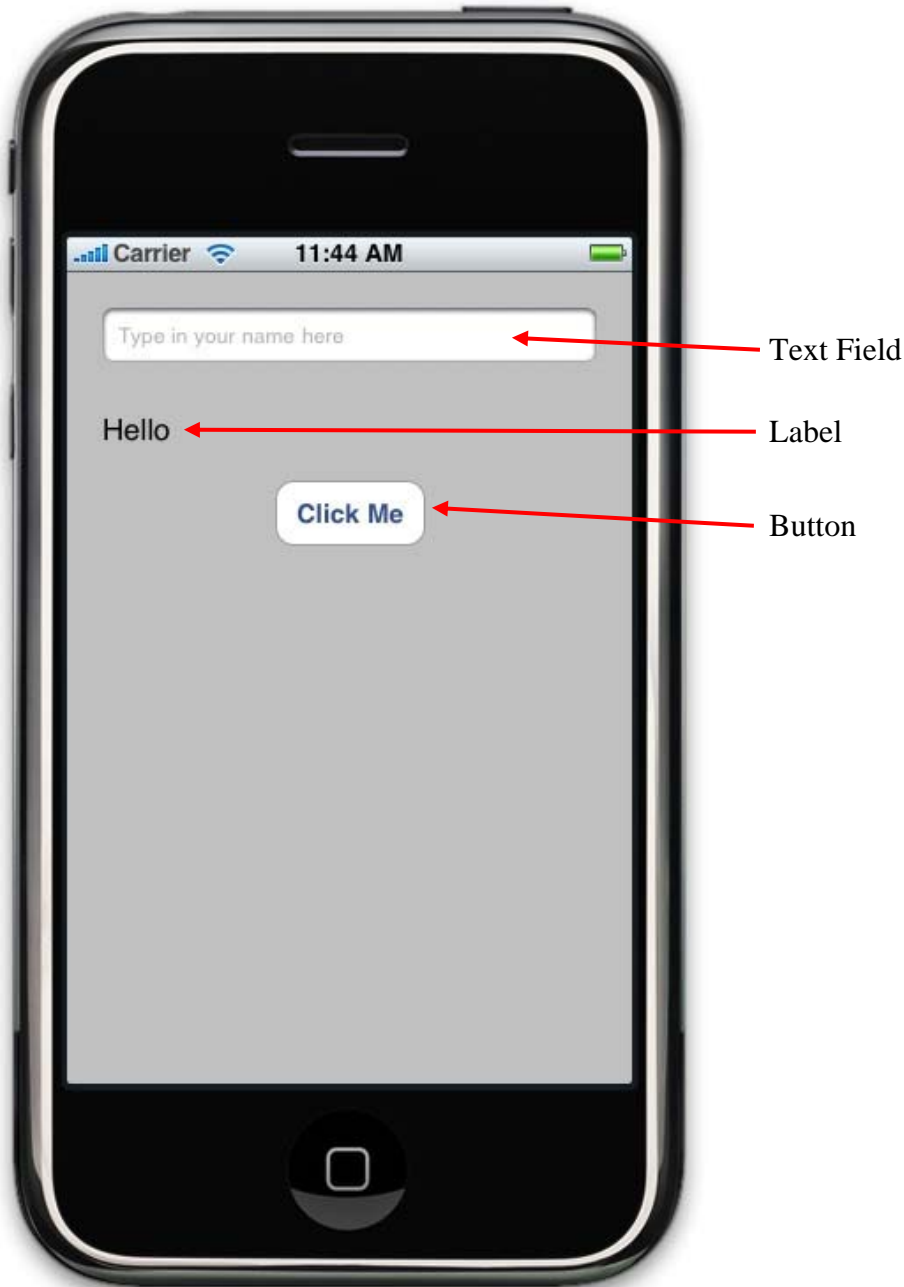


# My First iPhone App

## 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. You can type your name into the text field then click on the button and the label's text will be updated to show a welcome message as shown below.



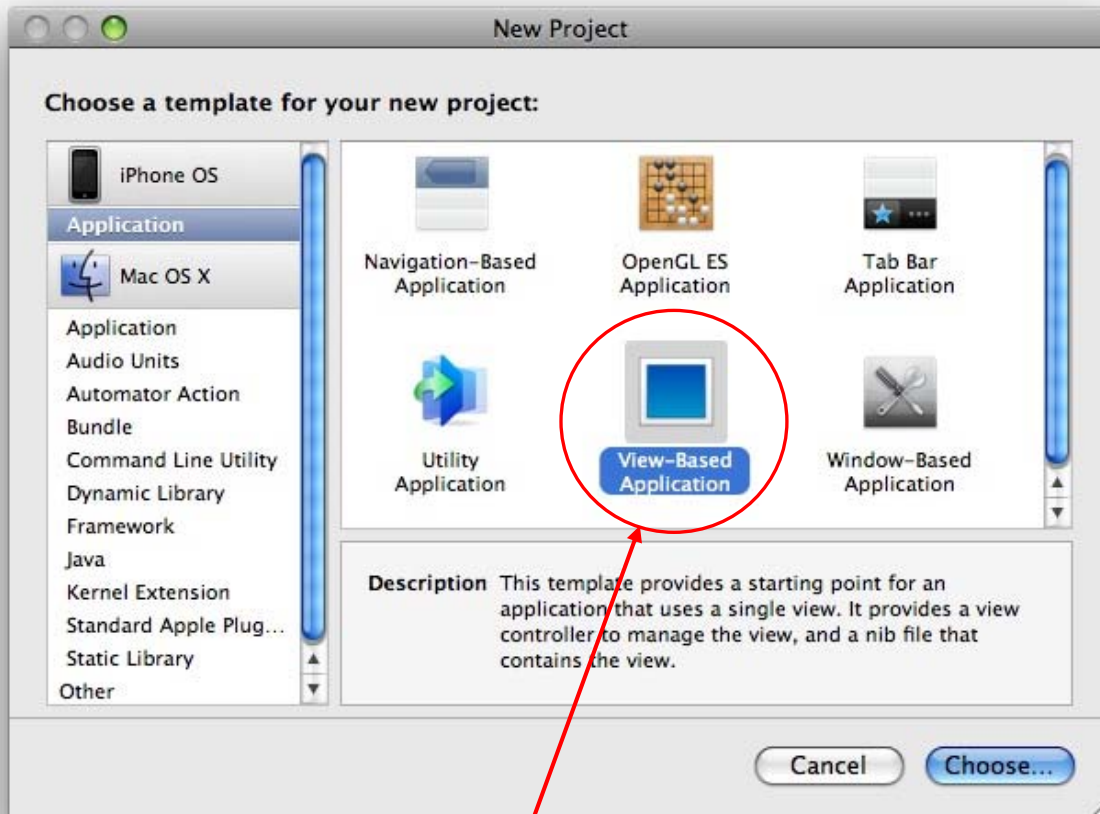
In order to do this tutorial and to develop apps for the iPhone, you need to have the following tools:

- An Intel-based Mac running Leopard (OS X 10.5.5 or later).
- The iPhone SDK. You can download the free version from the Apple web site. The SDK includes all of the necessary software tools that you will need to develop the apps. However, the free version will only allow you to run your app in the iPhone simulator running on your development machine. In order to run your app on the iPhone or iPod Touch, you will have to get the paid version which costs \$99 for the standard version.

