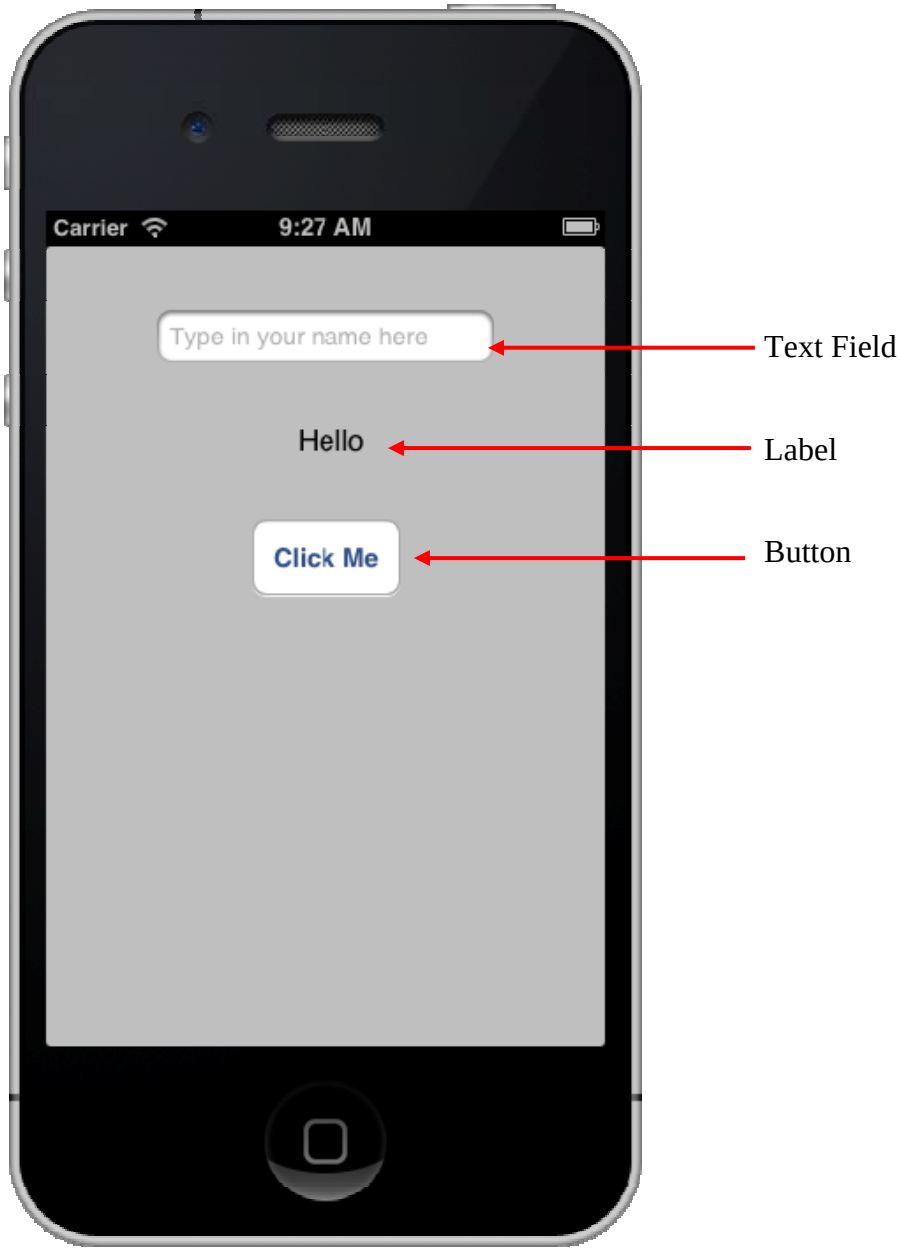
My First iPhone App
(for Xcode version 6.4)

1. Tutorial Overview

In this tutorial, you're going to create a very simple application on the iPhone or iPod Touch. It has a text field, a label, and a button as shown below. You can type your name into the text field and then click on the button, and the label's text will be updated to show a welcome message.



In order to do this tutorial and to develop apps for the iPhone, you need to have the following tools:

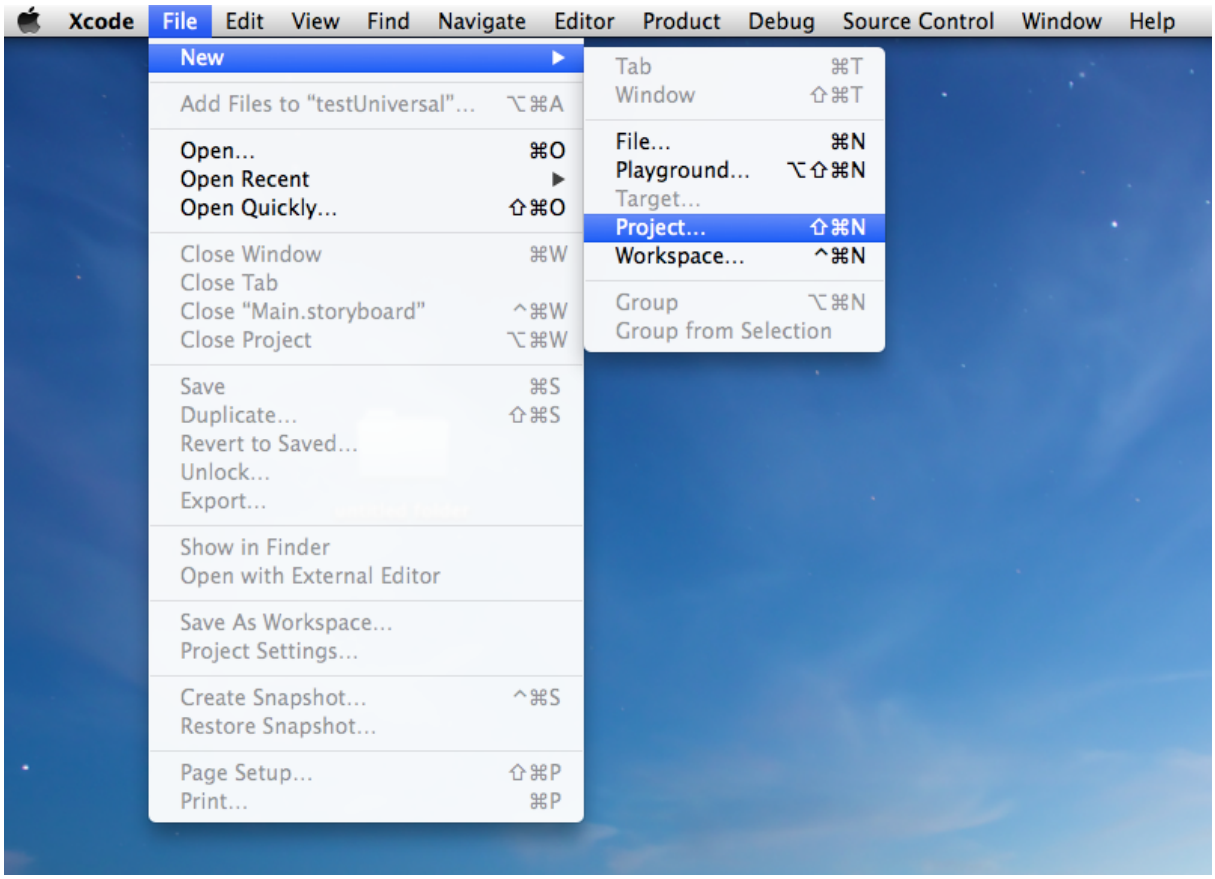
- An Intel-based Mac running OS X 10.9.5 or later.
- The Xcode development software. You can download the software from the App store for free.

