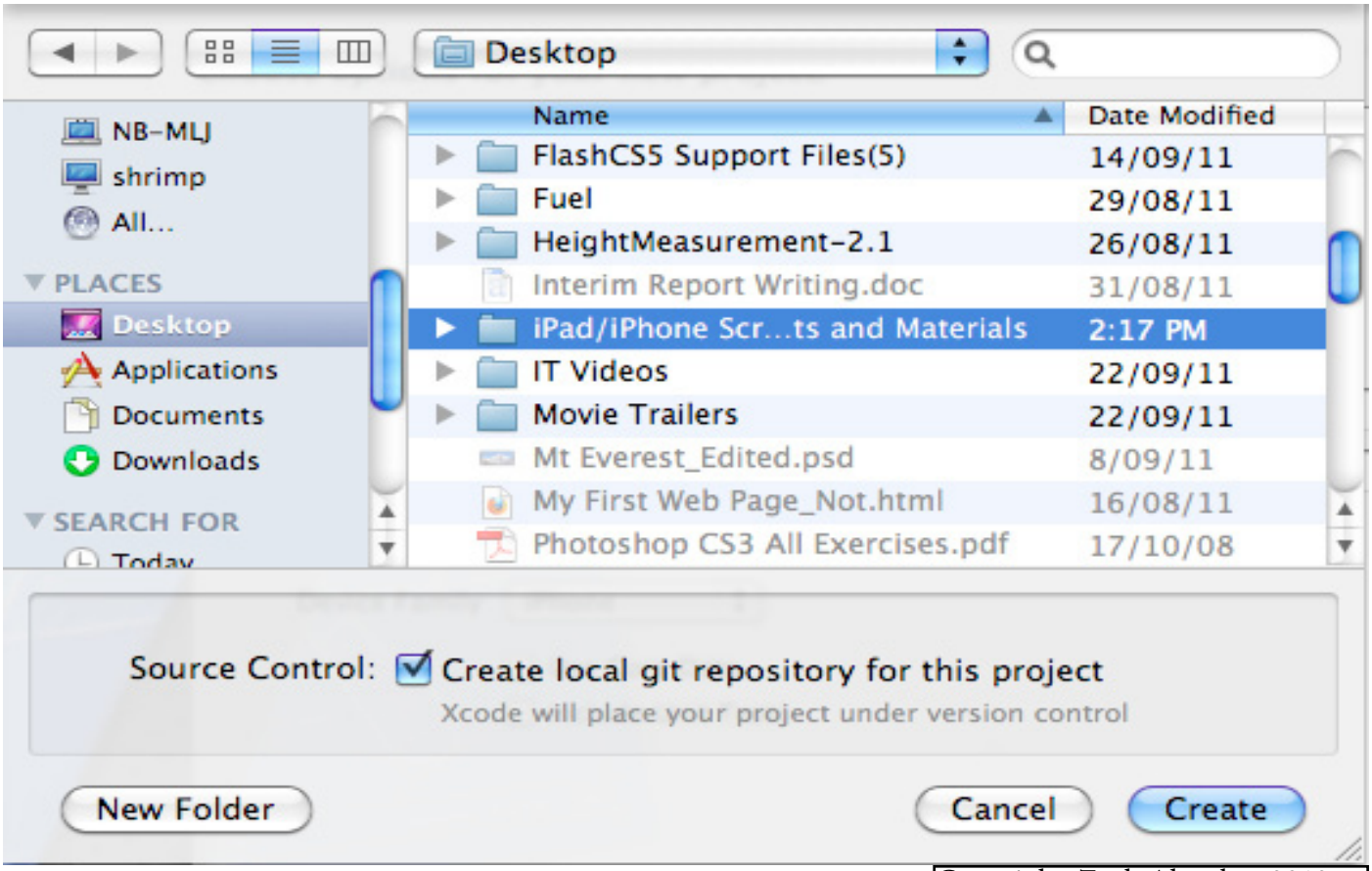
3) Now click on 'Window-based Application'. Also take note of the additional templates included.



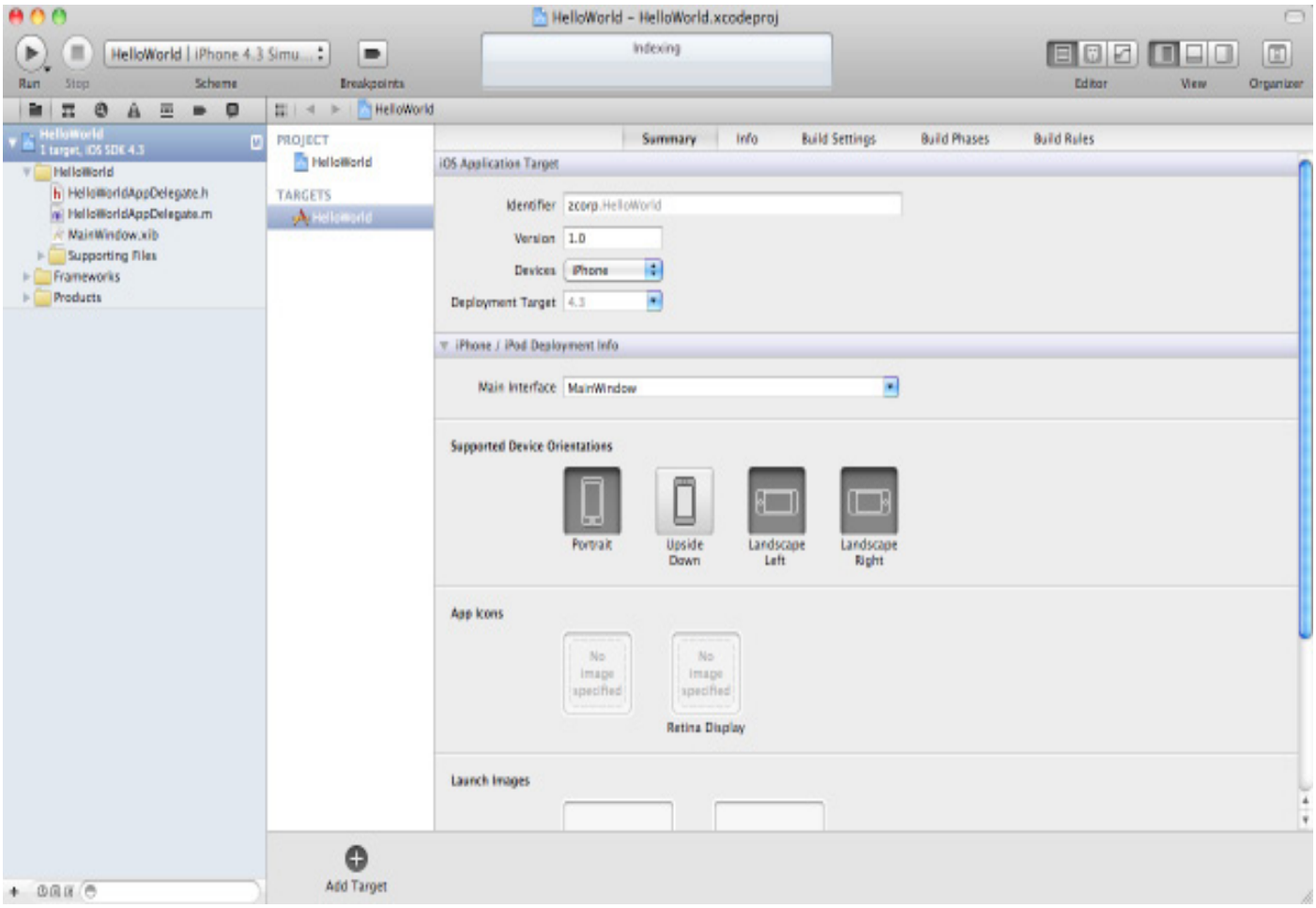
4) We are making an Application titled 'HelloWorld'. Type this in the Product Name and make sure your screen matches the settings below.



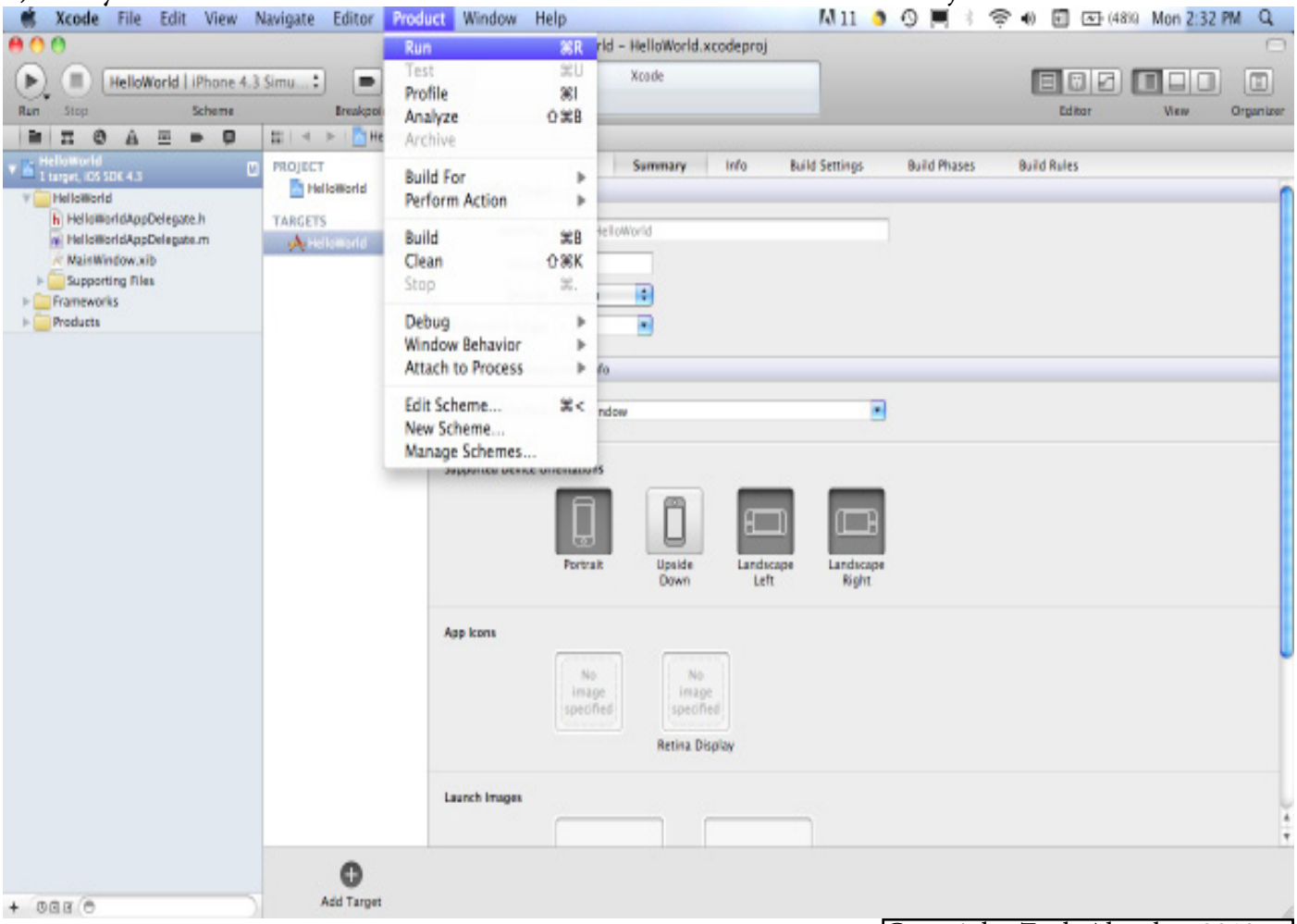
5) Click 'Next' and Create a Folder called iPad/iPhone Files. Save the File 'HelloWorld' in the Folder.