## 2. Creating Your Project

The main tool you use to create applications for the iPhone is Xcode—Apple’s IDE (integrated development environment).

Launch Xcode. By default it’s in /Developer/iPhone Applications. If the welcome window comes up, you can just close it. Create a new project by choosing from the menu File > New Project. The notation used here means that you first click on the File menu and then you click on New Project. You should see a window for choosing a template for your new project like this.

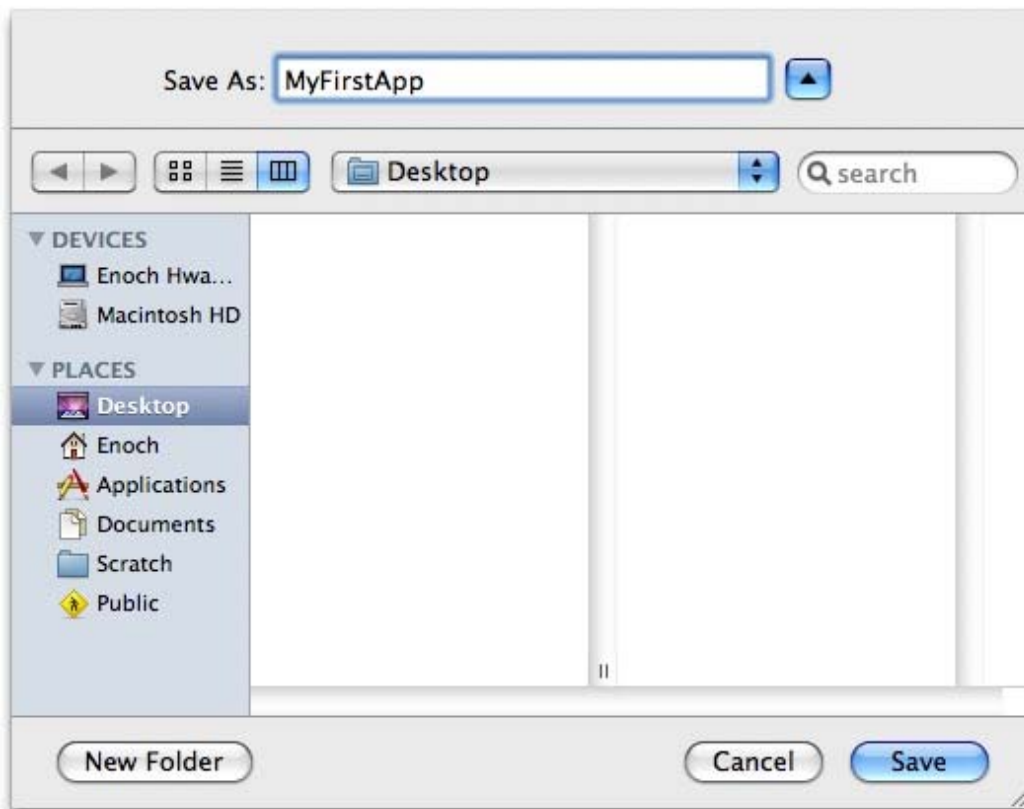


Select this template

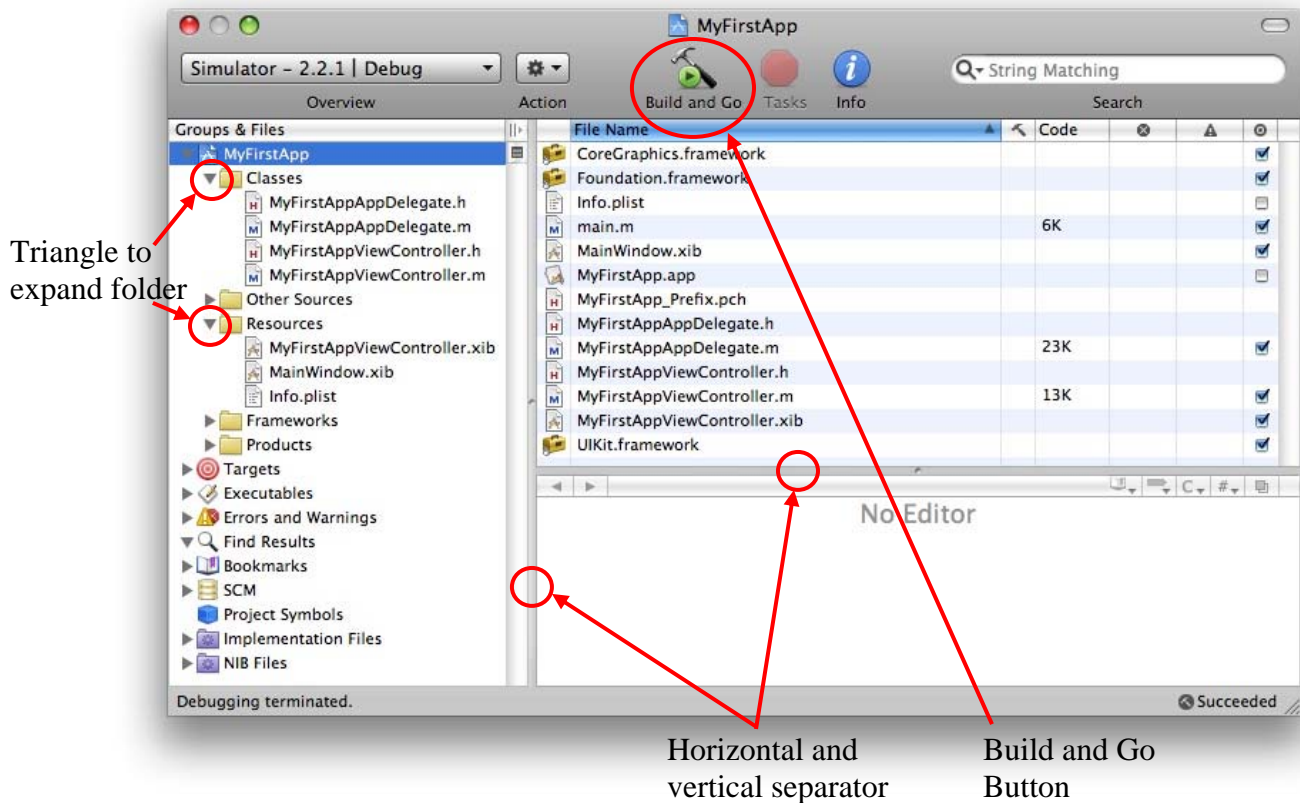
In the left side of the window, make sure that Application under the iPhone OS section is selected. If not, then just click on it to select it.

Select the View-Based Application template icon, and click Choose.

A New Project window appears to allow you to specify the name of your project and where you want to save your project. Select the Desktop for the location and type in MyFirstApp as the Save As: name for the project and click the Save button like this.