2. Creating Your Project

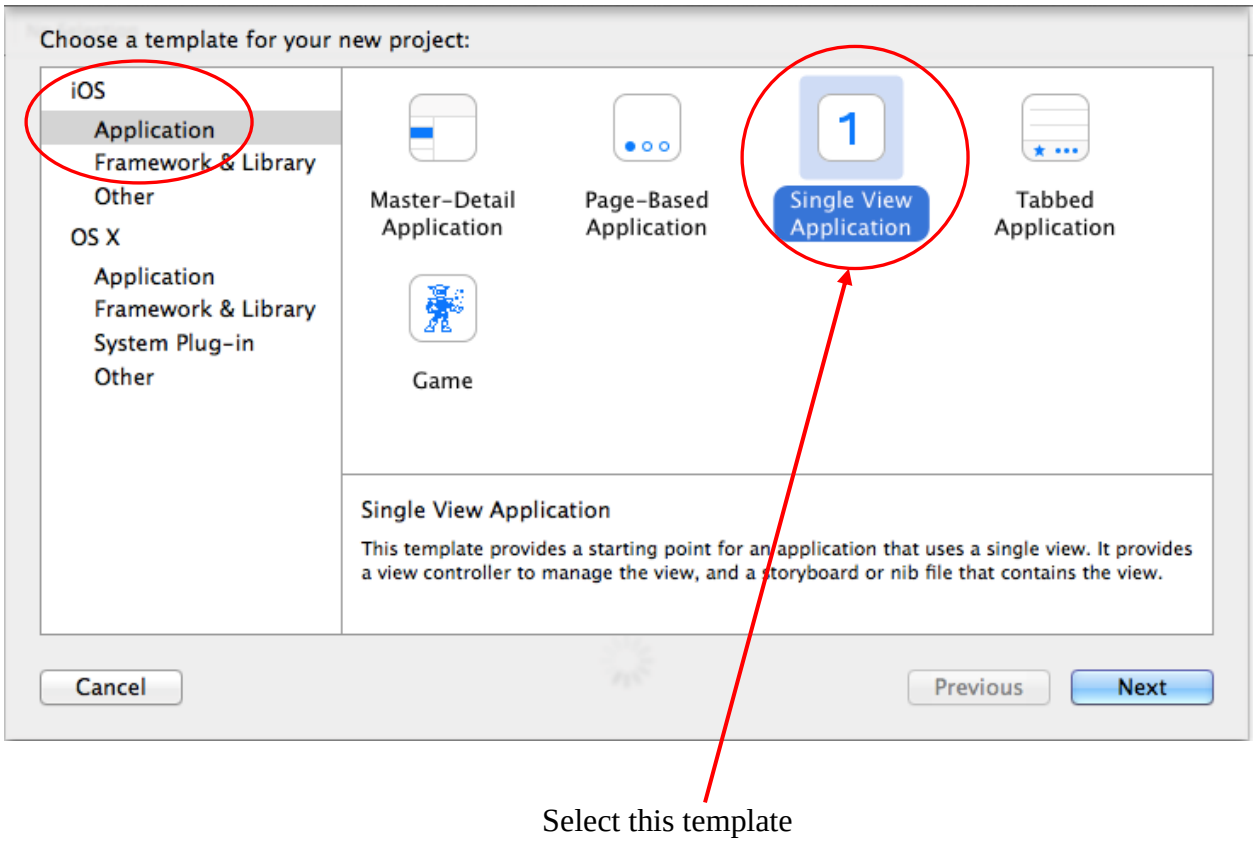
The main tool you use to create applications for the iPhone is Xcode.
Launch Xcode by clicking on the blue icon with a hammer on the dock or in the Applications folder.

If the welcome window comes up, you can just close it.

Create a new project by choosing from the Xcode menu File > New > Project. The notation used here means that you first click on the File menu and then you click on New in the drop-down menu, and finally click on Project.



You should see a window for choosing a template for your new project like this.



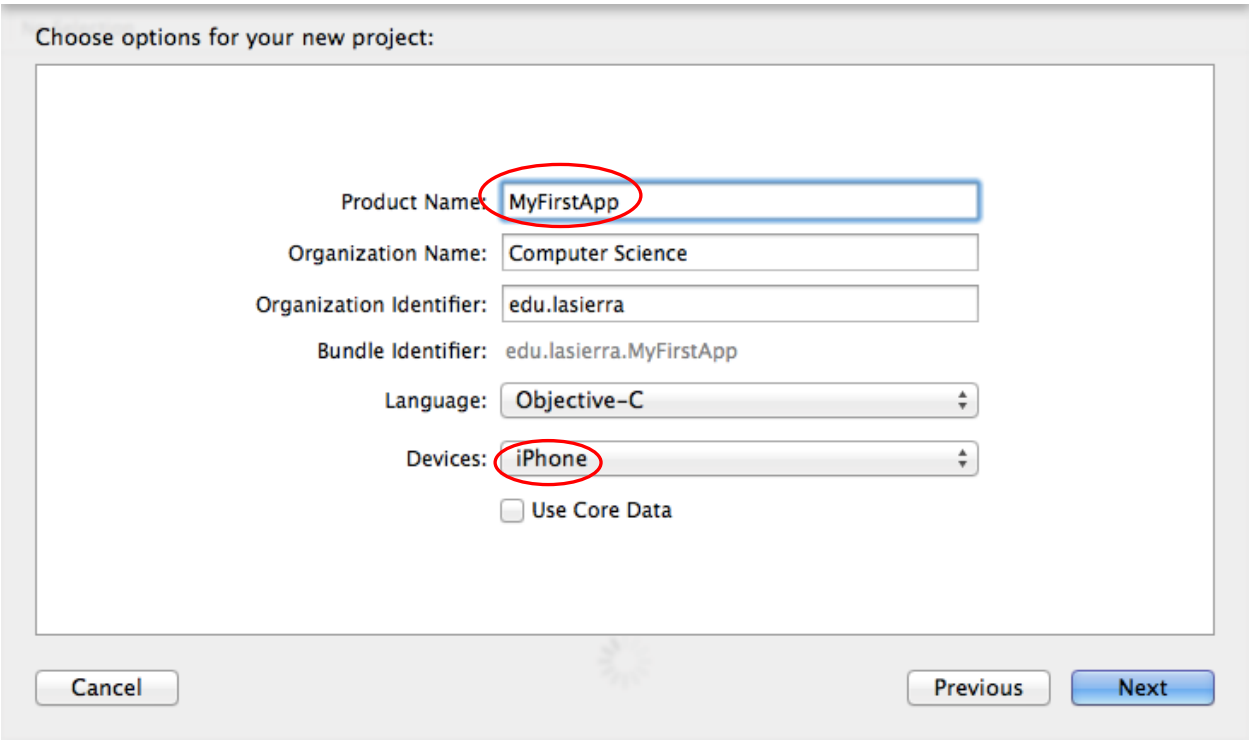
Select this template

In the left side of the window, make sure that the iOS Application category is selected. If not, then just click on it to select it. Select the Single View Application template icon, and then click the Next button.

A New Project window appears to allow you to specify the name of your project as shown next.