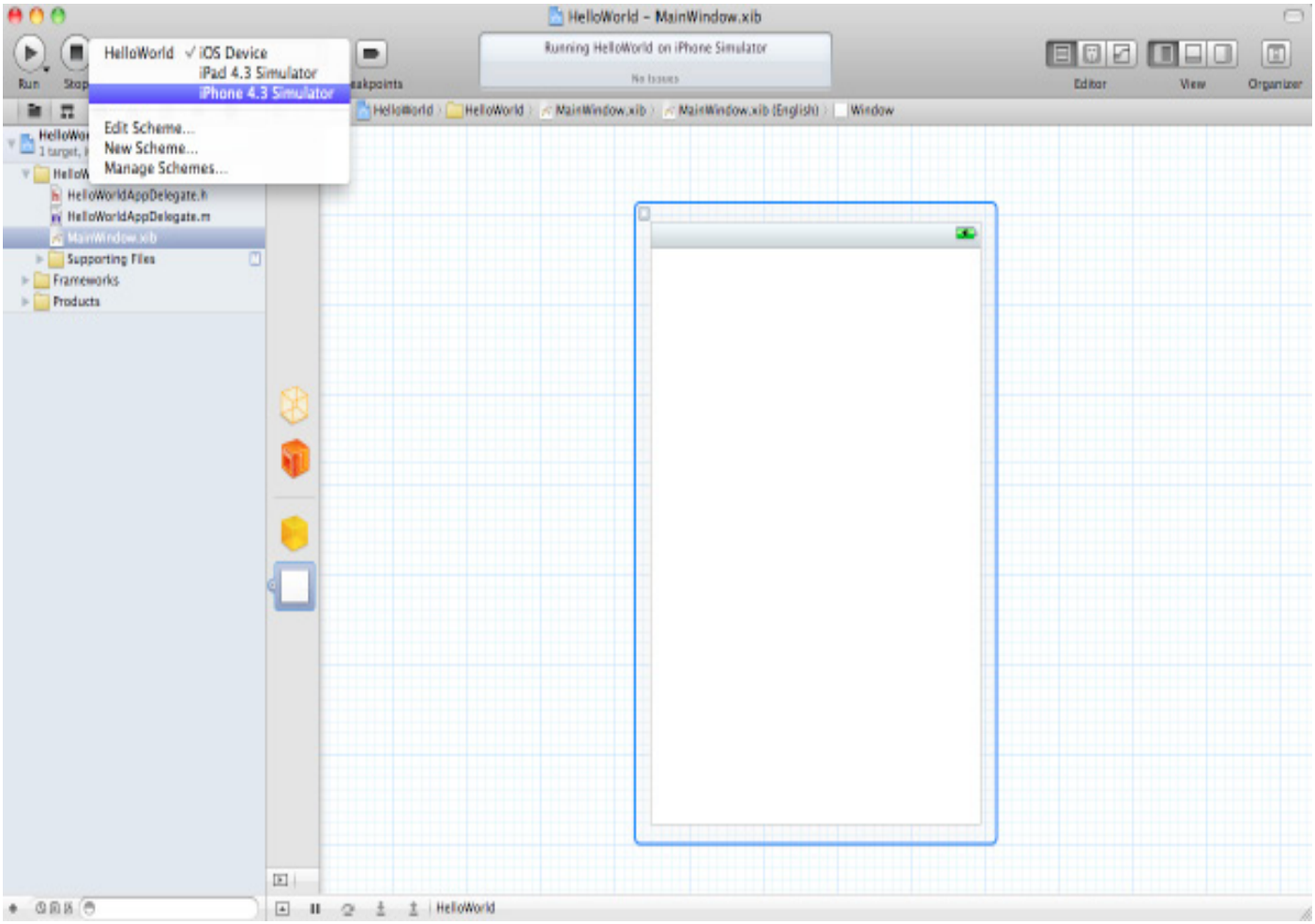
6) When you click the 'Create' button the following screen should appear.



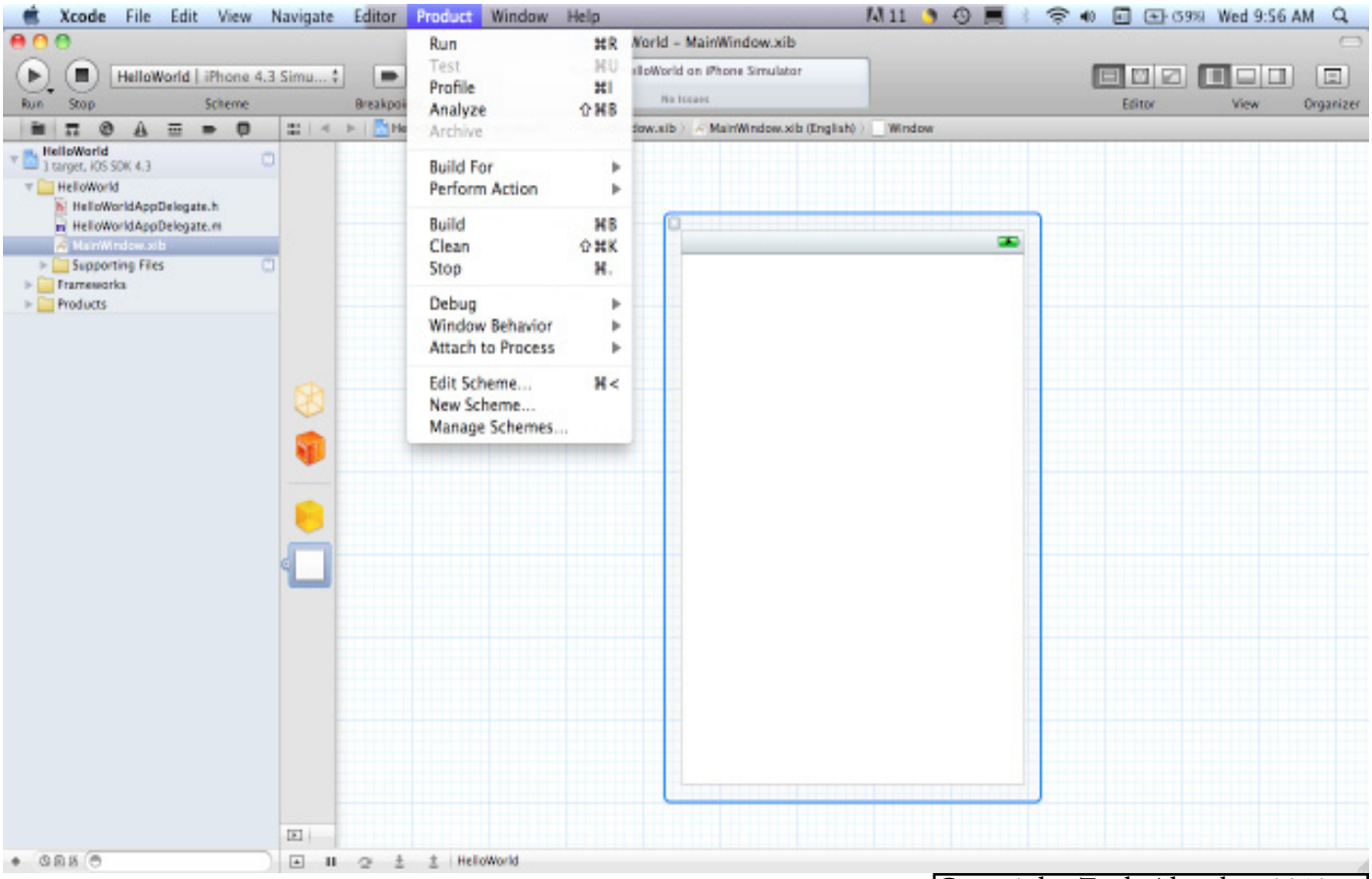
7) Study the features of the Xcode interface and then click on the Project Menu. Select Run.



8) The following screen will appear. This is your Design Window. On the Top Left Click on the Drop down list and select iPhone 4.3 Simulator.



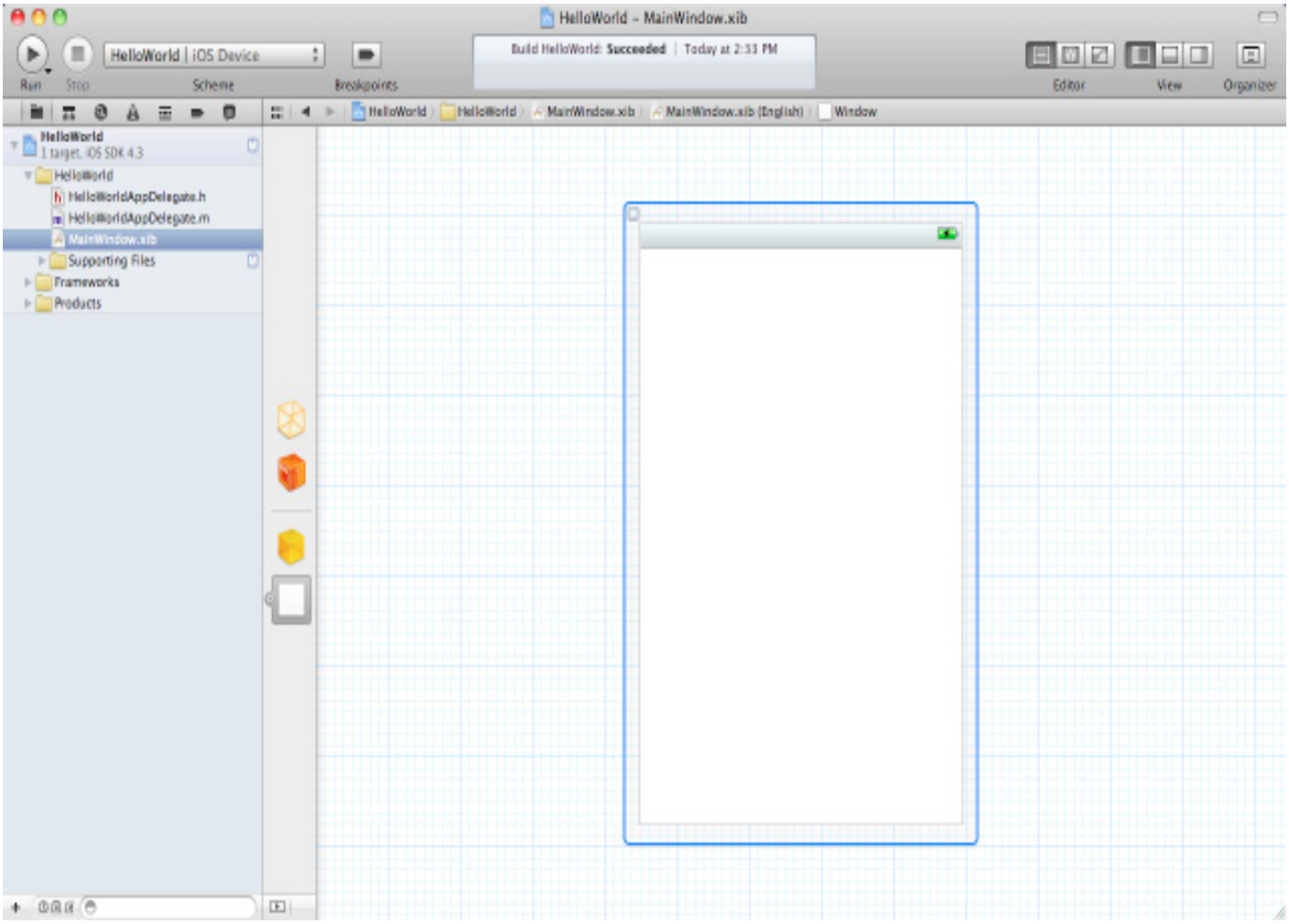
9) Now select the Product Menu and Click Run.



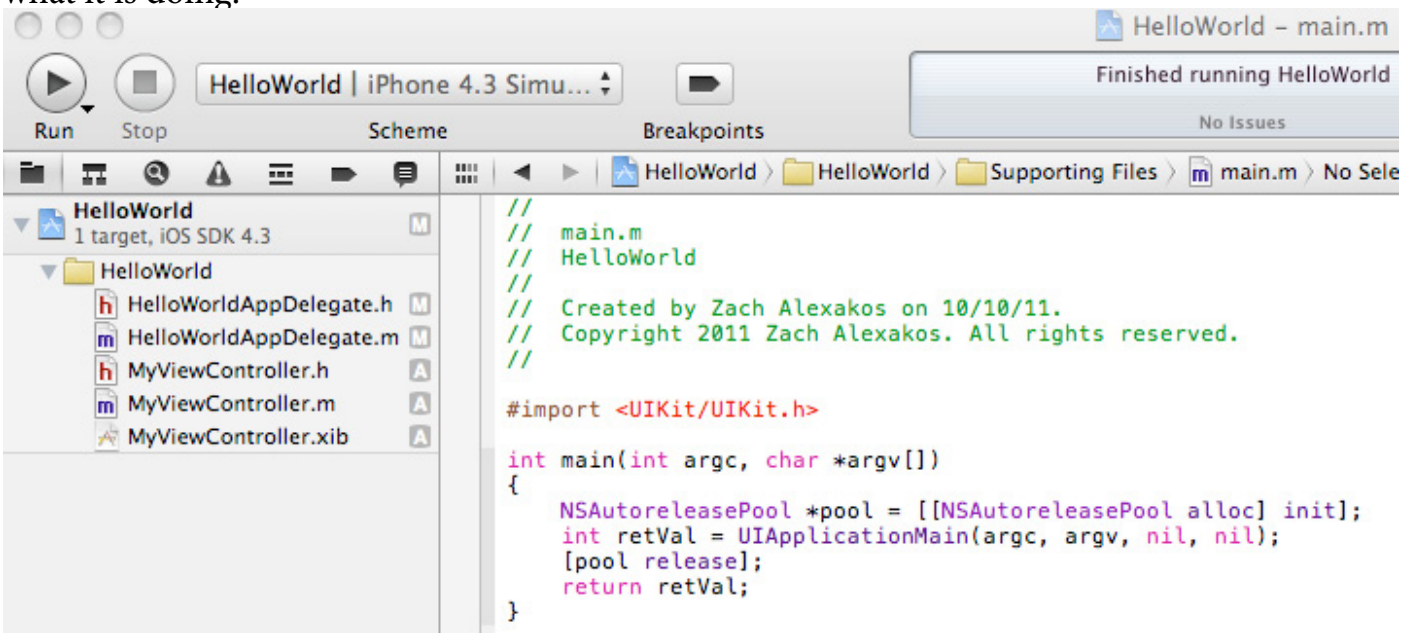
10) and 11) The following screen will appear. You can switch pages using your mouse pointer.