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 three sections or panes. The left pane shows the Groups and Files in your project. The top-right pane shows the files inside the folder that you have selected in the left pane. You can change the size of each pane by dragging the horizontal or vertical separator border. In the Groups and Files pane, expand the Classes folder by clicking on the triangle next to the folder. Also expand the Resources folder. By expanding the folder this way, you can also see the files inside the folder.

Since Xcode has included everything that you need to run the app inside your newly created project, you can now build and run the application by choosing from the menu **Build > Build and Go** or by clicking the **Build and Go** button in the toolbar.

The iPhone Simulator should launch automatically, and when your app starts up you should see a blank-white screen. The iPhone Simulator simulates (almost) exactly everything that you will see if you were to run the app on your iPhone or iPod Touch. 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-white 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 iPhone Simulator by choosing from the menu **iPhone Simulator > Quit iPhone Simulator** and continue.

### 3. Starting Interface Builder

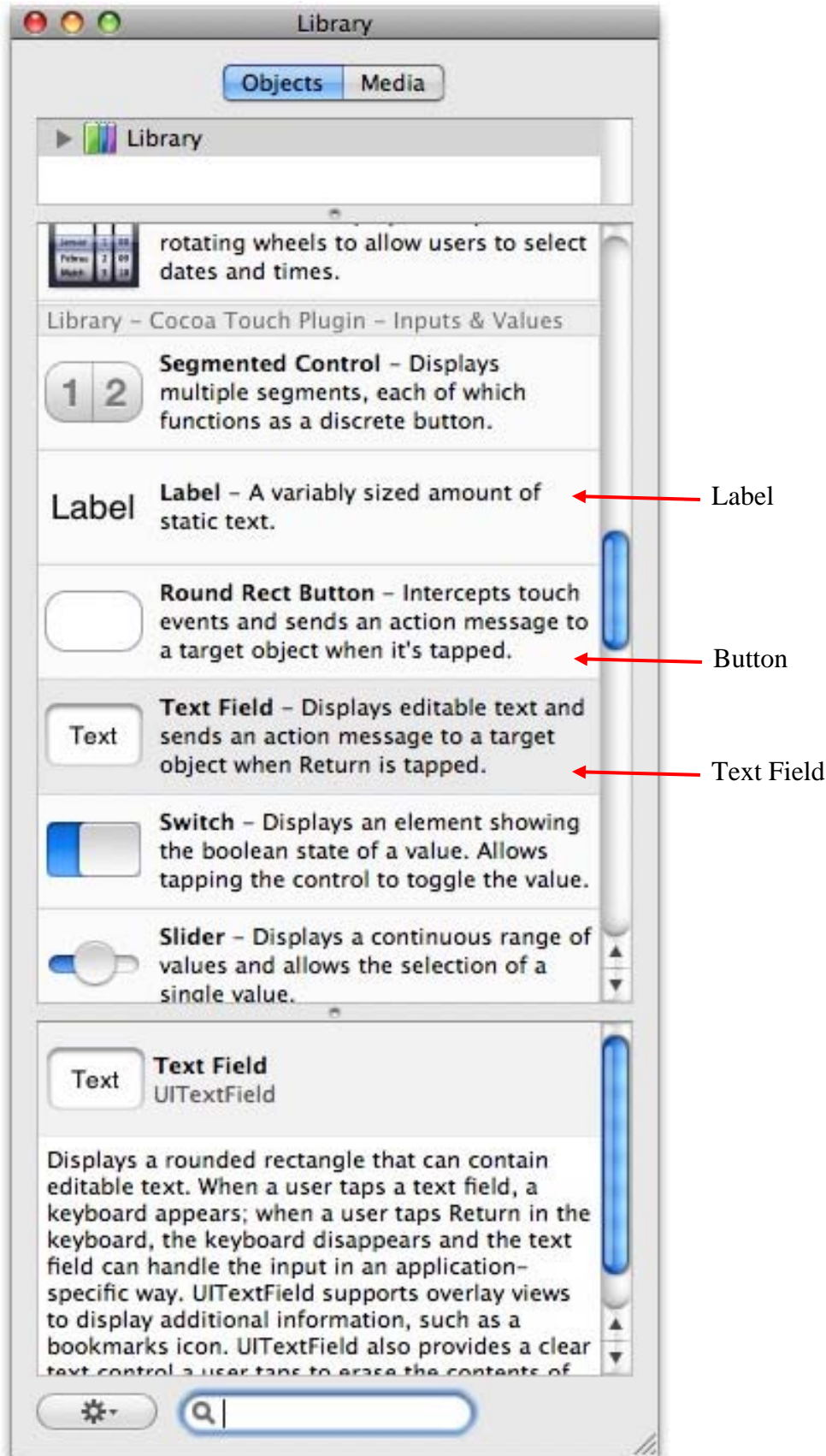
Interface Builder is the application you use to add user interface objects into your app. These objects are saved in a nib file having the name `MyFirstAppViewController.xib`. (Yes, all nib files have the extension `xib`. The file name begins with your project name with the word `ViewController` appended to it.)

Double-click the file `MyFirstAppViewController.xib` in the Groups & Files pane under the Resources folder in your project window to start Interface Builder. You should see the following window with the three icons.