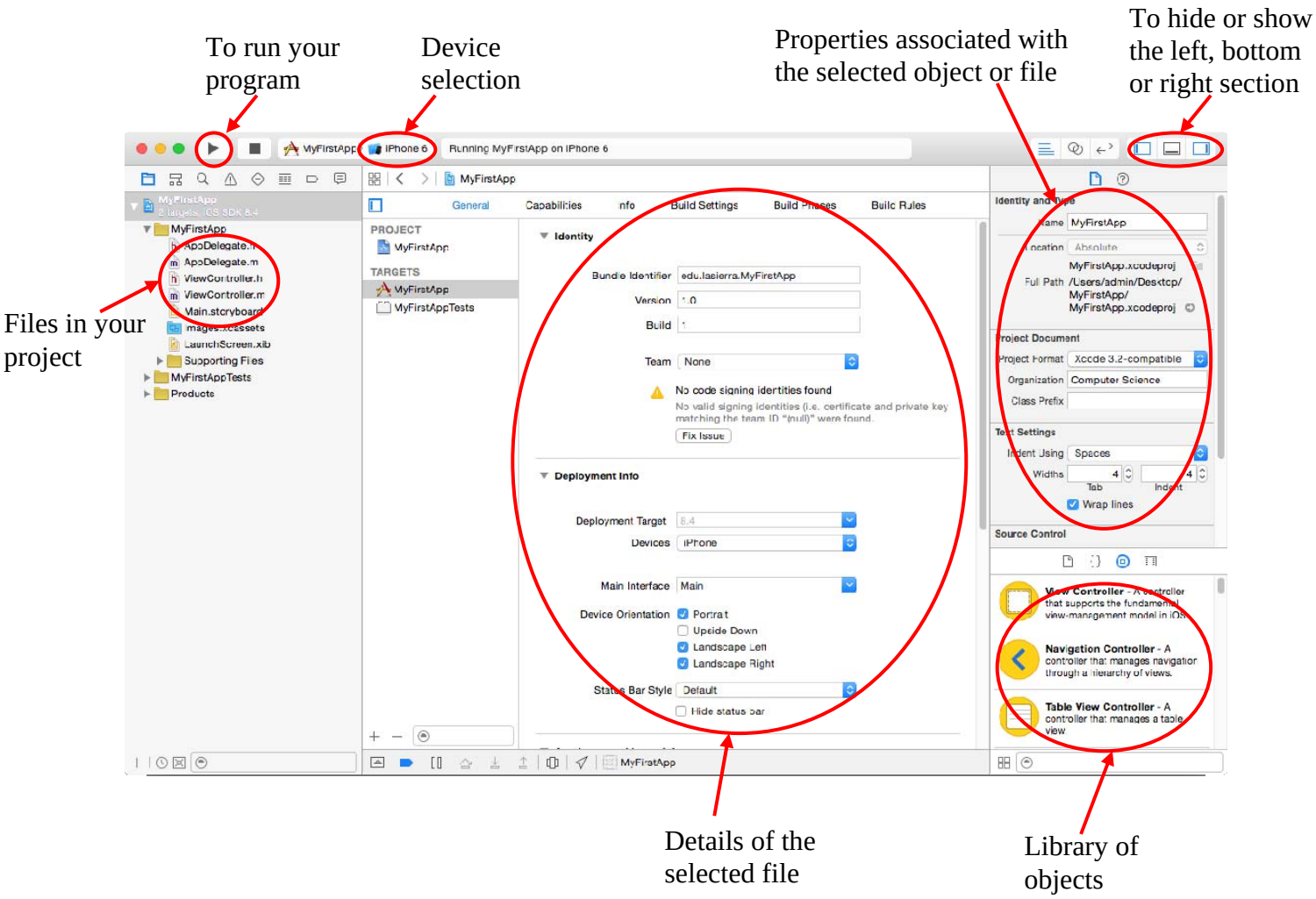
- Type in MyFirstApp for the Product Name.
- Type in Computer Science for the Organization Name.
- Type in edu.lasierra for the Organization identifier.
- Select Objective-C for the Language.
- Select iPhone for the Devices.

Click Next to continue.



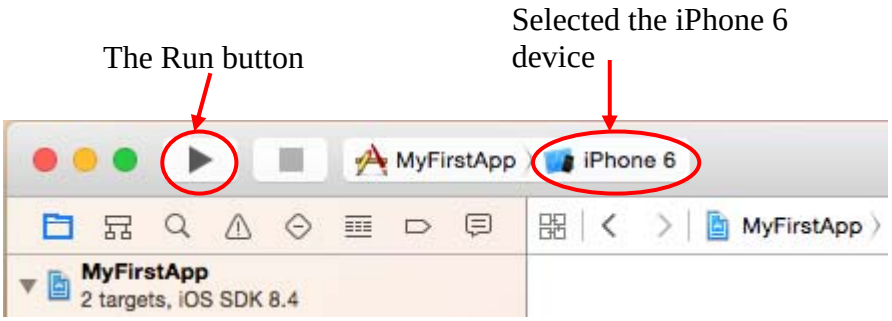
In the next screen, select the location of where you want to store your project. You can just store it on the Desktop. Click Create to create your project and continue.

You should now see the main project window with the name MyFirstApp containing all of the necessary files for your project like this.



This window consists of four sections or panes. The left pane (called the Navigator) shows the files in your project. To modify a particular file in your project simply select it here. The middle pane shows the details of the selected file, or details about your project. In the above picture, the project MyFirstApp is selected so the middle section is showing the details about your project. On the right side (called the Utilities section), the top section shows the properties or attributes of the selected file or object, and the bottom section shows a library of objects that you can use. Objects such as buttons, labels, text boxes, and so on, that you have available to add onto your iPhone screen are found in this library

Since Xcode has included everything that you need to run the app inside your newly created project, you can now run the application by clicking on the triangle run button in the top right corner. Before you click on the run button, make sure that you have selected one of the iPhone devices (and not an iPad device).



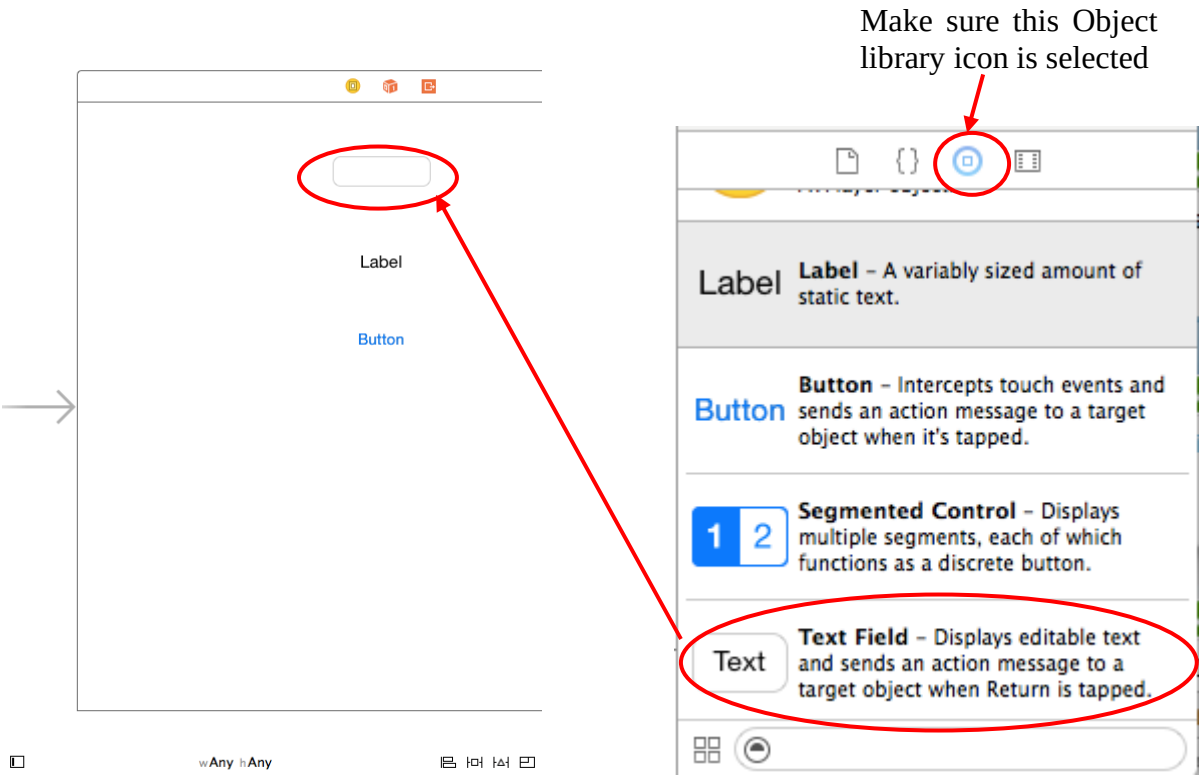
The iOS Simulator should launch automatically, and when your app starts up you should see a blank screen. The iOS Simulator simulates (almost) exactly everything that you will see if you were to run the app on your iPhone or iPad device. But for development purposes, it is much easier to just run it on the Simulator.