12) Quit the Simulator.

13) Now we are back to our Design View.



14) Double Click on 'main.m'. The following code will appear. Study the code to understand what it is doing.

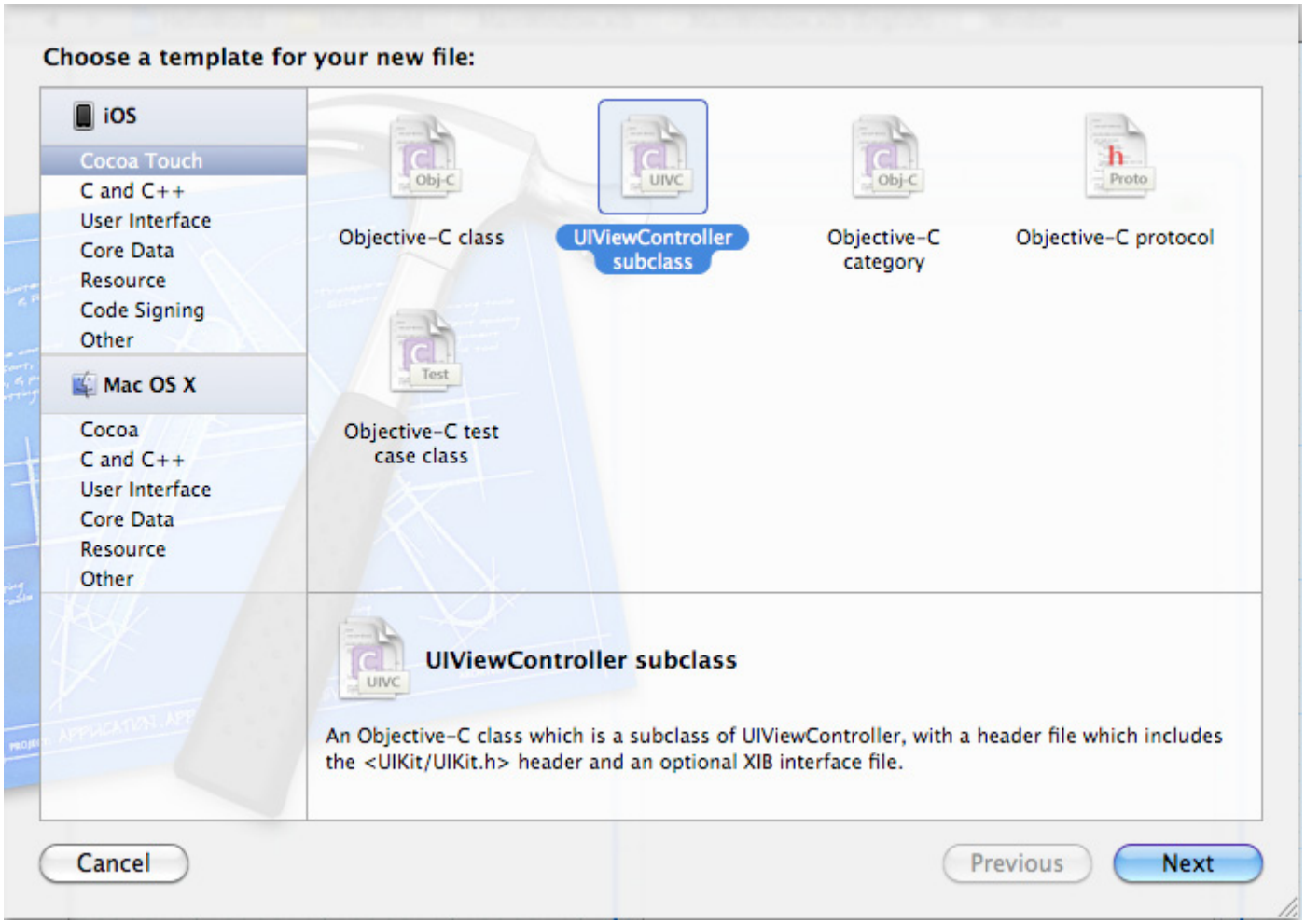


As shown in screenshot No. 14 we have created a instance of the Hello World Program, without inputs or outputs. The program runs but it doesn't do anything. What we now need to do is set it up to output some kind of message and permit the user to manipulate the message displayed.

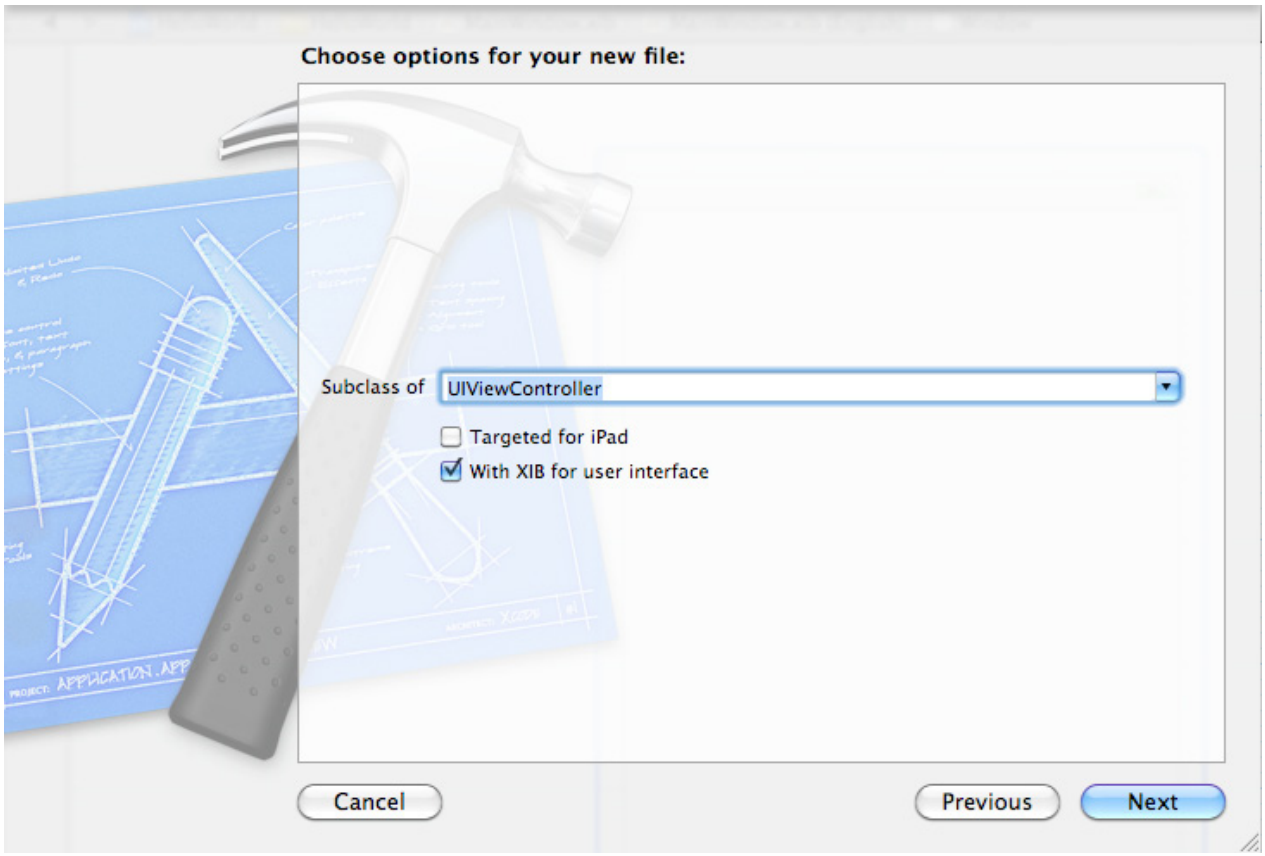
ADDING A VIEW CONTROLLER- "HelloWorld"

As the name suggests the View Controller is the screen the end user sees when loading an applcation on their device whether the device is an iPhone or iPad. As a developer it is a very important part of the application design and development process, enabling the developer to setup, view and modify the screen setup. The View controller also allows the designer to setup the navigation and memory management. To create a View Controller for our 'Hello World" application follow the sequence of window steps below.

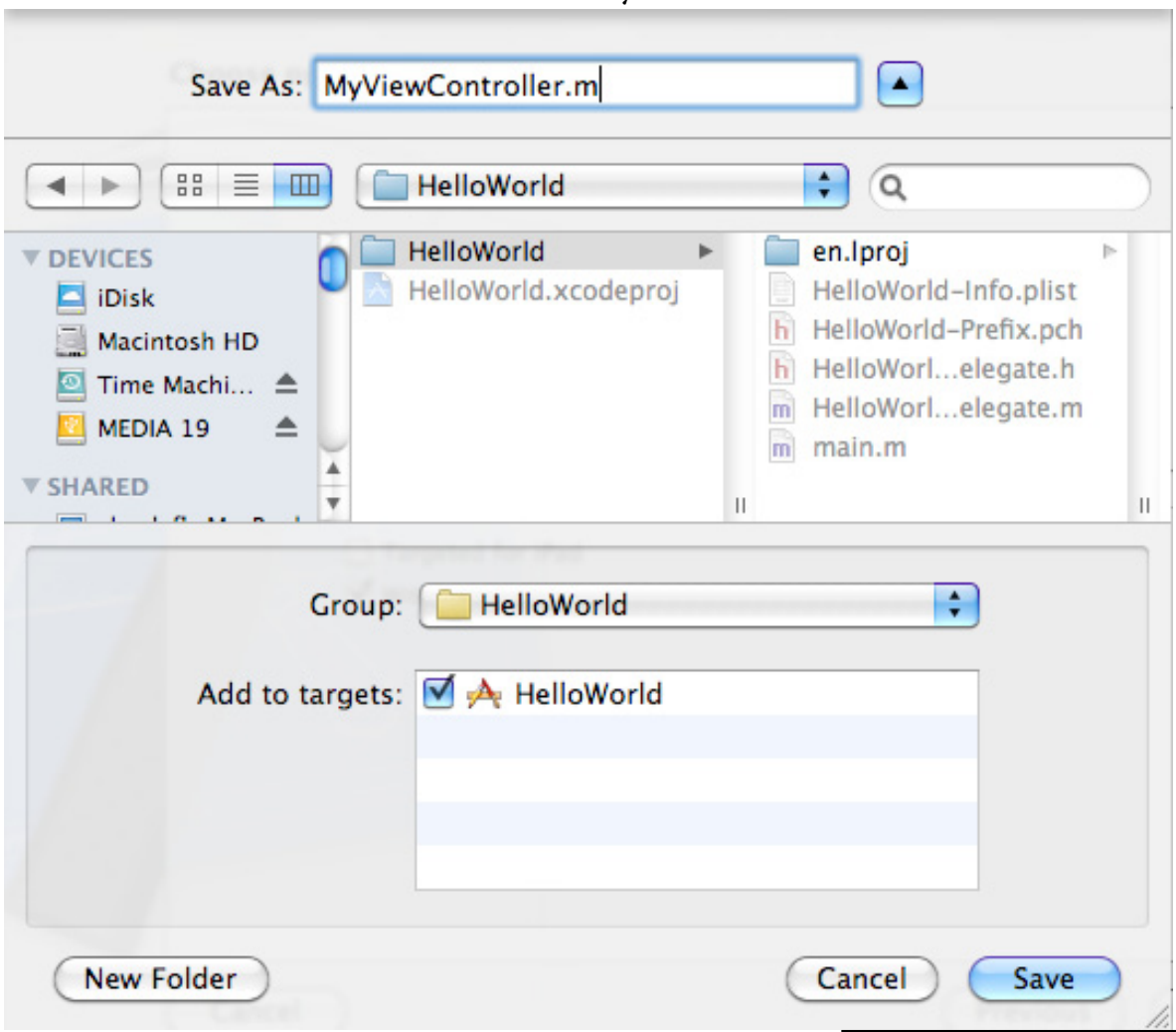
1) Click the File menu in Xcode and select File- New- New File. The following window will appear.



2) The following window should appear. Ensure that the "With XIB for user interface" is checked. Click Next.



3) Type in Save As as shown below and save into your Hello World Folder. Click Save.



4) The following window should appear. (Below is the exact code needed for your program to work. Don't be too concerned if you have different code on your screen at the moment). Leave as is and proceed to the next step.