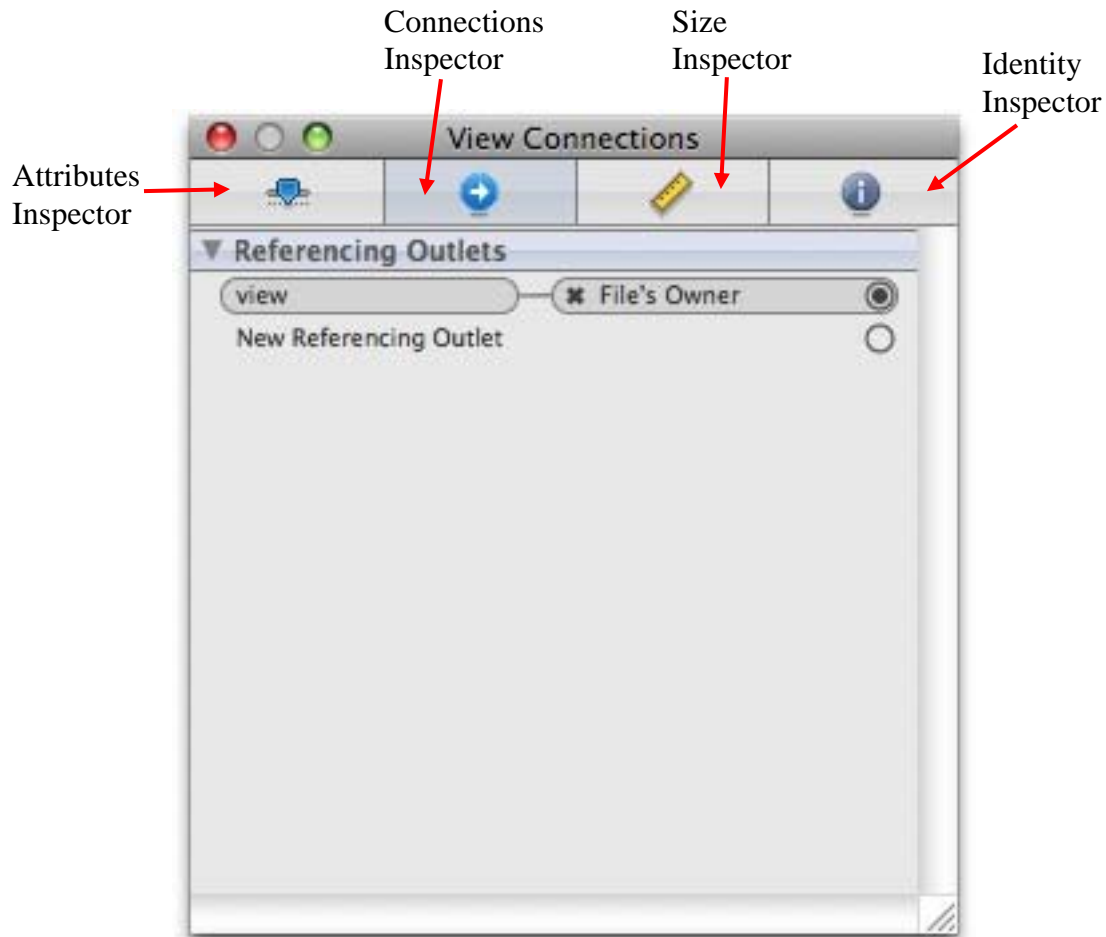


Double-click the View icon to see what your current iPhone user interface looks like. This view window allows you to customize the user interface. You should see the same blank-white screen that you saw earlier from the iPhone Simulator.

Objects such as buttons, labels, text boxes, and so on, that you have available to add onto your iPhone user interface are found in the Library. Bring up the Library window by choosing **Tools > Library**. Scroll down and scan through the list to see the objects available for you to use. You should recognize some of the objects that you have used in other apps such as the Label, Rounded Rect Button, and Text Field as shown next.



Another window that you will need to use frequently is the Inspector window. The Inspector allows you to customize the objects that you have added to your iPhone user interface, and to link them to your code. Yes, you will be writing code using the Objective-C programming language<sup>1</sup>. Bring up the Inspector window by choosing from the menu Tools > Inspector. You should see something similar to the following.

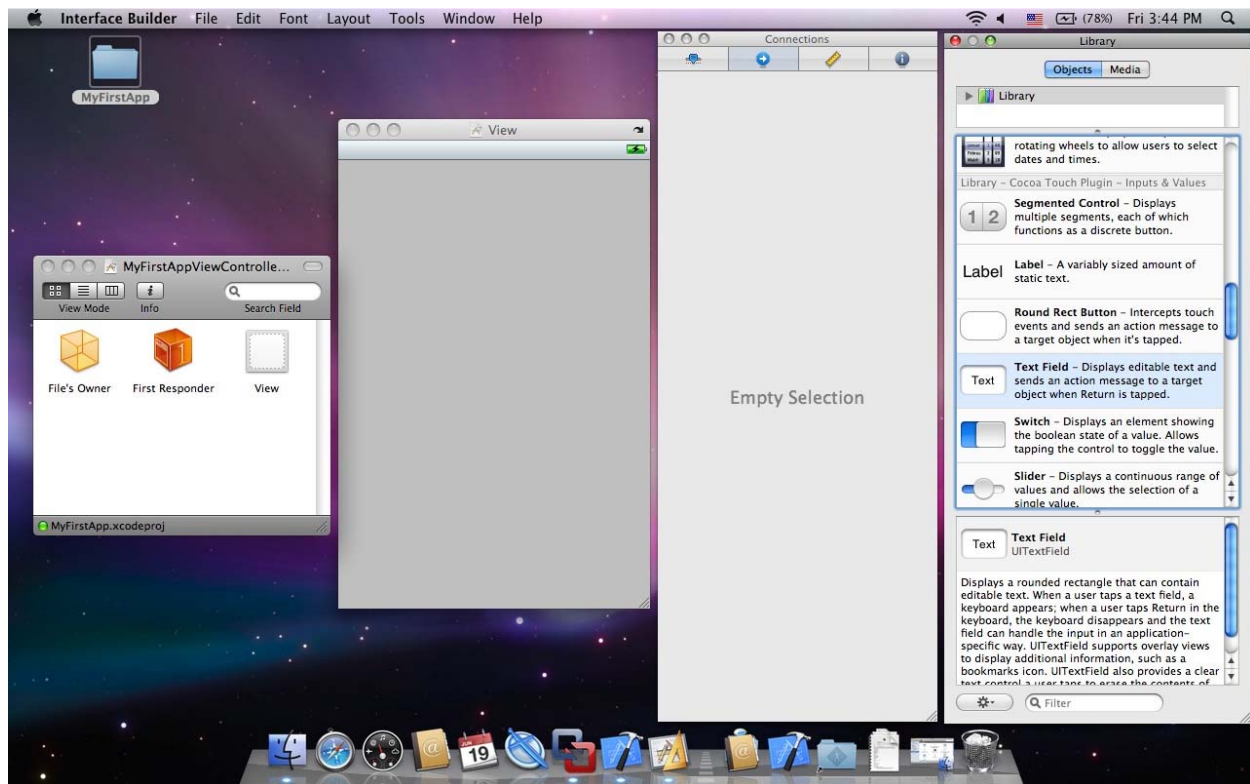


Within the Inspector window, there are four different tabs across the top of the window for accessing the four different inspectors. Going from left to right, they are Attributes, Connections, Size, and Identity inspectors. You can also access these inspectors directly from the menu by choosing Tools > Attributes Inspector or one of the other three. Click on each of the Inspector tab to familiarize yourself with what is in each one. For now, do not make any changes.

Your desktop should now look something like the following with the four windows.

---

<sup>1</sup> The Objective-C language is an object-oriented language very similar to C and C++.