Now, back to our project, why is there just this blank screen? Well, because you have not added anything into your app! The template that Xcode has created for you is just an empty app. You will have to put whatever you want into it. Let's quit the iOS Simulator by choosing from the menu iOS Simulator > Quit iOS Simulator to continue.

3. Designing Your Screen Layout

The first step in creating your app is to design your screen layout by placing objects onto it. For our first app, we will be placing the three objects, Text Field, Label and Button on the screen. Back in the main Xcode window, select the file Main.storyboard in the left section. This file defines the objects that you want to have on your iPhone screen and how they are positioned. After selecting this file, the middle section of the Xcode window will show a picture of your iPhone screen. If the window is not large enough to show the entire iPhone screen, you will have to scroll around to see the entire iPhone screen.

From the Object library on the bottom right side of the Xcode window, find the Text Field object and drag it onto the layout screen in the middle. If you don't see the Object library, make sure that the Object library icon is selected. Position it towards the top of the screen and horizontally centered. A blue vertical line will appear and the object that you are dragging will snap to the line when it is centered.



- Drag a Label object and place it below the Text Field as shown in the picture above. Make it snap to the vertical centered blue line.

- Drag a Button object and place it below the Label as shown in the picture above. Make it snap to the vertical centered blue line.

Now, let's save your changes and then run your app to see what happens. Do you remember how to do that? In the Xcode project window, click on the Run button. Again, the iOS Simulator comes up, but this time, instead of the blank screen, you also see the three objects that you have added. However, the three objects are not centered horizontally. We will fix this later.

But for now, play around with the three objects. Click on the objects and see what happens. When you click on the button, it flashes blue telling you that you have clicked it, but nothing more. When you click in the text box, the soft keyboard pops up and you can type in your name or whatever inside it.

If the soft keyboard does not appear, then select from the iOS Simulator menu Hardware > Keyboard > Toggle Software Keyboard.

However, once the soft keyboard is up, there is no way that you can get rid of it – at least not for now.

After you are done playing, quit the iOS Simulator by choosing from the menu iOS Simulator > Quit iOS Simulator.

3.1. Adding Layout Constraints

Why are the three objects not centered horizontally even though you have horizontally centered them on the layout screen? The reason is because this is only a generic layout screen, and this layout screen is used for different devices with different actual screen sizes. For example, the iPhone 6 screen size is different from the iPad Air screen size.

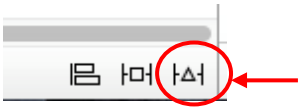
What we need to do is to add layout constraints to each of the objects so that they will be placed correctly regardless of the actual device screen size. Perform the following to add the layout constraints:

- Control drag (i.e. hold down the control key and drag with your mouse) the Text Field up a little. You will see a short blue line following your mouse pointer. Let go the mouse button and a pop-up menu will appear. Select Center Horizontally In Container

- Repeat the above action for the Label and the Button.

- Several orange lines will appear. Orange lines mean that you have added at least one layout constraint for the object but there is not enough information for the system to determine the exact position of the object.

- Click on the Resolve Auto Layout Issues icon at the bottom right corner of the layout screen.



- In the pop-up menu, select Add Missing Constraints. If you select each of the three objects on your layout screen now, you will not see the orange lines, and instead there will be blue lines telling you that all of the necessary layout constraints have been added.

Now run your app to see the result. Notice that the three objects are now centered.

- From the iOS Simulator menu, click on Hardware > Rotate Left. Notice that the three objects are still centered horizontally even in landscape mode.

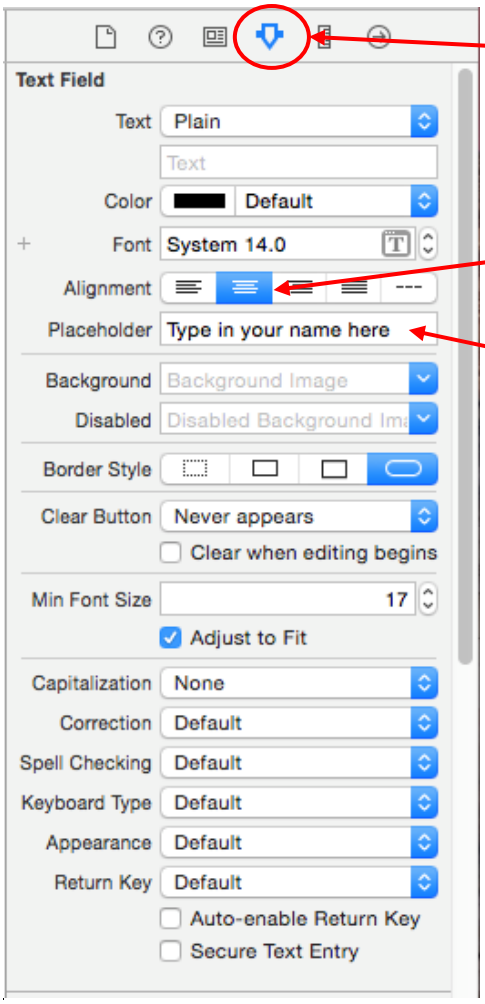
Exit the simulator. Back in Xcode, select the iPad Air device. Run the app again, and see that the three objects are also centered horizontally in this bigger screen.

3.2. Changing Object Properties

All of the objects have many properties associated with it such as the size and color. Different objects will have different properties. For example, the Text Field will have an Alignment and Font size properties.

To select an object on the layout screen simply click on the object. To deselect an object just click anywhere outside of the object.

We will now change some properties associated with the Text Field object. Select the Text Field object and review the properties associated with it in the Attributes window in the top-right section of the Xcode window as shown next.



Make sure this Attributes inspector icon is selected

Alignment property set to Center

Placeholder property with the words "Type in your name here"

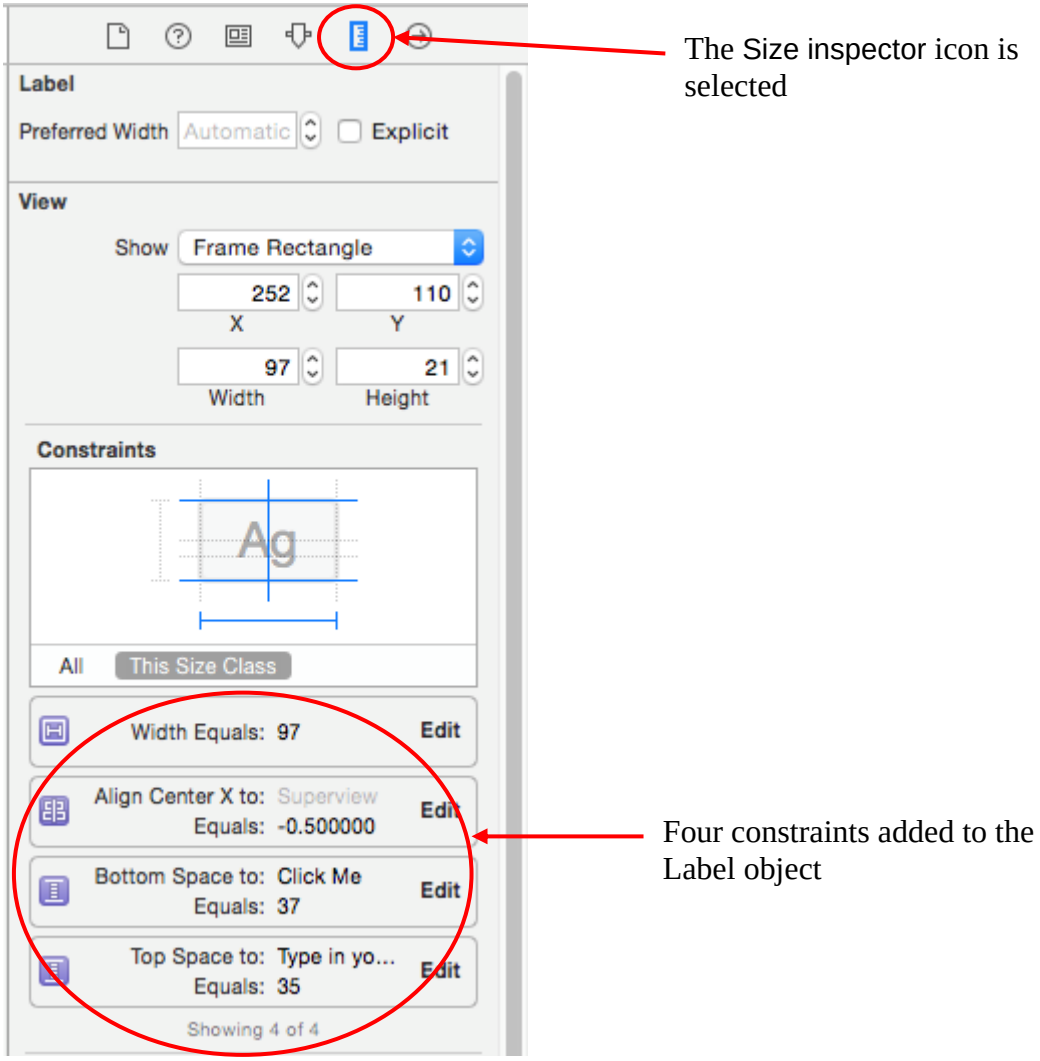
Here, you can change the Text Field properties such as the font color, font size, alignment, etc.