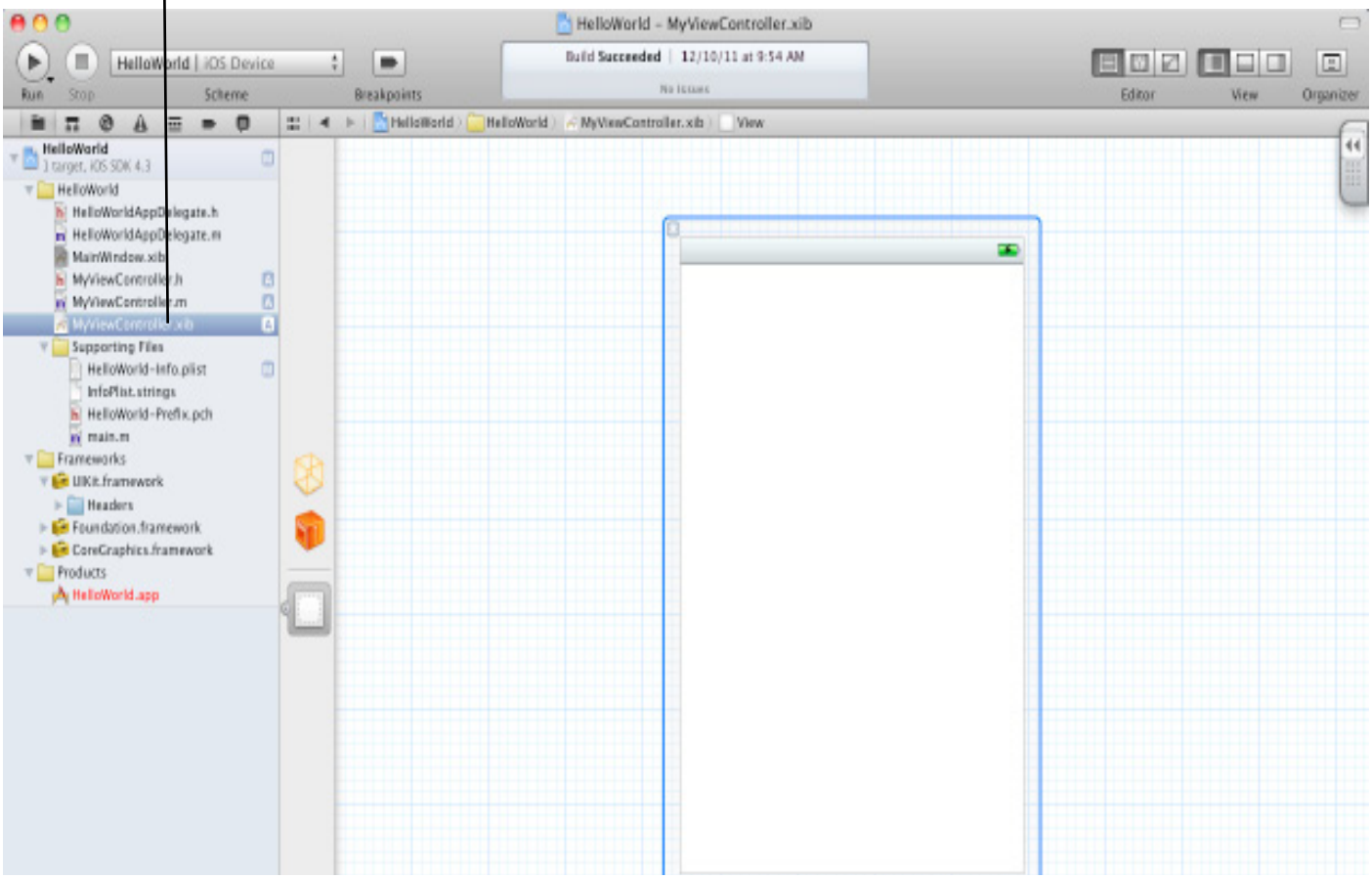
HelloWorld 〉 HelloWorld 〉 MyViewController.m 〉 No Selection

```
//
//  MyViewController.m
//  HelloWorld
//
//  Created by Zach Alexakos on 17/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import "MyViewController.h"

@implementation MyViewController;

@synthesize label=_label;
@synthesize textField=_textField;
@synthesize userName=_userName;


- (void)dealloc
{
    [_textField release];
    [_label release];
    [_userName release];
    [super dealloc];
}

- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
    if (theTextField == _textField) {
        [theTextField resignFirstResponder];
    }
    return YES;
}

- (IBAction)changeGreeting:(id)sender {
    self.userName = self.textField.text;

    NSString *nameString = self.userName;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
         NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
    self.label.text = greeting;
    [greeting release];

}
@end
```

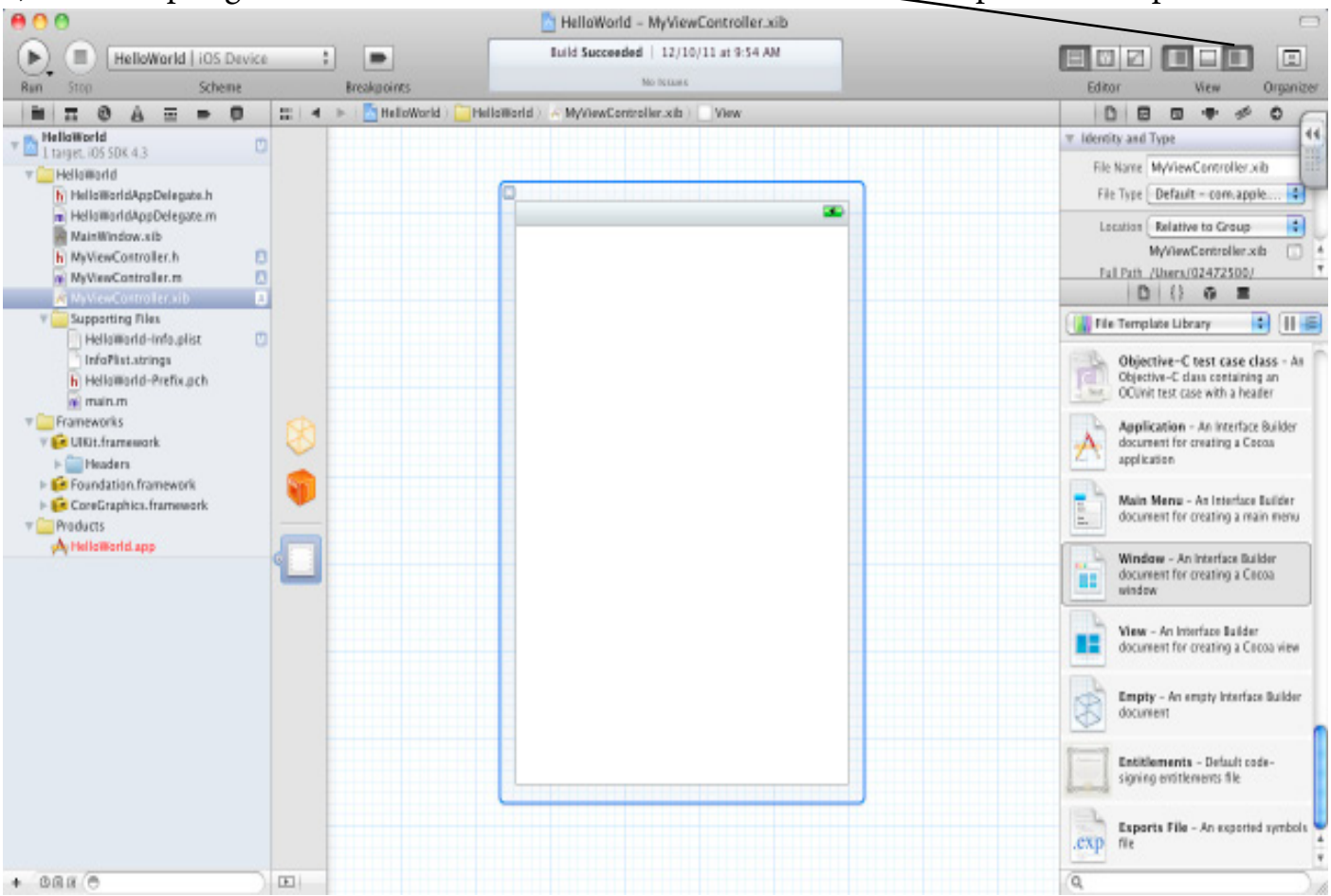
5) Click on the HelloWorldAppDelegate.h file and study the code.

```
//
//  HelloWorldAppDelegate.h
//  HelloWorld
//
//  Created by Zach Alexakos on 10/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import <UIKit/UIKit.h>

@class MyViewController;

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate> {
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) MyViewController *myViewController;

@end
```

6) Now Click on the .xib file MyViewController.xib. You should see The View Based Application Template in your window. (Note: this view can also be accessed when creating a new file).

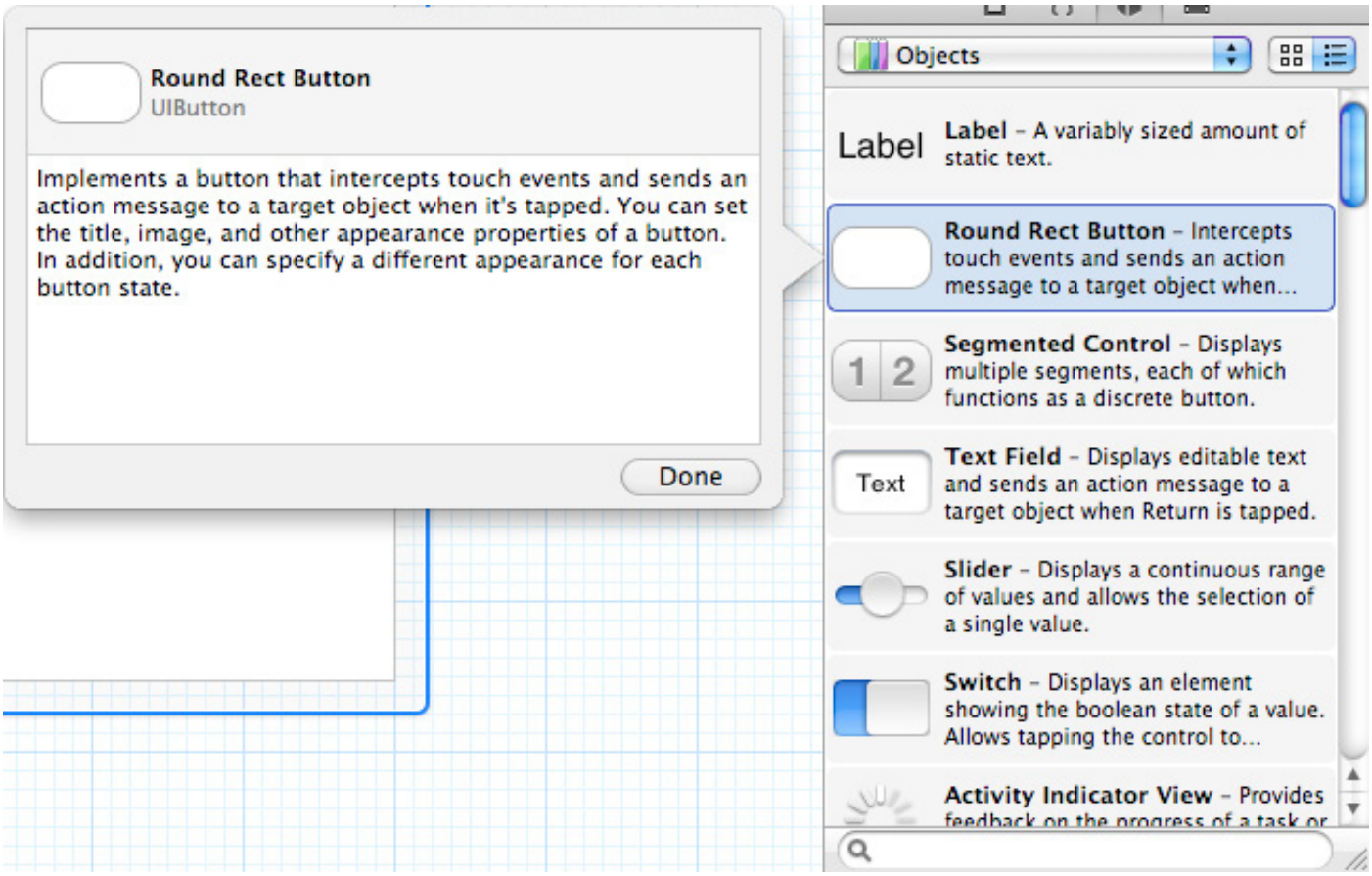


7) In the Top Right side of the window select the third View icon to open the Templates.