#### 4. Adding Objects Into Your App

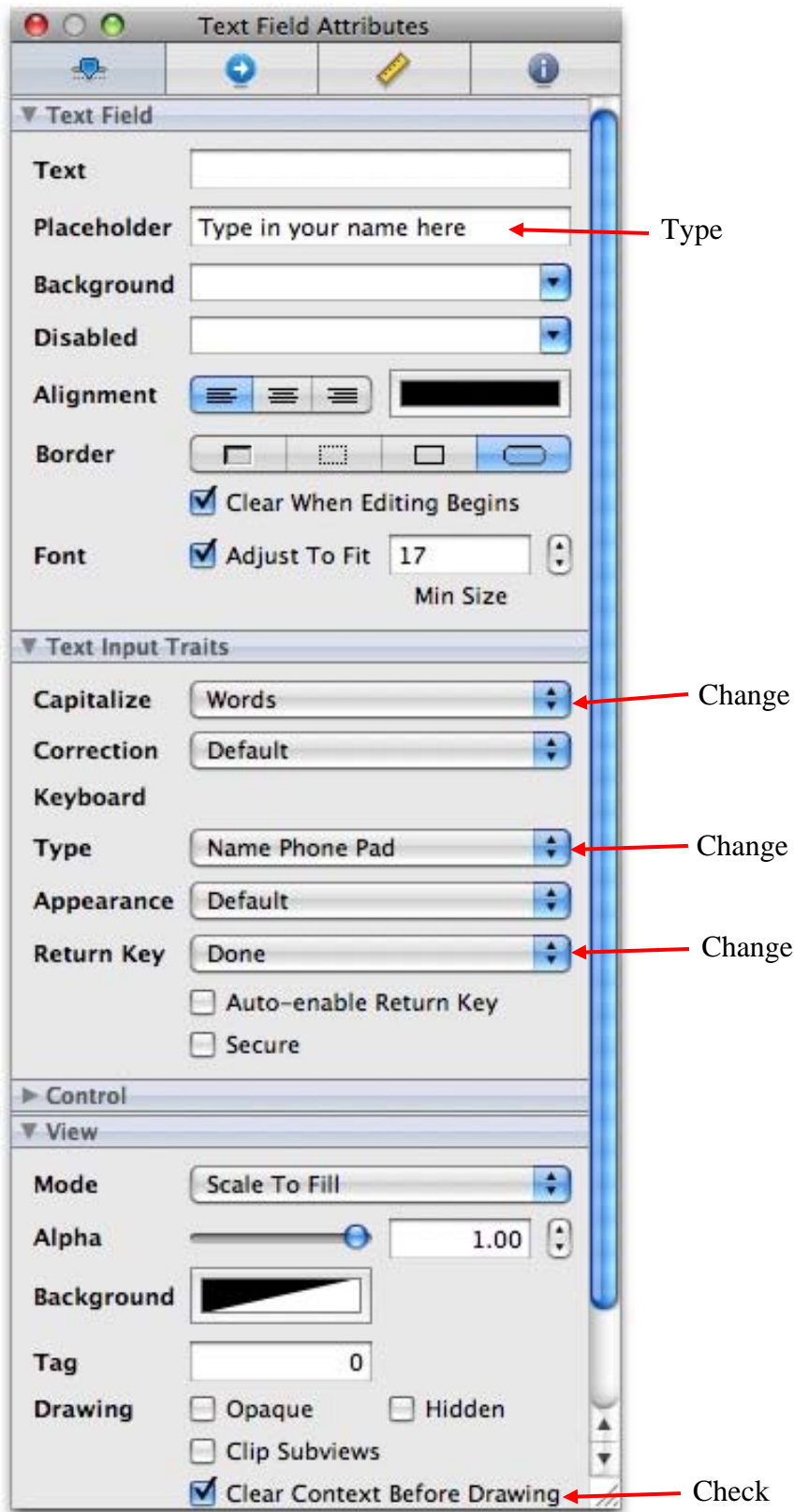
We are now ready to add some objects from the Library into our iPhone user interface.

First find the Text Field object in the Library. Once you have found it, drag it onto the iPhone user interface view. Notice that when you move it close to the edge of the view, blue lines will appear to guide you in placing the object. Place the Text Field object close to the top left corner of the view like so.



The two little circles located at both ends of the Text Field box allow you to change the width of the box by dragging them. When you place your mouse cursor over the circle, it changes to a double-pointing arrow. Drag the circle on the right side to the right until the blue vertical guide line on the right side appears.

With the Text Field box still selected, click on the Attributes Inspector and you should see this.

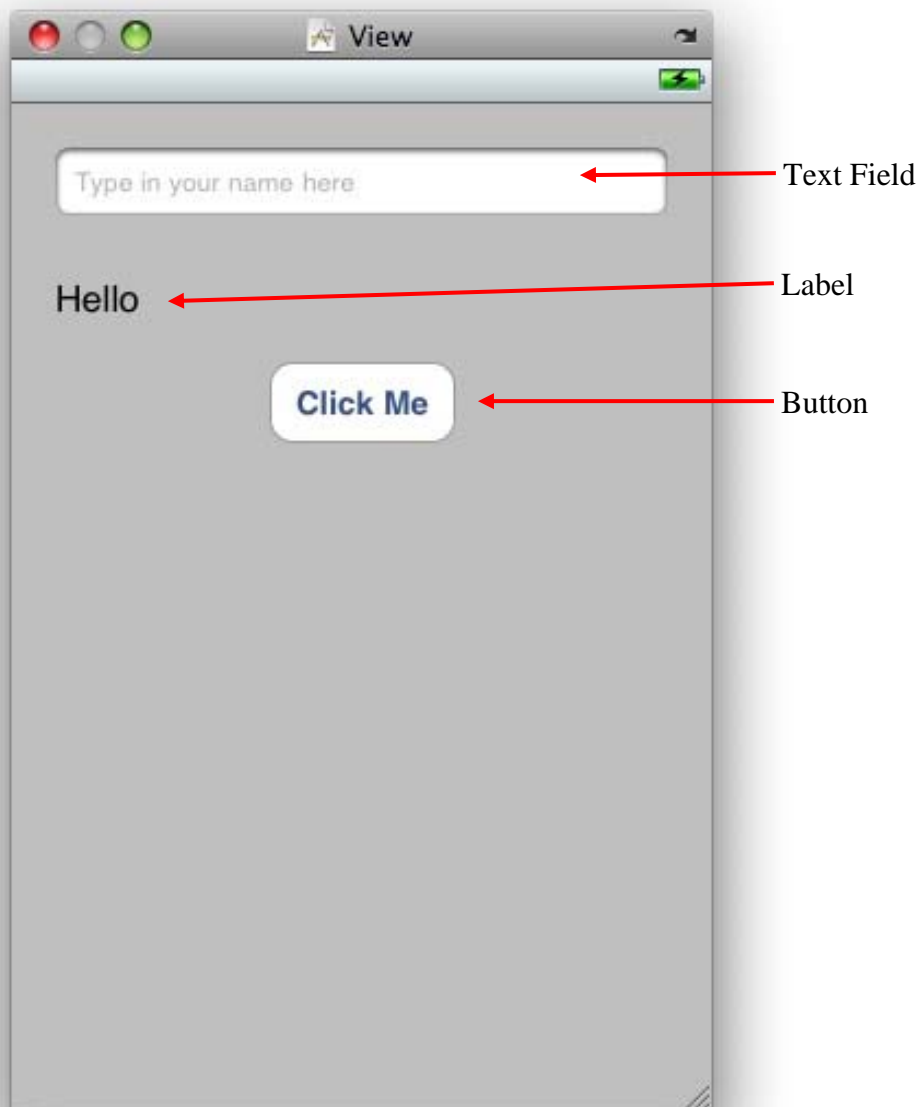


The Attributes Inspector allows you to customize the properties associated with the selected object, which in this case is our Text Field. In the Placeholder entry area, type in the words “Type in your name here.” We want to capitalize each word entered, so under the Text Input Traits section, for Capitalize select Words. For the Keyboard Type, select Name Phone Pad. For the Return Key, select Done. Finally, in the View section towards the bottom of the Attributes Inspector, put a check mark next to Clear Context Before Drawing.