- For this tutorial, type the words "Type in your name here" in the Placeholder property box.
- Change the Alignment property to Center.

Noticed in the Text Field box are now the words “Type in your name here,” but the box is not wide enough to display the entire phrase. You can make the Text Field box wider by dragging the left or right edge of the box. Make the box wide enough so that the entire phrase is shown.

Noticed that some of the blue layout constraint lines are changed back to orange as soon as you change the width of the box. Click on the Resolve Auto Layout Issues icon again. In the pop-up menu, select Update Constraints. You may also need to select Reset to Suggested Constraints.

You can see the layout constraints added to an object by first selecting the object and then click on the Size inspector icon to display the constraints in the window as shown next.



- Change the Label text property to say “Hello,” and
- Change the Button text property to say “Click Me.”

Run your app to make sure that everything is still working correctly up to this point.

4. Writing the Code

What we want to do is to type in a name in the text box and then when you click on the button it will display a personalize hello message in the label.

In order to get the objects to do something when you click on them, you will have to write some instructions to tell the computer what to do and when to do them. All of the instructions put together are called a *computer program*, and the process of writing these instructions is called *computer programming*. We will now learn to write a computer program using the Objective-C language.

Back in Xcode, there are two files, ViewController.h and ViewController.m, that we will be modifying. You will find both of them listed in the Project Navigator window on the left side of the main Xcode window. For now, don't worry too much about the details of the code. You will have the entire quarter to learn and figure that out!

The .h file usually contains the declaration of names for the objects and names of actions to be performed by the objects. The actions are known as **methods**. The .m file will contain the actual code or instructions for the methods.

Single click on your ViewController.h file and the contents of the file will show up in the middle section of your project window. Modify the file so that it matches the following listing. Note that some of the lines already exist in the template so do not re-type them. I suggest that you don't just copy and paste the code in but actually type it in yourself.

Be very careful that you type everything exactly like in the listing. Things to watch out for are:

- Upper case and lower case letters are different. So if the word is UITextField, then don't type in UITextfield because you will get an error when you try to build and run it.

- Most lines will end with the semicolon ; and some with the braces { or }.

- At least one space is required to separate between words, however, no spaces are required to separate between a symbol such as :, *, @ and), and a word.

- Empty lines and line indentations only serve to enhance the readability of the programmer. They don't matter much to the computer, but helps you and other programmers looking at your code. So get into the habit of indenting your code as in the sample code.

- Text after the two slashes like this // are comments, but text before the two slashes are not. Comments are shown in green and are ignored by the computer.

- Notice that when you type the first few characters of a keyword, the rest of the word will appear. If the word that appears is what you want, then just press the Tab key to accept it. This is the autofill feature. If it is not the word that you want then continue to type in what you want.


```
//
//  ViewController.h
//  MyFirstApp
//
//  Created by admin on 8/18/15.
//  Copyright (c) 2015 Computer Science. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController {
   UITextField *myTextField;    // name for the Text Field
   UILabel *myLabel;            // name for the Label
}

@property (nonatomic, retain) IBOutlet UITextField *myTextField;
@property (nonatomic, retain) IBOutlet UILabel *myLabel;

// method for changing the greeting in the label
- (IBAction)changeGreeting:(id)sender;

// method to remove the keyboard
- (IBAction)removeKeyboard:(id)sender;


@end
```

The two lines

```
   UITextField *myTextField;    // name for the Text Field
   UILabel *myLabel;            // name for the Label
```

are for declaring the names myTextField and myLabel that we will give to the Text Field and Label objects respectively.

The two lines

```
- (IBAction)changeGreeting:(id)sender;              // method for changing the greeting

- (IBAction)removeKeyboard:(id)sender;              // method to remove the keyboard
```

are for declaring the two methods changeGreeting and removeKeyboard that we will have. The method changeGreeting will be executed when the button is pressed to change the welcome message, and the removeKeyboard method will be executed when you press Done on the keyboard. This method is used to get rid of the keyboard.

Save your code and then Run the app to make sure that everything still works. Notice that there are several yellow warnings. For now, you can ignore them. The iPhone Simulator should come

up if you did not make any mistakes. If the simulator doesn't come up then just go back to the code and make sure that you have typed in everything correctly.

Next we need to modify the ViewController.m file. Single click on your ViewController.m file and the contents of the file will show up in the middle section of your project window. In this file, we will write the code for the two methods changeGreeting and removeKeyboard. Modify the file so that it matches the following listing.

Again be very careful that you type everything exactly like in the listing. Things to watch out for are:

- Get into the habit of using the Tab key to accept the correct words as much as possible. This will speed up your typing and also minimize errors.

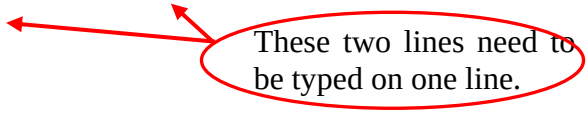
- Lines of code that are bracketed by /* and */, and shown in green, are comments.

- When you type an open brace { and then press the Enter key, the editor will automatically insert a closing brace on the next line. So don't manually type in another closing brace.

- When you move your cursor over a closing brace }, the matching open brace { will flash yellow. This will help you to match up the open and close braces.