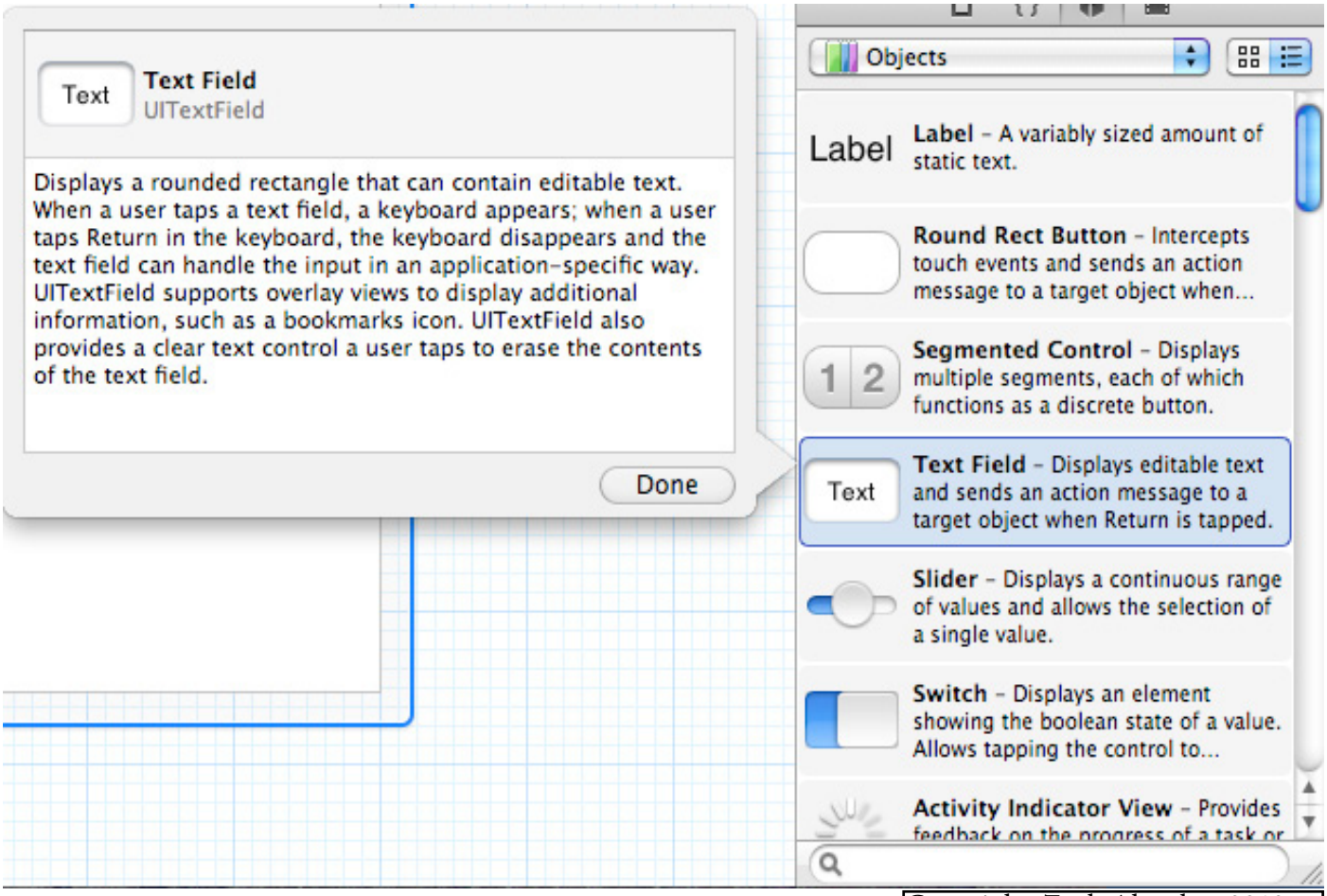


8) We will now add a button (UIButton), a text field (UITextField) and a label (UILabel) to the controller's nib file. The three different objects are shown in the following screenshots.

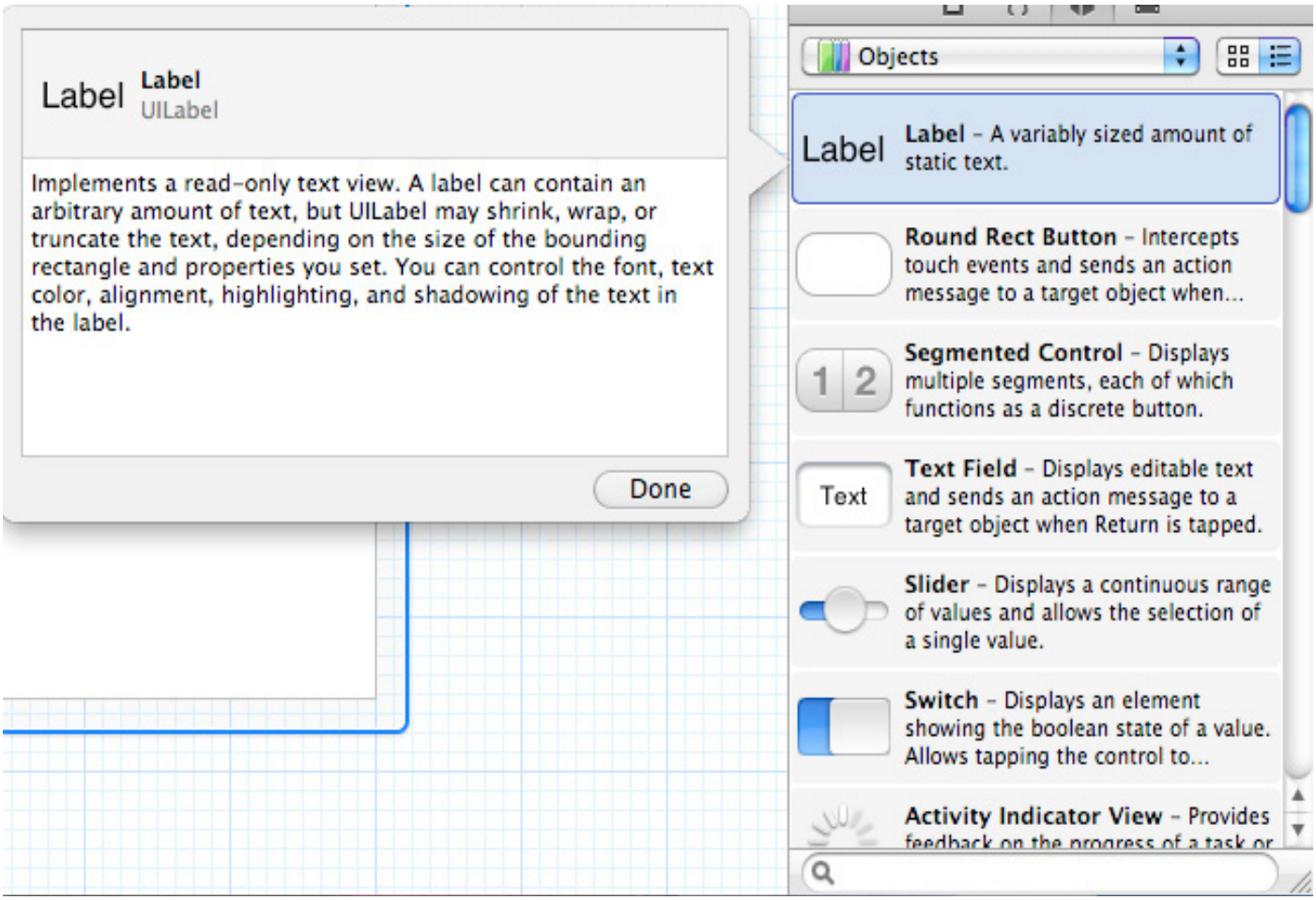
The UI Button.



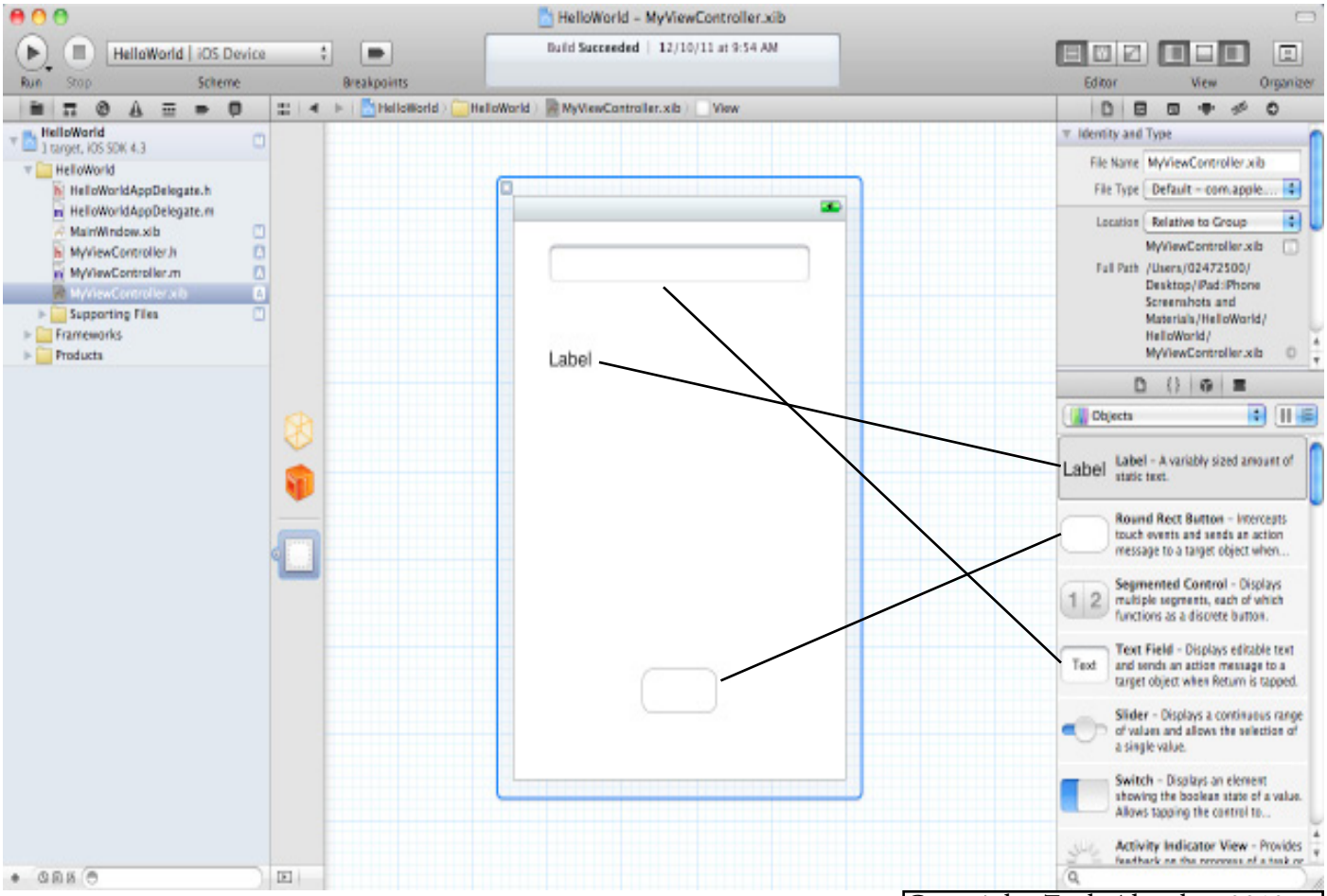
The UI Text Field.



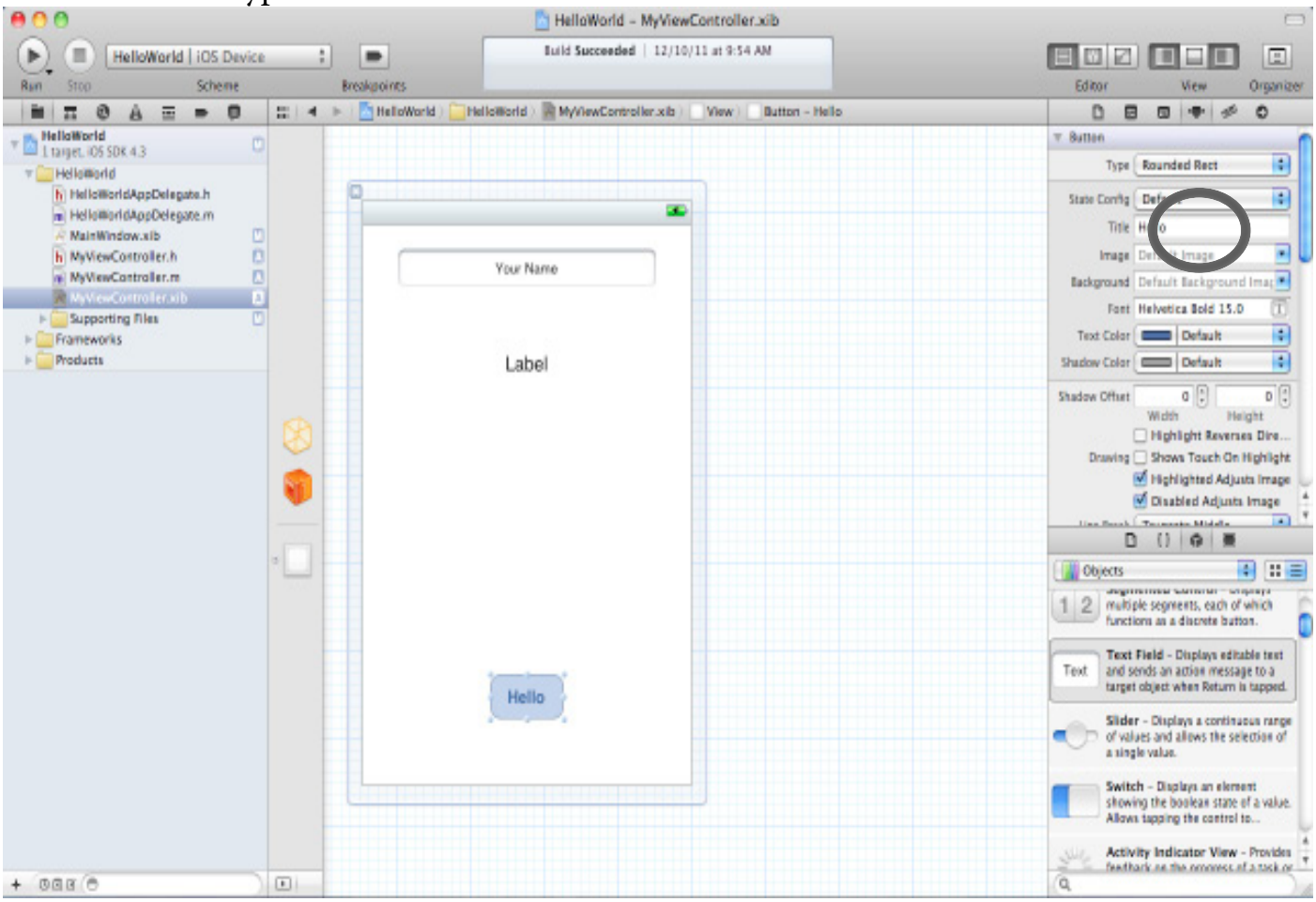
The UI Label.