Next, drag a Label object from the Library to the iPhone view and place it just below the Text Field. (Go back and look at the picture on page 6 if you forgot what the Label object looks like.) Align the left edge of the Label with the Text Field box using the blue alignment line that appears. Change the label name by double-clicking on the Label and typing in the word “Hello.” Notice that you can also change the label name from the Attributes Inspector by typing the name in the Text Field under the Label section. Again, change the width of the Label to be the same as the Text Field.

Finally, drag a Rounded Rect Button from the Library to the iPhone view. (Go back and look at the picture on page 6 if you forgot what the Button object looks like.) Place the Button just below the Label and center it horizontally. Notice the center blue guide line appears to help you to center the object. Double-click inside the Button and the text insertion cursor should appear. If it doesn't, then double-click on it again until it appears. Type the words “Click Me” and hit the Enter key.

Save your iPhone user interface by choosing from the Interface Builder menu File > Save. Your user interface should now look like this.



Now, let's go back to Xcode to build and run your app and see what happens. Do you remember how to do that? In the Xcode project window, click on the **Build and Go** button. Again, the iPhone Simulator comes up, but this time instead of the blank-white screen, you also see the three objects that you have added. Play around with it; click on the objects and see what happens. Nothing much, not even when you click on the button that says "Click Me." When you click in the text box, the keyboard pops up and you can type your name or whatever inside it. However, once the keyboard is up, there is no way that you can get rid of it—at least not for now. What we want to do is to type your name in the text box and then when you click on the button it will display a personalized hello message in the label. After you are done playing, quit the iPhone Simulator by choosing iPhone Simulator > Quit iPhone Simulator.

## 5. Writing Code

In order to get the objects to do something when you click on them, you will have to write code. You do this from the Xcode project window, so go back to the Xcode project window now. During a project development cycle, you will frequently be going back and forth between Xcode and Interface Builder, so just get used to that now.

There are two files, `MyFirstAppViewController.h` and `MyFirstAppViewController.m`, that we will be modifying, both of which are located inside the `Classes` folder in the `Groups & Files` pane of the project 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 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 `MyFirstAppViewController.h` file and the contents of the file will show up in the bottom-right pane 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. 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 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 understand 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 Enter 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.

```
//  
// MyFirstAppViewController.h  
// MyFirstApp  
//  
// Created by Enoch Hwang on 6/19/09.  
// Copyright La Sierra University 2009. All rights reserved.  
//  
  
#import <UIKit/UIKit.h>
```

```

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

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

- (IBAction)changeGreeting:(id)sender;      // method changeGreeting
- (IBAction)textFieldDoneEditing:(id)sender; // method textFieldDoneEditing

@end

```

The two lines

```

UITextField *myTextField;
UILabel *myLabel;

```

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 changeGreeting
- (IBAction)textFieldDoneEditing:(id)sender; // method textFieldDoneEditing

```

are for declaring the two methods `changeGreeting` and `textFieldDoneEditing` that we will have. The method `changeGreeting` will be executed when the button is pressed to change the welcome message, and the `textFieldDoneEditing` 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 do a Build and Go. 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 `MyFirstAppViewController.m` file. Single click on your `MyFirstAppViewController.m` file and the contents of the file will show up in the bottom-right pane of your project window. This file basically contains the code for our two methods `changeGreeting` and `textFieldDoneEditing`. Modify the file so that it matches the following listing.

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

- Lines of code that are bracketed by `/*` and `*/`, and shown in green, are comments.
- There are two lines (one in the `changeGreeting` method and one in the `textFieldDoneEditing` method) that reach close to the right margin and doesn't end with a

semicolon ; because they are 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 them in the editor, you need to type them on one line instead of two lines. Inside the editor, they will not be wrapped around to two lines.

- For the dealloc method, there is already one declared at the end of the template listing. Do not duplicate it, but just add the new lines into the existing one.

```
#import "MyFirstAppViewController.h"

@implementation MyFirstAppViewController

@synthesize myTextField;
@synthesize myLabel;
@synthesize myString;