- There is a line the changeGreeting method that reaches close to the right margin and doesn't end with a semicolon ; because it is too long to fit on one line on a printed page. Therefore, when printed they wrap around and continue onto the next line. However, when you type it in the editor, you need to type it on one line instead of two lines. Inside the editor, they will not be wrapped around to two lines.


```
//
//  ViewController.m
//  MyFirstApp
//
//  Created by admin on 8/18/15.
//  Copyright (c) 2015 Computer Science. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

@synthesize myTextField;
@synthesize myLabel;

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

// details of the changeGreeting method
- (IBAction)changeGreeting:(id)sender {
    NSString *nameString = myTextField.text;
    if ([nameString length] == 0) {
        nameString = @"Word";
    }
    NSString *greeting = [[NSString alloc] initWithFormat:@"Welcome %@ to La Sierra
University!", nameString];
    myLabel.text = greeting;
}

// details of the removeKeyboard method
- (IBAction)removeKeyboard:(id)sender {
    NSString *greeting = [[NSString alloc] initWithFormat:@"You clicked DONE"];
    myLabel.text = greeting;
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```



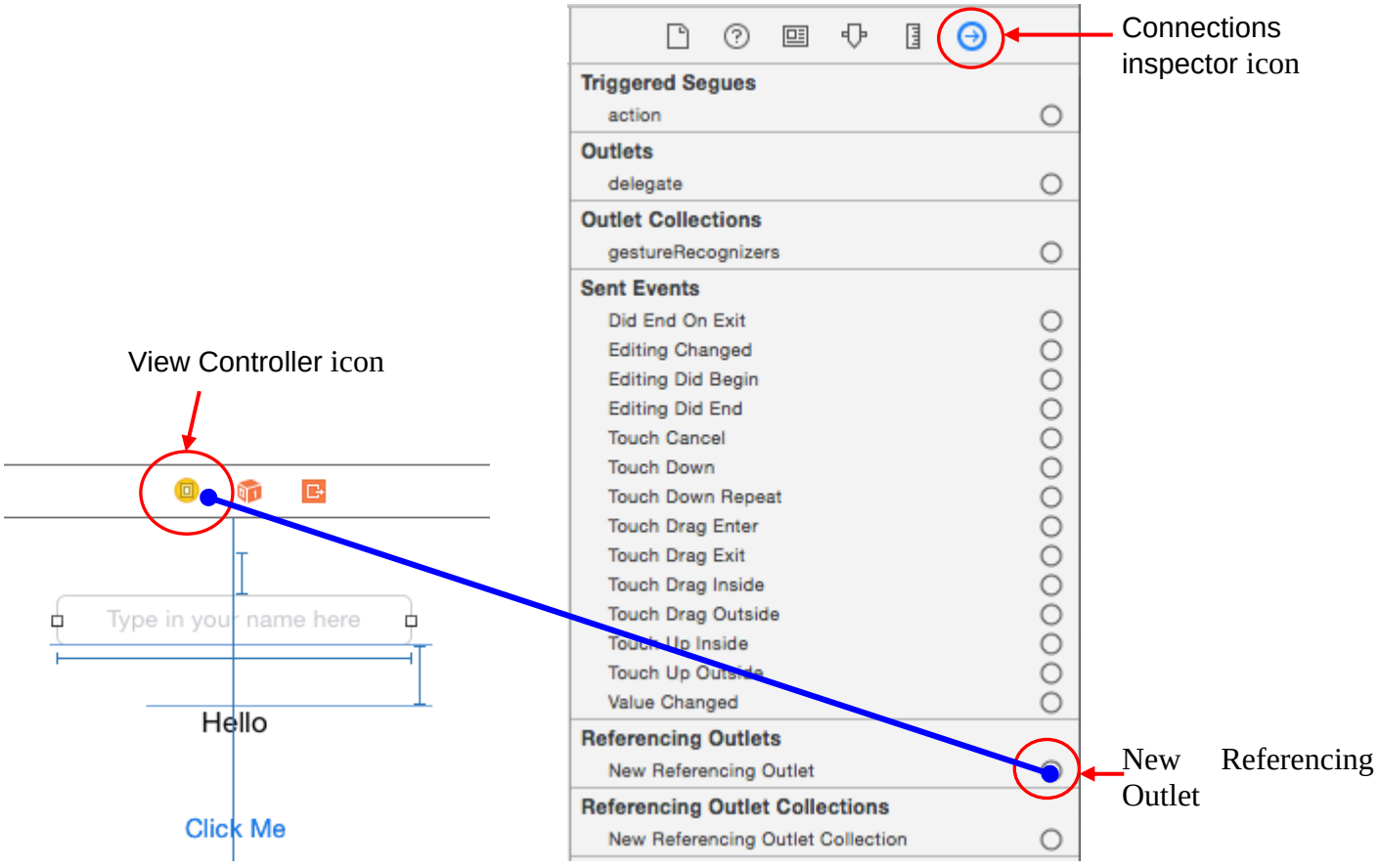
Just to make sure that you have not made any mistakes in your typing, once again, run your app to see if the simulator will come up. Again, if the simulator doesn't come up then go back to the code and make sure that you have typed in everything correctly.

At this point, even with the instruction code typed in, the app still doesn't do anything more. Clicking the button still won't do anything and you still can't get rid of the keyboard. The reason is that the code is still not being executed by the computer because you have not told the computer when to execute the code.

5. Connecting the Objects to the Code

We are now ready to connect the objects in your iPhone user interface to the code that you have just typed in. By connecting the code to the objects, the computer will execute the associated code when an action is performed on the object. There are two types of connections that we need to make, Outlets and Events. Outlets are for objects to output messages. So if you want an object to output some text, you need to connect its outlet to the corresponding name that you have given to it in your code. Events are methods that you want an object to perform when that object is triggered. So if you want your button to perform a particular method when it is clicked, you need to connect that button clicked action to that method.

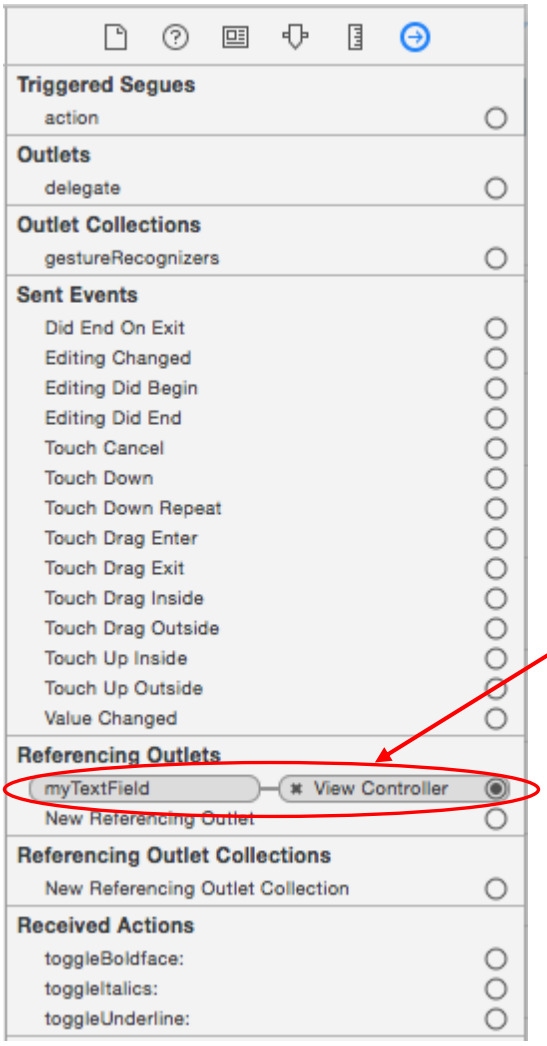
You will be making a total of four (4) connections from the Main.storyboard layout screen. For the first connection, select the Text Field object. Click on the Connections inspector icon (the arrow pointing to the right) on the right section of the Xcode window. This right section window will list all of the events and outlets available for the selected object.



Towards the bottom of the list there is a line that says New Referencing Outlet. Move your mouse over the empty circle next to New Referencing Outlet to see it change to a plus symbol. Then drag from the circle with the plus symbol to the yellow View Controller icon above the layout screen. A blue line will follow your mouse pointer as you drag the mouse. When you