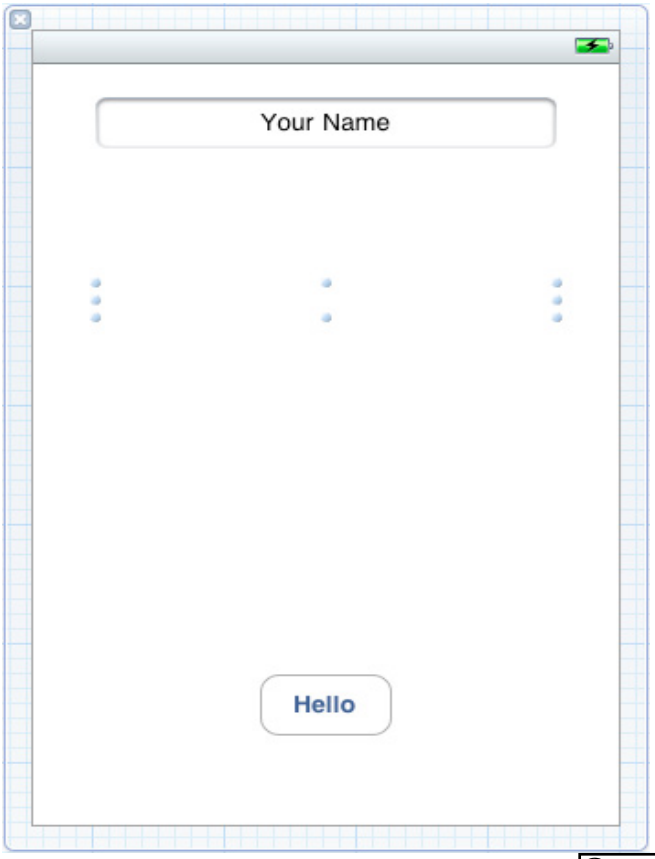
9) Drag the three UI Fields



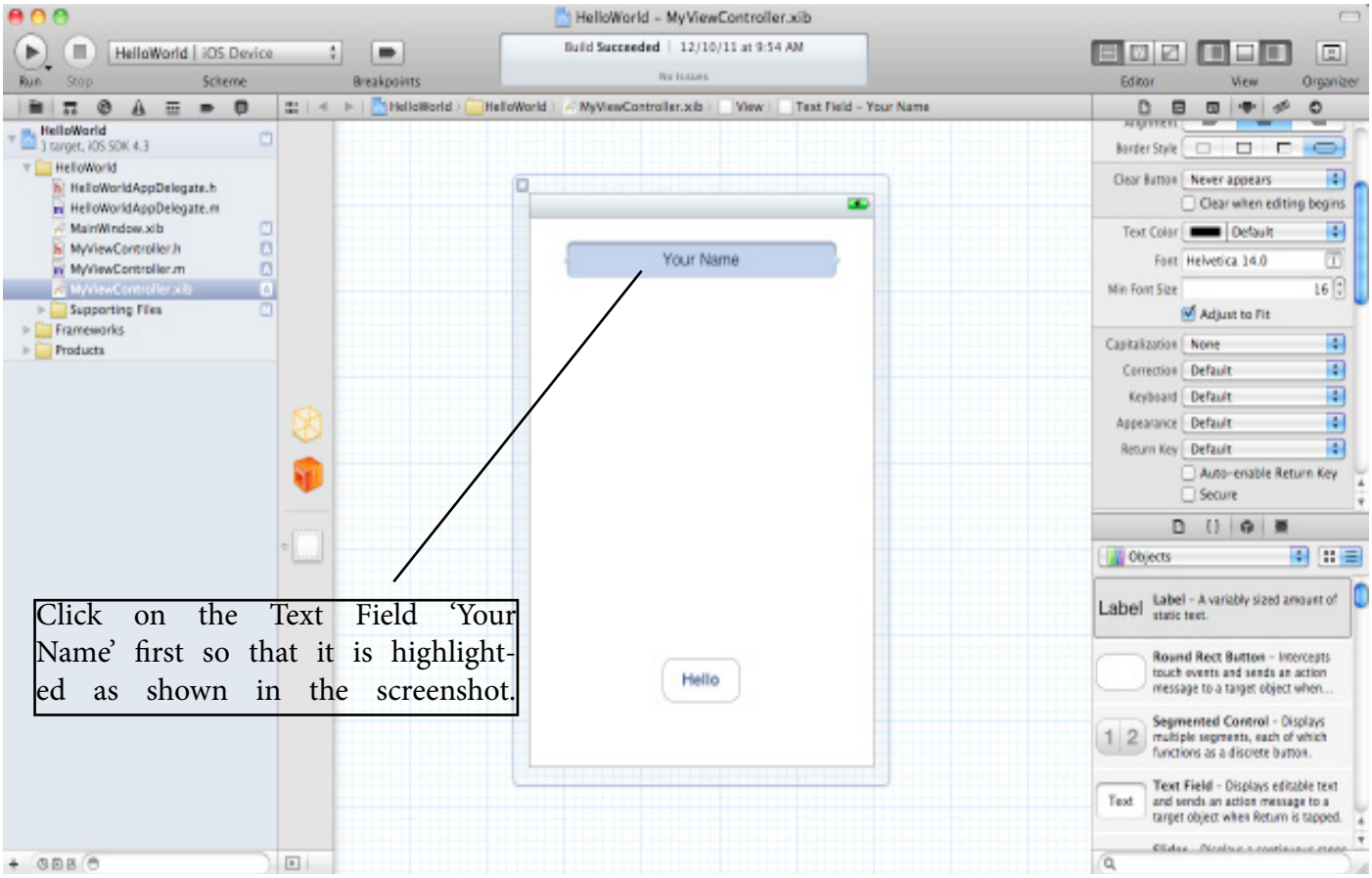
10) Drag the UI Fields to a location on the screen. Make sure you select the View icon in the Assistant Editor section. Type in 'Your Name' in the Text Box, Delete the 'Label' tag in the Label Field and Type 'Hello' in the UI Button.



11) Centre Align your text in each of the UI Fields. Set the Font Size to 14pt for all text. Your screen should look like the one below.

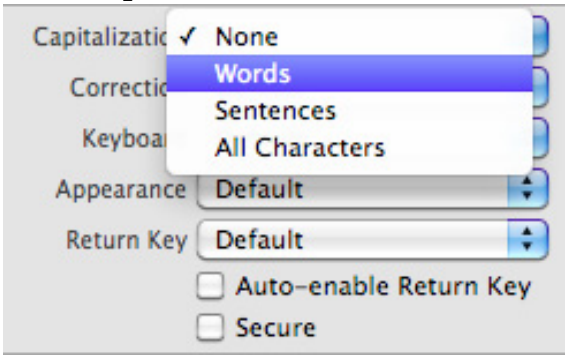


12) The next step is to set up the attributes for the Text Field. For example, if we want to capitalise the text being entered we need to select it from the drop down list. Follow the screen shots below to set up your Text Field attributes.

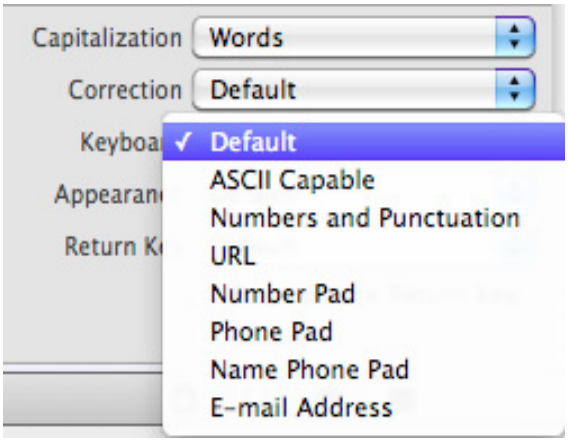


13) Then move to the right of the screen and in the Attributes Inspector select the following Text Field traits for the Text Field 'Your Name'.

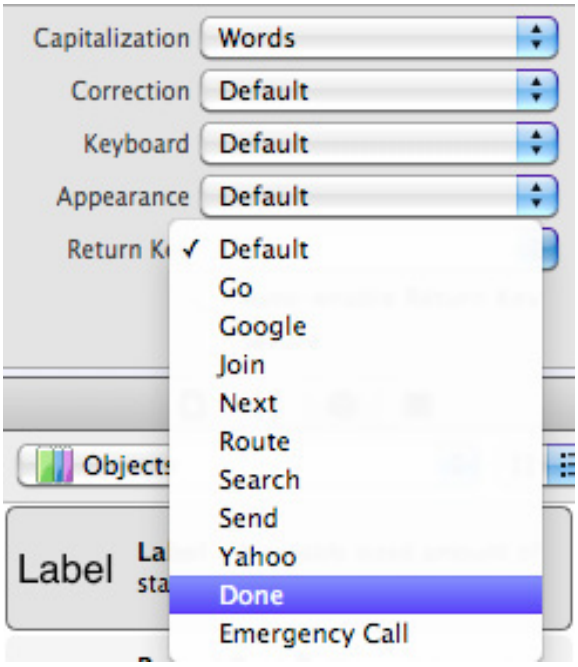
a) In Capitalisation Select 'Words'



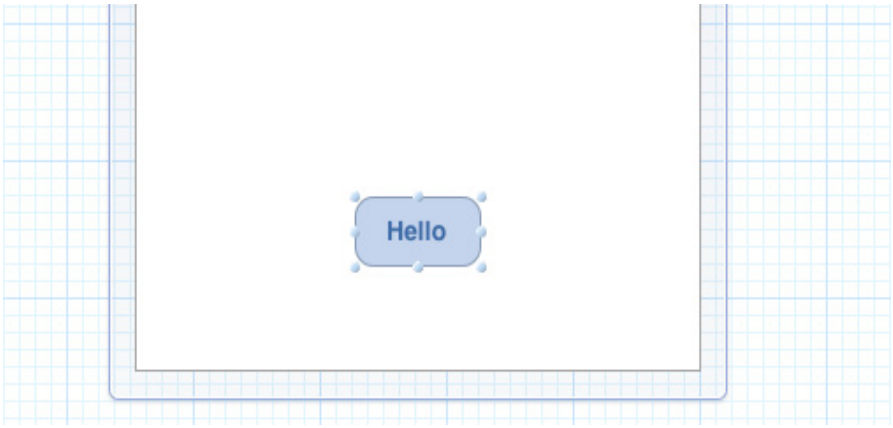
b) In Keyboard Select 'Default'



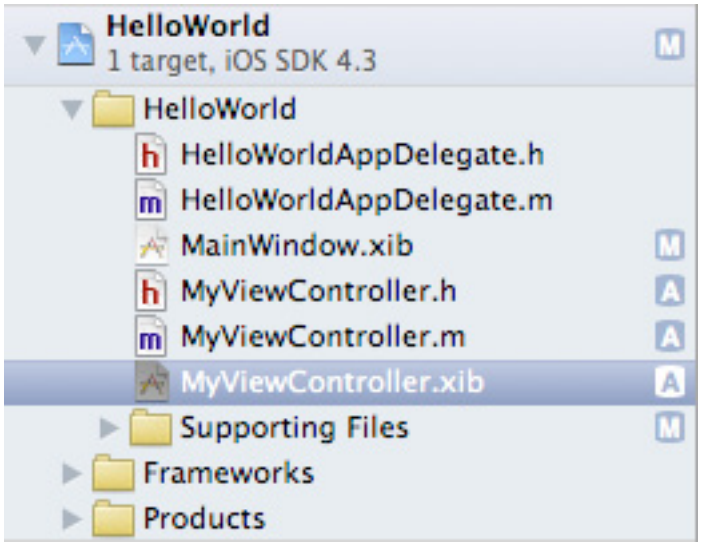
c) In Return Keyboard Select 'Done'.



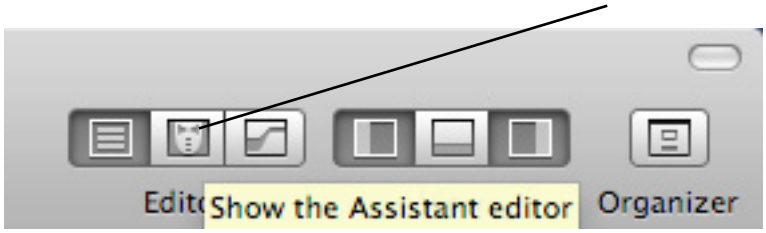
14) Now we want to make our 'Hello' button perform an action when pressed. Click on the 'Hello' button as shown below.