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

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

// the dealloc method
- (void)dealloc {
    [myTextField release];
    [myLabel release];
    [myString release];
    [super dealloc];
}
```

These two lines need to be typed on one line.

These two lines need to be typed on one line.


Note that this method is at the end of the template listing.

Just to make sure that you have not made any mistakes in your typing, once again, do a Build and Go 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.

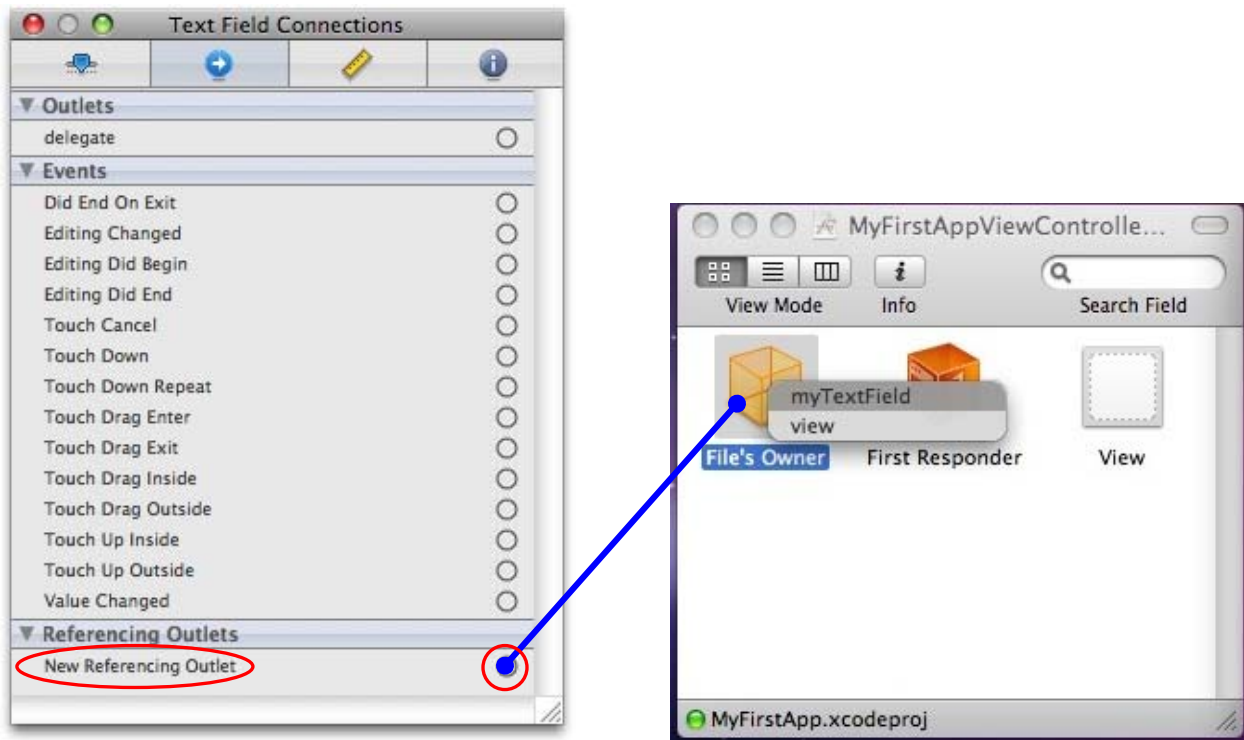
## 6. Connecting The Objects To The Code

We are now ready to connect the objects in your iPhone user interface to the code that you have just typed in. Guess where we do this? Yes, back in the Interface Builder. There are two types of connections that we need to make, Outlets and Events. Outlets are for objects to output

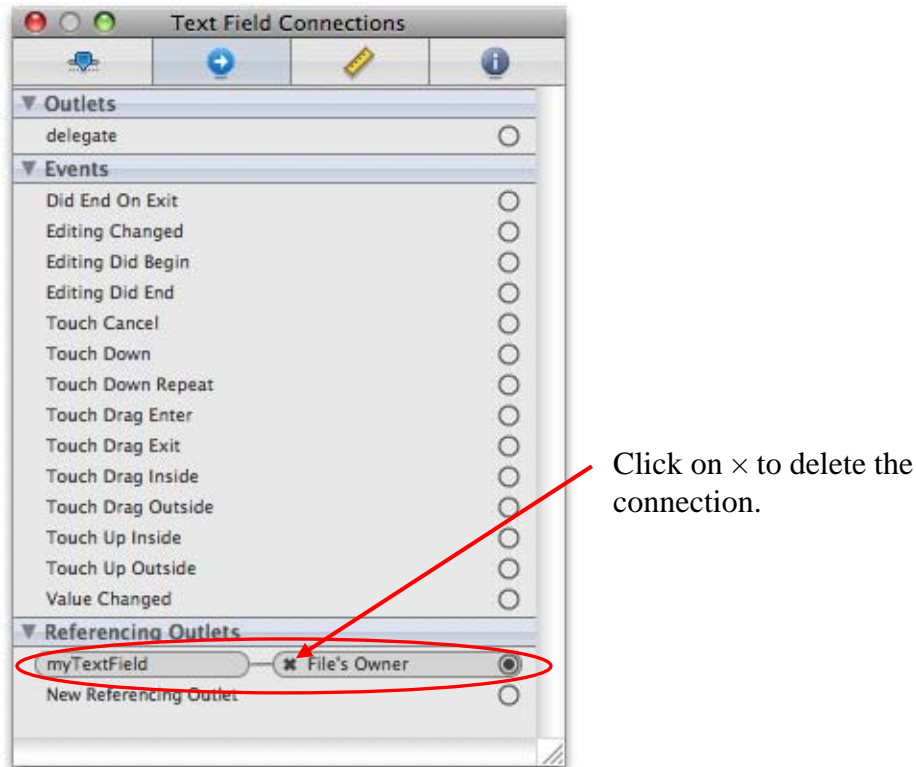
messages. So if you want an object to output some text, you need to connect its outlet to the corresponding name that you have given it in your code. Events are methods that you want an object to perform when that object is triggered. So if you want your button to perform a particular method when it is clicked, you need to connect that button clicked action to that method.

In the Inspector window, select the Connections Inspector. You can do that by either clicking on the second tab , or choose Tools > Connections Inspector. Single-click on the Text Field box in the View window to select it.

Towards the bottom of the list in the Connections Inspector, 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 File's Owner icon. A blue line will follow your mouse pointer as you drag the mouse. When you release your mouse on the File's Owner icon, a selection box will pop up. Select myTextField.



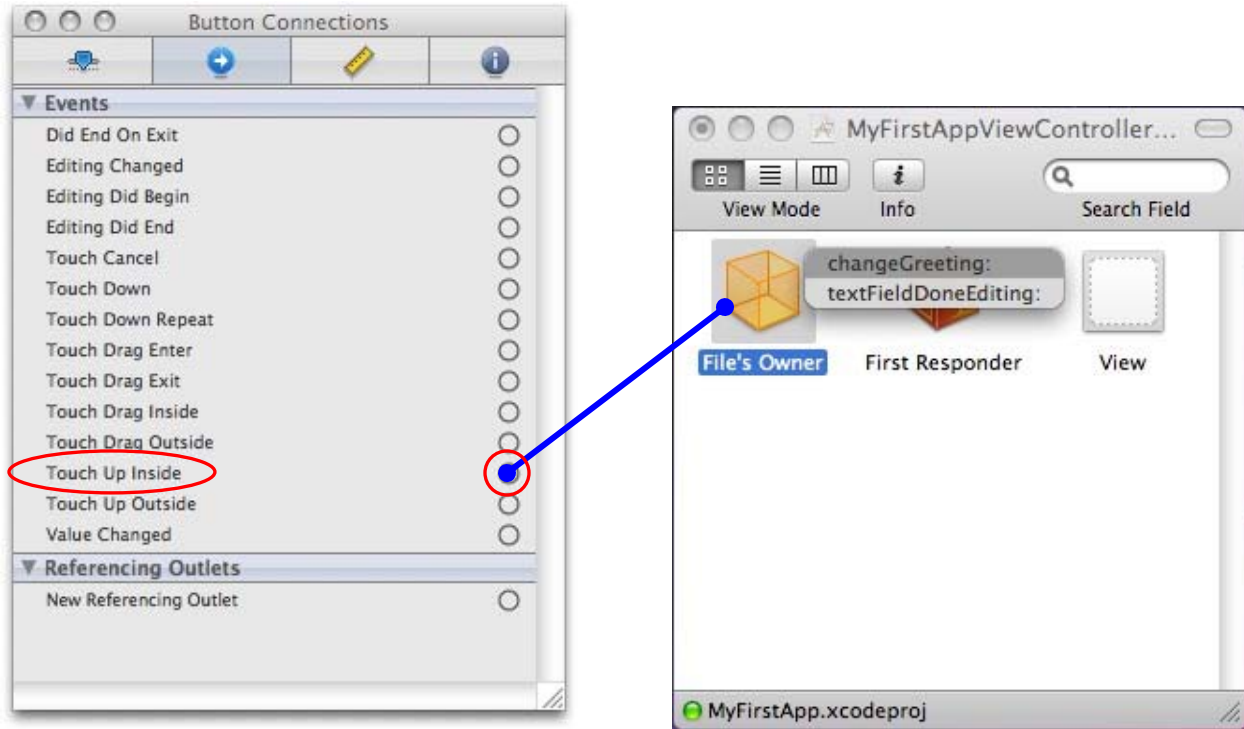