release your mouse on the View Controller icon, a selection menu will pop up. Select myTextField.

Once you have done that, the Connections inspector for the Text Field box will look like this.

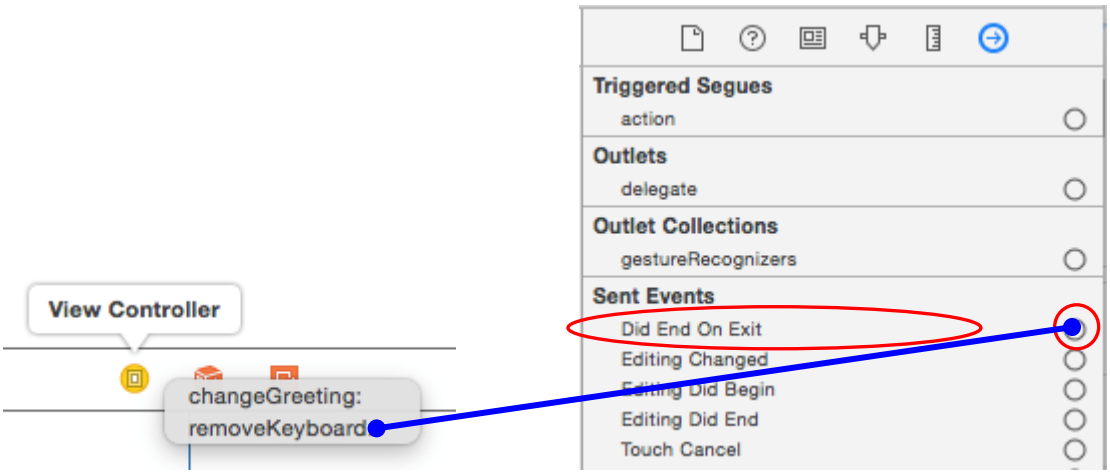


Click on × to delete the connection.

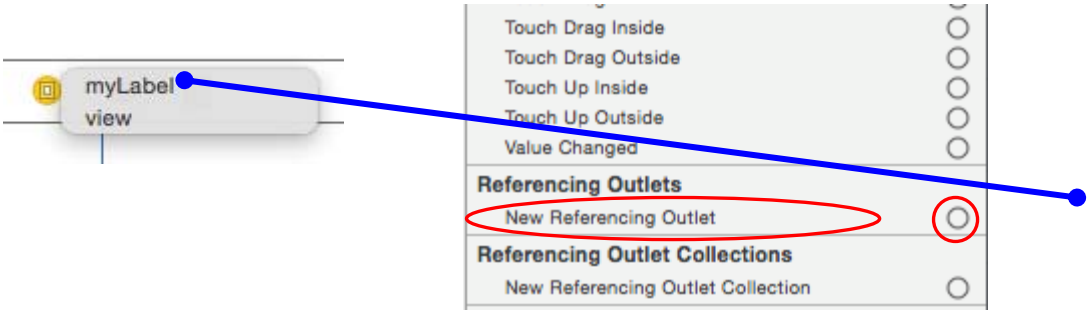
If you made a mistake, you can click on the × next to View Controller that is connected to myTextField to delete the connection. In fact, it will be a good exercise for you to click on the × now to delete the connection even if it is correct and then re-connect it back again.

While running this app, after entering your name in the Text Field box and you press the Return button on the keyboard, you want to execute the removeKeyboard method to remove the keyboard. When you press the Return button on the keyboard, the Did End On Exit event for that object is triggered.

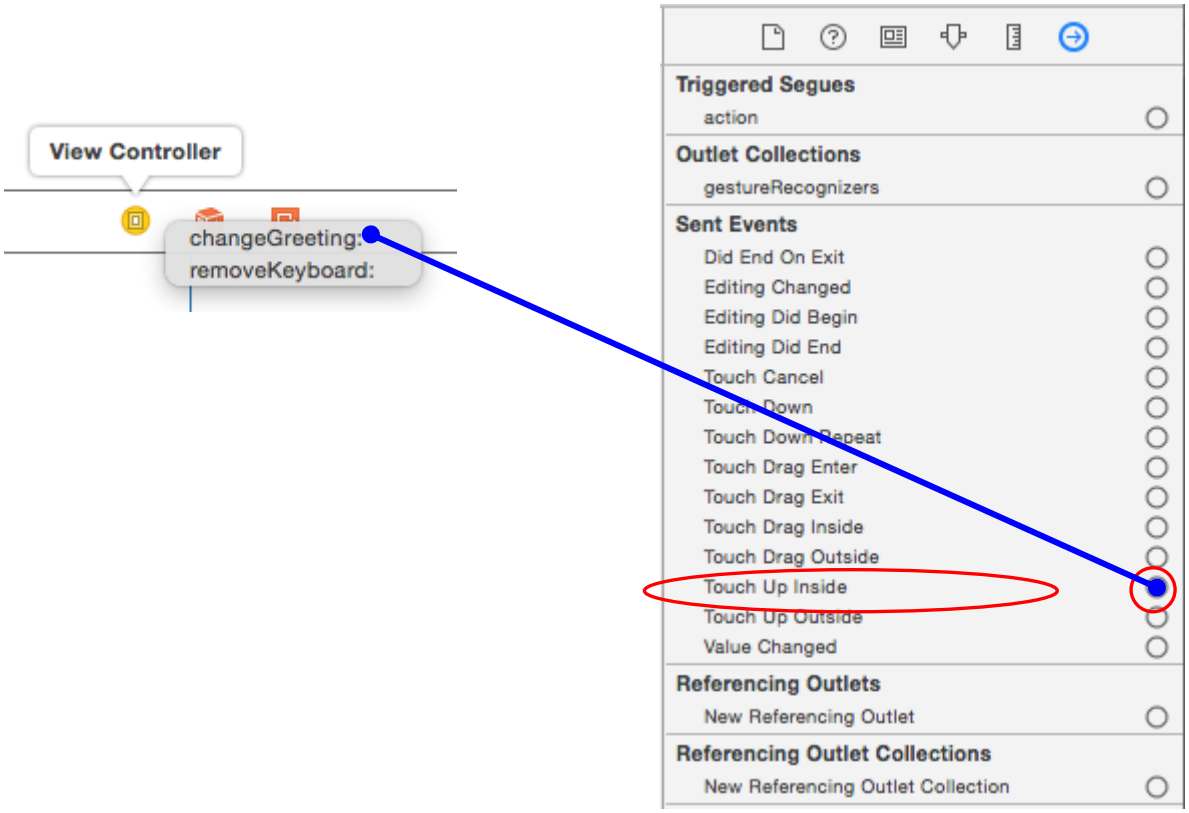
So select the Text Field object, and connect the Did End On Exit event to the removeKeyboard method. You should know how to do that by now. You drag from the empty circle next to Did End On Exit to the File's Owner icon and select removeKeyboard from the pop-up menu.



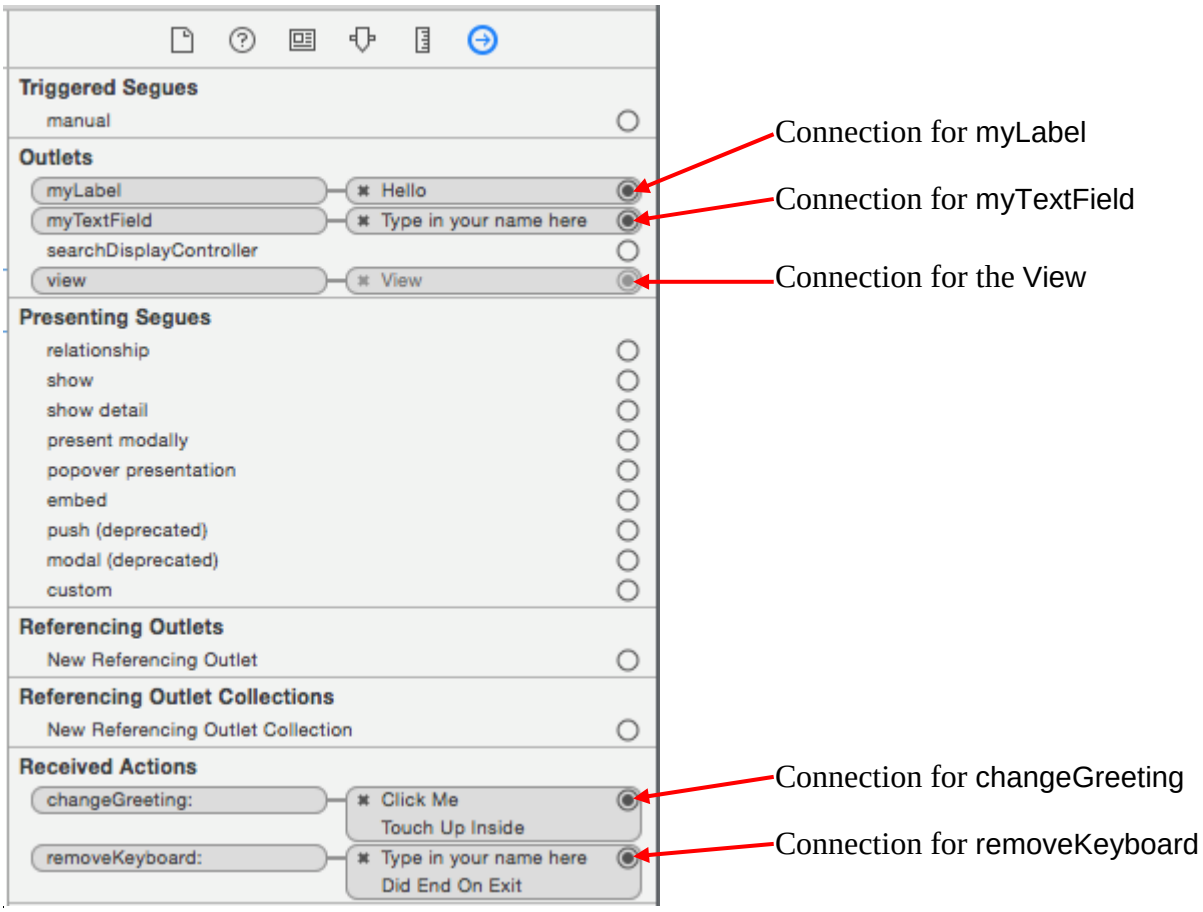
Make the connection for the Label object. Select the Label object in layout screen. Drag from the empty circle next to New Referencing Outlet to the View Controller icon. This time in the pop-up menu select myLabel.



For the button, we want to execute the changeGreeting method when it is clicked. When an object is clicked, the Touch Up Inside event is triggered, so we want to connect the Touch Up Inside event for the button to the ChangeGreeting method. First select the button so that you are seeing the button's Connections Inspector. Next, drag from the empty circle next to Touch Up Inside to the View Controller icon as shown next. From the pop-up menu, select the changeGreeting method.

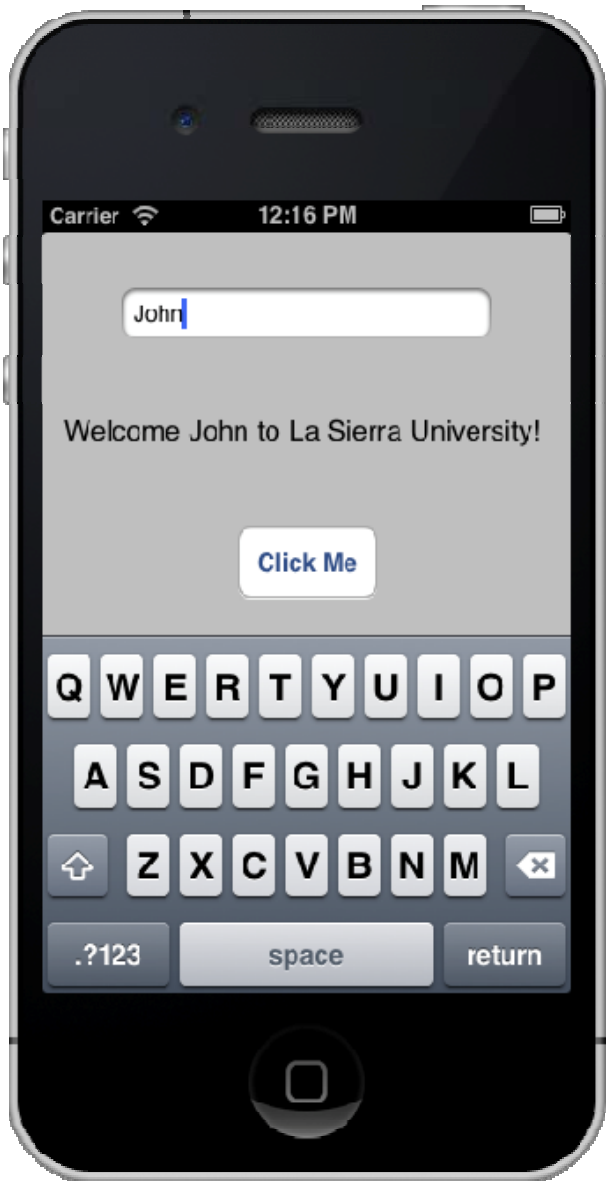


To double check that you have made all of the connections correctly, select the View Controller icon. You should see the following **five (5)** connections in the Connections inspector. If you don't, then you need to reconnect the objects correctly.



Notice that there is one extra connection that you did not make, and that is the View connection. This connection was made automatically for you when you selected the Single-View Application template for your project.

We are all done. Save the changes that you have made and click on Run to try out your very first app. Type in your name in the Text Field and then click on the Click Me button. You should see something like the following.



The label probably is not wide enough to display the entire welcome message. If this is the case, then go back into Xcode and make the label wider.

Again, if the iPhone simulator does not come up, then you have made a mistake somewhere, and at this point, most likely it is in the connections.

6. Uploading your app to your actual iPhone device

If you want to try this app on you iPhone, all you need to do is connect your iPhone to the computer. From the Xcode project window, select iOS Device from the drop-down menu at the top left corner of the window, and then Run the app again[1]. If the download is successful, you should see an icon on your iPhone with the name MyFirstApp. You can now tap on it to run it.