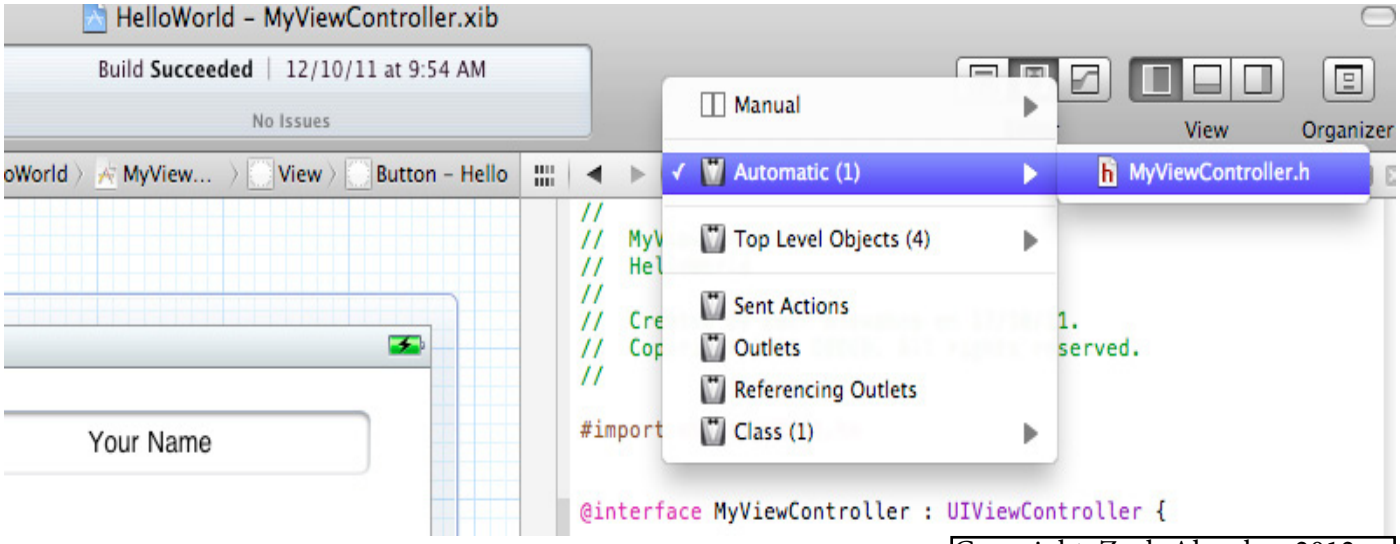
15) Now make sue the MyViewController.xib file is selected in the Project Navigator.



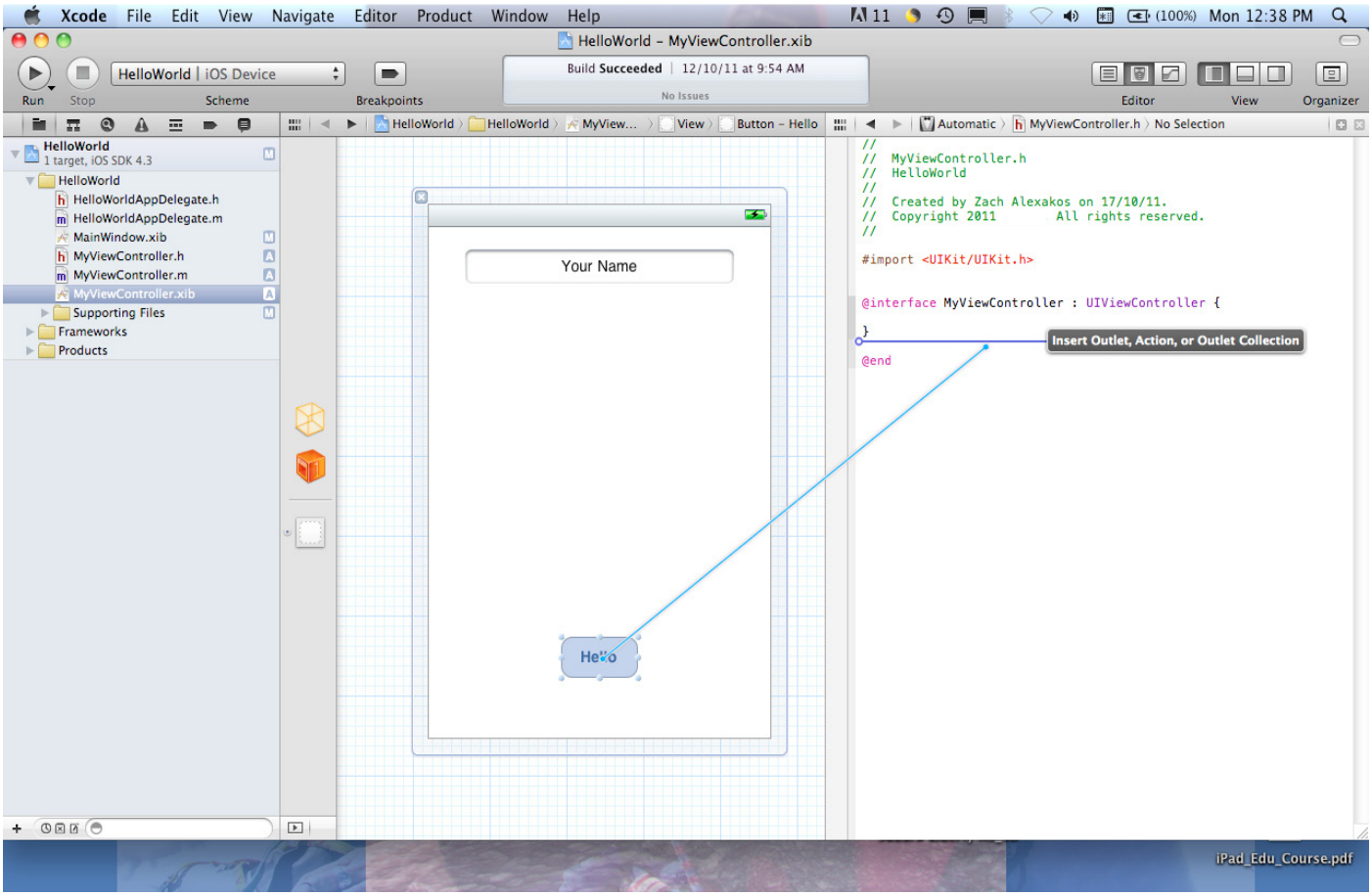
16) Now click and select the Assisstant Editor.



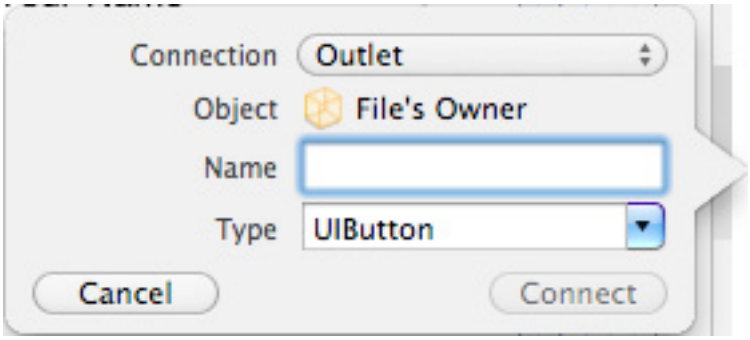
17) Make sure the 'Automatic' / MyViewController.h is selected.



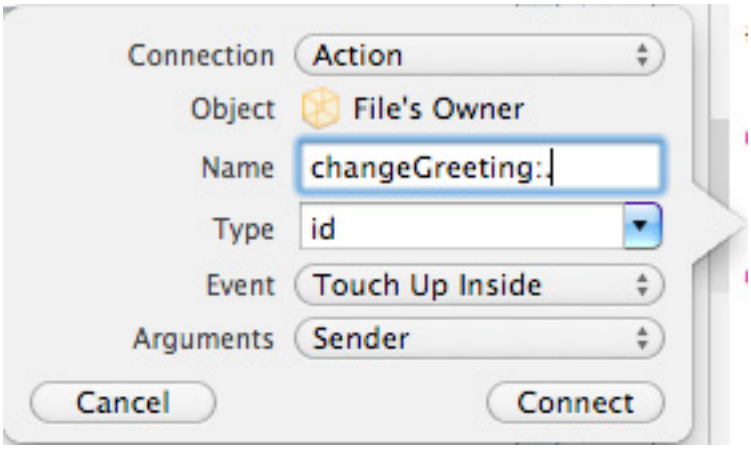
18) While holding down the 'Control' key or 'Ctrl' Key down, click on the Hello button with your mouse and drag the arrow to where it is located on the screen.



19) When you release the mouse button a window will open as follows



20) Change the settings to the following then click Connect



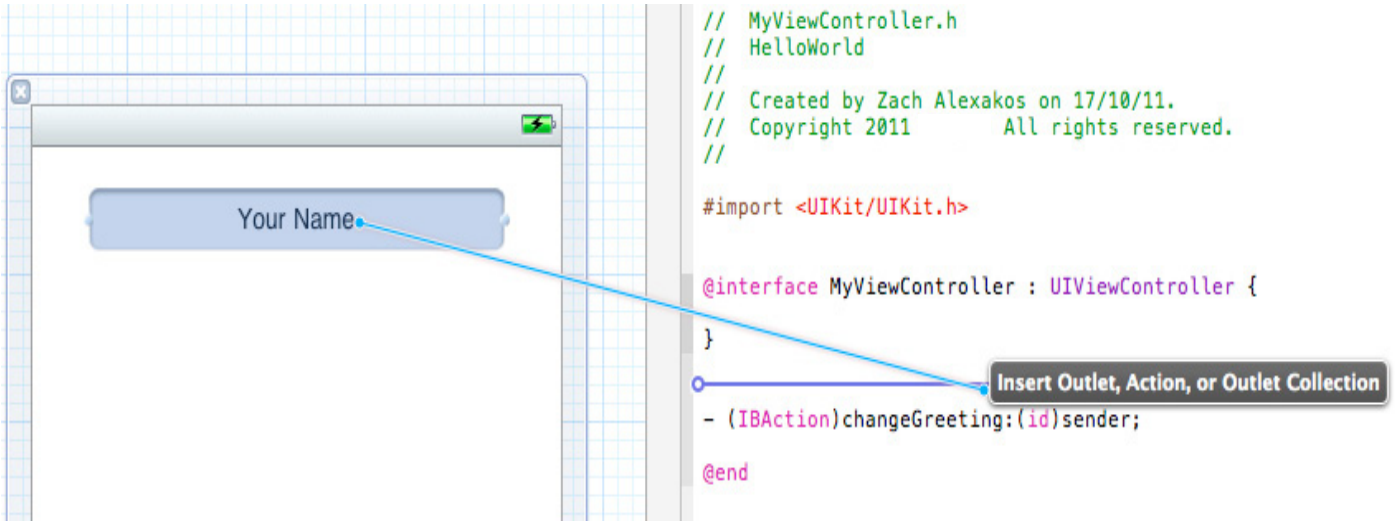
21) You will notice that in the Assistant Editor a line of code has been added to our MyViewController.h file as shown below

```
//
//  MyViewController.h
//  HelloWorld
//
//  Created by Zach Alexakos on 17/10/11.
//  Copyright 2011        All rights reserved.
//

#import <UIKit/UIKit.h>


@interface MyViewController : UIViewController {

}
- (IBAction)changeGreeting:(id)sender;

@end
```

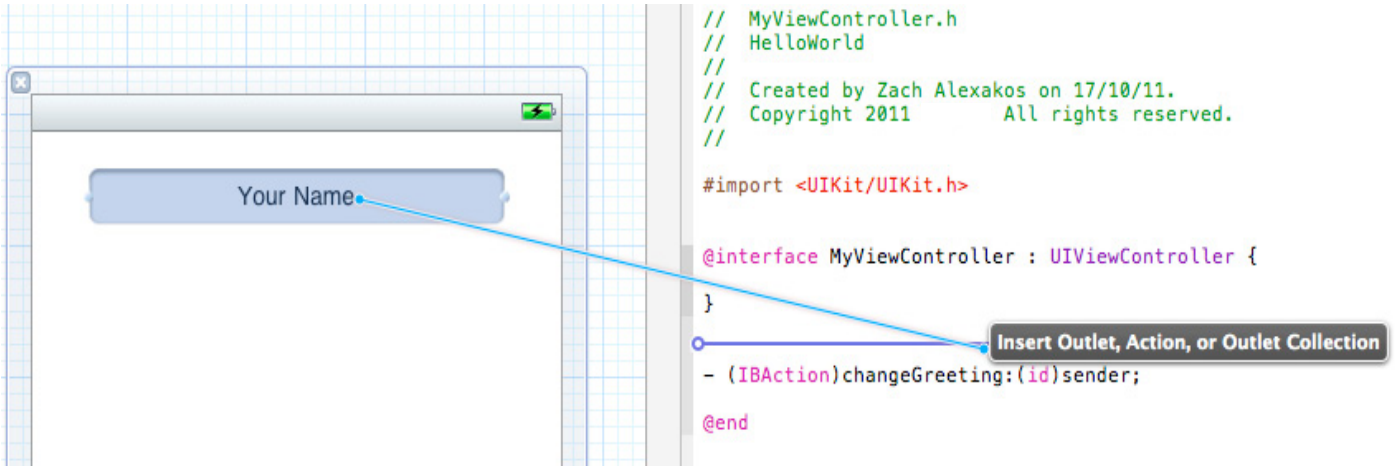
22) A short explanation of the code - (IBAction)changeGreeting:(id)sender;
IBAction is a special keyword that is used to tell thew Interafce Builder to treat a method as an action for target/action connections. It is defined as void.