[1] In order for this to work, you'll need to get a paid developer account and a device that is registered to this account.

7. Exercises

1. When you first start the app, the label shows the word “Hello.” Modify the app so that it will first show “My First App” instead. Hint: modify the Text attribute for the Label.

2. Modify the app so that the text displayed in the Label is red instead of black. Hint: modify the Color attribute for the Label.

3. When you type in a name that is long, the font size in the Label remains fixed. Modify the app so that the font size for the Label will automatically shrink smaller. Hint: there is a Label attribute called Autoshrink that affects this.

4. Modify the code so that when you click on the Click Me button, the label will always display the text “Hello World” regardless of what you have typed in the Text Field box. Hint: there is one line of code in the ViewController.m file that sets your name. You need to change this line accordingly.

5. Modify the code so that when you press the Done key on the keyboard, the label will display “Hello <your name>” where <your name> is your actual name instead of displaying “You clicked DONE”. Hint: you need to make the change in the removeKeyboard method.

6. In the ViewController.h file where you typed in the label name declaration line

   ```
   UILabel *myLabel;          // name for the Label
   ```

   Remove the semicolon at the end of the line, and then Run the app. What happens? First, the iPhone Simulator did not start. More importantly, look at the status at the top of the Xcode project window, it says that the Build has failed with 1 error. The error is flagged with a red exclamation mark. Notice that the error is not exactly on the line with the missing semi-colon. Sometimes, one little error can cause numerous errors!

   Put the semi-colon back. Experiment by deleting other characters and see what other error messages you get.

7. Remove all unnecessary code in the ViewController.m file. You should be able to remove the variable greeting from the lines in both the changeGreeting method and the removeKeyboard method and combine the two lines together into one line.

8. Change the label name myLabel with myFirstLabel. Note that there are several places that you will have to make the change, some in the ViewController.h file and some in the ViewController.m file. Furthermore, you will also have to redo the connections for the Label in the Connection inspector.

9. Add a second Label to the app that will display exactly the same thing as for the first Label, but in a different color.

10. Modify the code in the changeGreeting method so that when the user enters a name longer than 10 characters, it will display the message “Name too long”. If the name is 10 or less characters long, then it will display the original welcome message.

11. Reverse the two connections to the two methods changeGreeting and removeKeyboard so that when you click on the Click Me button it will execute the removeKeyboard method instead of the changeGreeting method, and when you press the Return key on the keyboard it will execute the changeGreeting method instead of the removeKeyboard.