The implementation file has also had a line of code added. Click on the MyViewController.h file to see addition made.

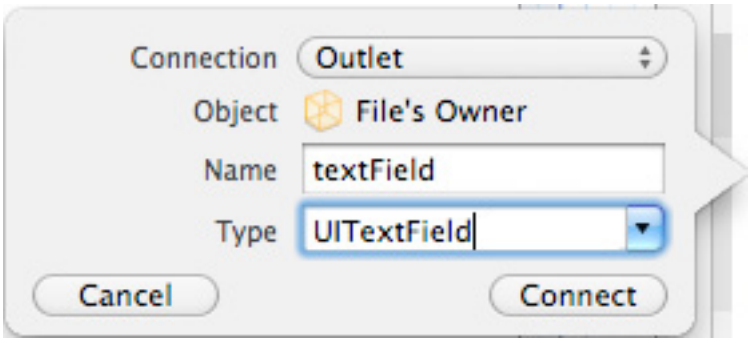
23) We will now create an outlet in the Text Field by performing a similar task in Step 18. Click on the Text Field and once it is highlighted hold the 'Control' or 'Ctrl' key down and drag the line from the Text Field to the Assistant Editor as shown below



24) Drag while holding down the 'Control' or 'Ctrl' key across to the Assistant Editor as shown below. Release the mouse and proceed to step 25.



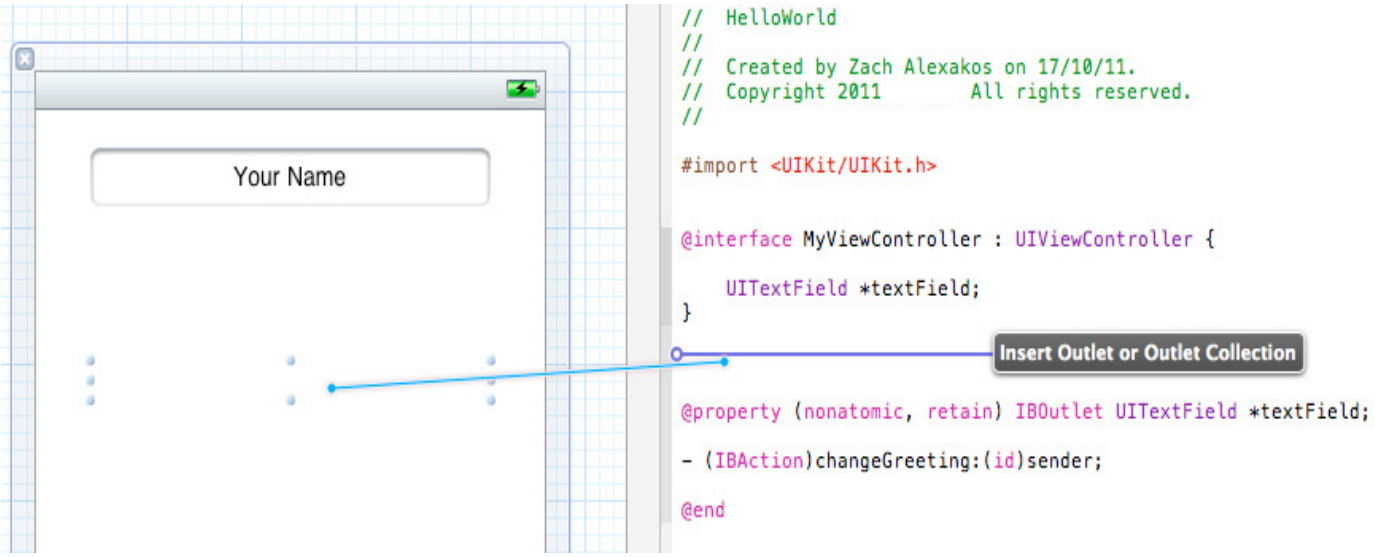
25) As occured with the 'Hello' button once you release the mouse button a window will appear to allow you to configure the Text Field settings. Change the settings as indicated below.



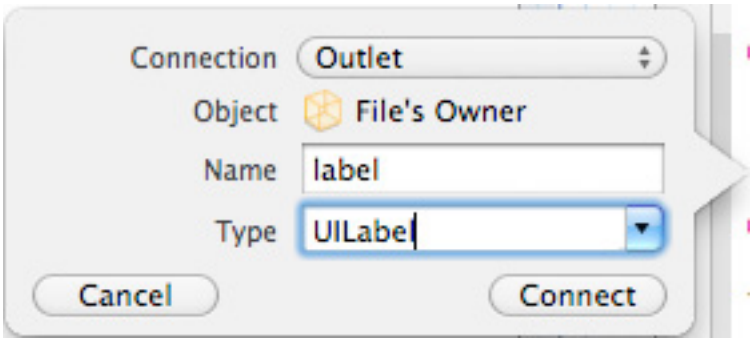
26) A short explanation of the code @property (nonatomic, retain) IBOutlet UITextField *textField;

IBOutlet is a special keyword that is used to tell the Interafce Builder to treat a property as an outlet. It is defined as nothing so it has no effect at compile time.

27) Finally the same steps need to be followed to configure the 'Label'. Follow the screenshots below.



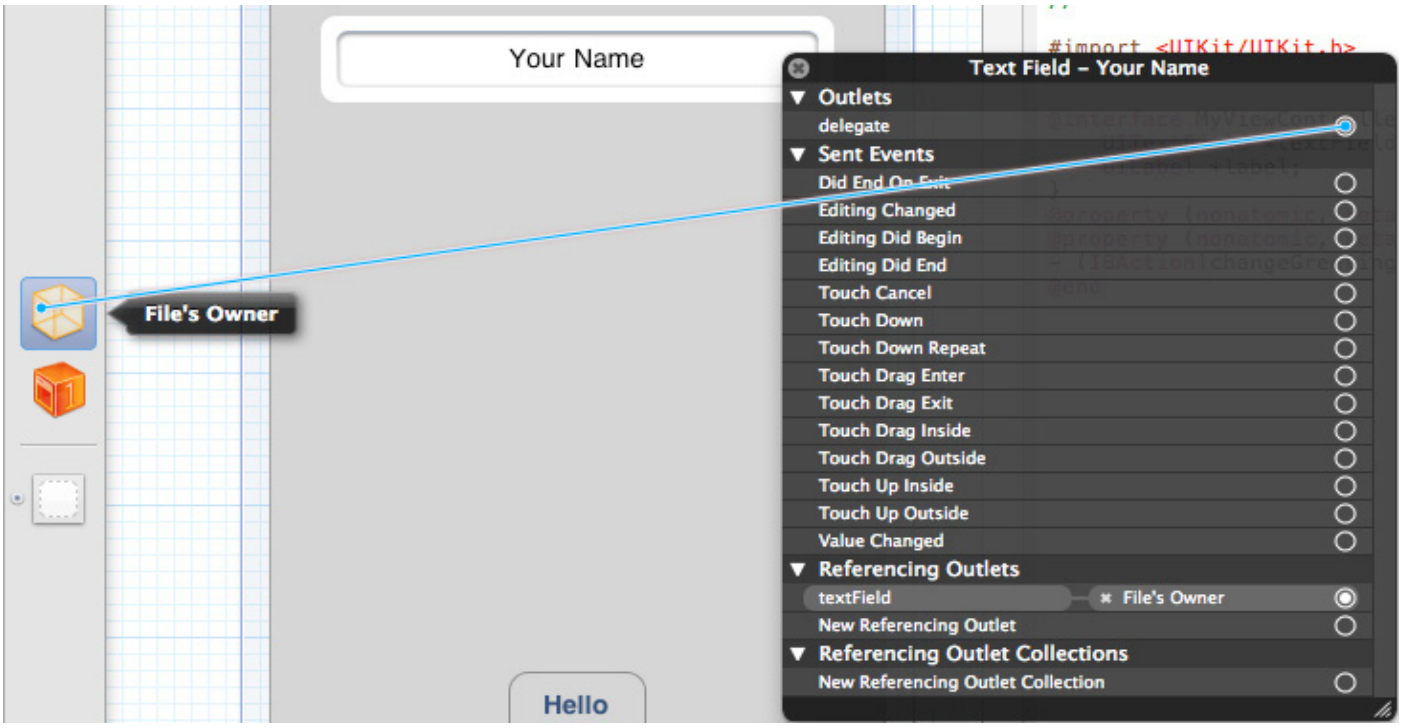
28) Fill in the details and click Connect



29) Your .xib file should display the following code

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


@interface MyViewController : UIViewController {
    UITextField *textField;
    UILabel *label;
}
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UITextField *textField;
- (IBAction)changeGreeting:(id)sender;
@end
```

30) Press the 'Control' or 'Ctrl' key and hold it down and double click on the Text Field 'Your Name'. A translucent panel will appear. Click on the 'delegate' radio button and holding down the mouse button drag the line to the 'File's Owner' symbol as shown below



31) Now Save the file
32) Click Run
33) Make sure the iOS Simulator is selected
34) Type in your name and Press the 'Hello' button

TROUBLESHOOTING:

You may find that when you run the application you have created it outputs a blank screen. There could be a number of reasons for this. First, you may included the incorrect code syntax, alternatively you may not have linked objects correctly and so on. Any program will require some tweaking and correction. Below the source code for the "HelloWorld" application has been included so that you can check over the code, delete irrelevant sections and include parts that may have been missed.

There are four files from two classes that we will be most concerned with, assuming that the remainder of the project has been set up correctly.

These files are called:

A) From the HelloWorldAppDelegate Class:
1) The Header File: "HelloWorldAppDelegate.h"
2) The Implementation File: "HelloWorldAppDelegate.m"

B) From the MyViewController Class:
3) The Header File: "MyViewController.h"
4) The Implementation File: "MyViewController.m"

1) The Header File: “HelloWorldAppDelegate.h”

```
//
//  HelloWorldAppDelegate.h
//  HelloWorld
//
//  Created by Zach Alexakos on 10/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import <UIKit/UIKit.h>

@class MyViewController;

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate> {
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) MyViewController *myViewController;

@end
```

2) The Implementation File: “HelloWorldAppDelegate.m”

```
//
//  HelloWorldAppDelegate.m
//  HelloWorld
//
//  Created by Zach Alexakos on 10/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import "MyViewController.h"
#import "HelloWorldAppDelegate.h"

@implementation HelloWorldAppDelegate

@synthesize window=_window;
@synthesize myViewController=_myViewController;


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
    launchOptions
{
    MyViewController *aViewController = [[MyViewController alloc]
                                         initWithNibName:@"MyViewController" bundle:nil];
    [self setMyViewController:aViewController];
    [aViewController release];

    self.window.rootViewController = self.myViewController;

    // Override point for customization after application launch.
    [self.window makeKeyAndVisible];
    return YES;
}

- (void)dealloc
{
    [_myViewController release];
    [_window release];
    [super dealloc];
}

@end
```

3) The Header File: “MyViewController.h”

```
//
//  MyViewController.h
//  HelloWorld
//
//  Created by Zach Alexakos on 17/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface MyViewController : UIViewController <UITextFieldDelegate> {
}
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) IBOutlet UITextField *textField;
@property (nonatomic, copy) NSString *userName;

- (IBAction)changeGreeting:(id)sender;
@end
```

4) The Implementation File: “MyViewController.m”

```
//
//  MyViewController.m
//  HelloWorld
//
//  Created by Zach Alexakos on 17/10/11.
//  Copyright Zach Alexakos 2011. All rights reserved.
//

#import "MyViewController.h"

@implementation MyViewController;

@synthesize label=_label;
@synthesize textField=_textField;
@synthesize userName=_userName;


- (void)dealloc
{
    [_textField release];
    [_label release];
    [_userName release];
    [super dealloc];
}
```

continued next page-

4) The Implementation File: “MyViewController.m” cont’d...

```
- (BOOL)textFieldShouldReturn:(UITextField *)theTextField {
    if (theTextField == _textField) {
        [theTextField resignFirstResponder];
    }
    return YES;
}

- (IBAction)changeGreeting:(id)sender {
    self.userName = self.textField.text;

    NSString *nameString = self.userName;
    if ([nameString length] == 0) {
        nameString = @"World";
    }
         NSString *greeting = [[NSString alloc] initWithFormat:@"Hello, %@!", nameString];
    self.label.text = greeting;
    [greeting release];

}
@end
```