Once you have done that, the Connections Inspector for the Text Field box will look like this.



If you made a mistake, you can click on the  $\times$  next to File's Owner that is connected to myTextField to delete the connection and then re-connect it. In fact, it will be a good exercise for you to click on the  $\times$  now to delete the connection even if it is correct and then re-connect it back.

Do the same thing for the label. Select the Label in the View window. Drag from the empty circle next to New Referencing Outlet to the File's Owner 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 File's Owner icon as shown next. From the pop-up menu, select the changeGreeting method.



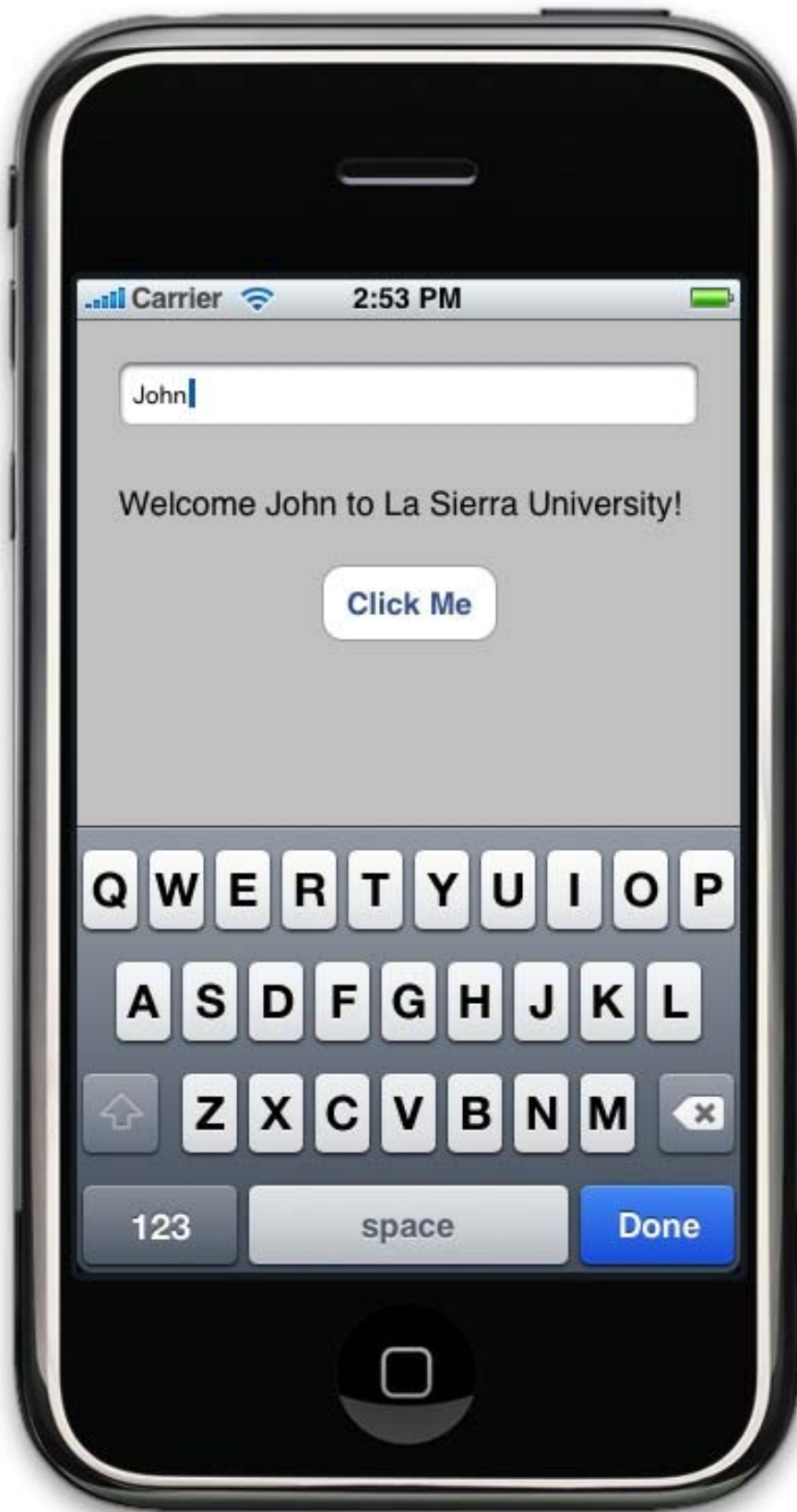
While running this app, after entering your name in the Text Field box and you press the Done button on the keyboard, you want to execute the `textFieldDoneEditing` method to remove the keyboard. When you press the Done or Return button on the keyboard, the Did End On Exit event for that object is triggered. So select the Text Field object, and connect the Did End On Exit event to the `textFieldDoneEditing` 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 `textFieldDoneEditing` from the pop-up menu.

To double check that you have made all the connections correctly, select the File's Owner icon. You should see the following connections in the Connections Inspector window. If you don't, then you need to reconnect the objects correctly.



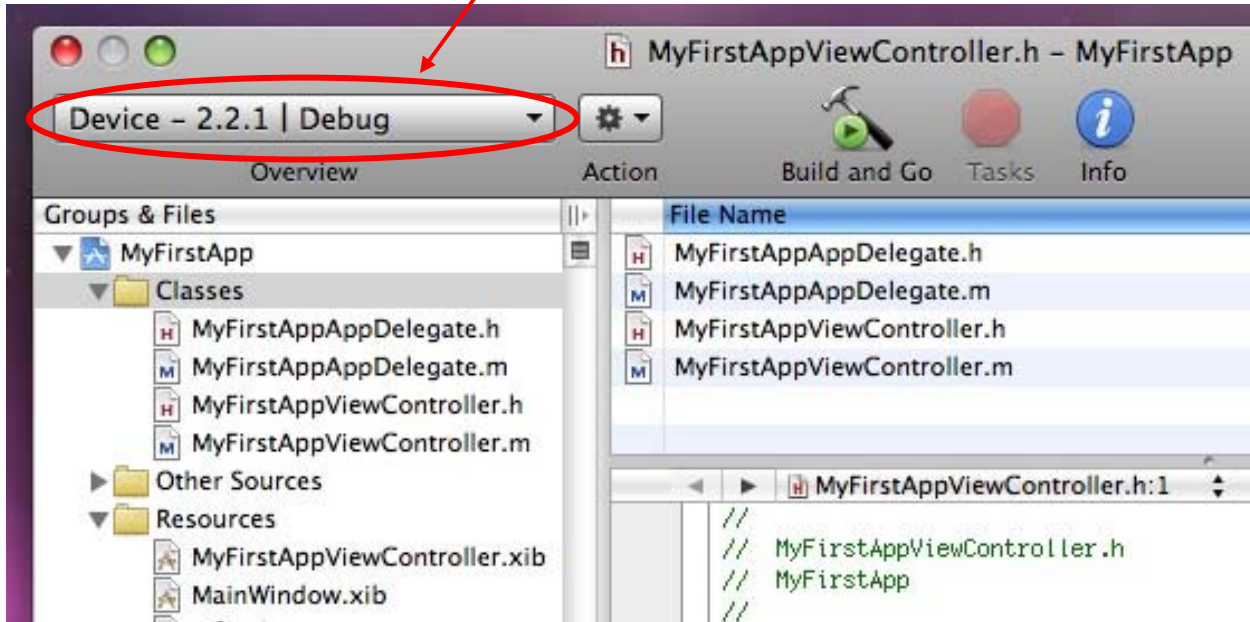
Notice that there is one extra connection that we did not make, and that is the View connection. This connection was made automatically for you when you selected the View-Based Application template for your project. However, if you accidentally deleted this connection, you can manually connect it back by selecting either the View icon or an empty area in the View window, then drag from the empty circle next to New Referencing Outlet to the File's Owner icon, and select view.

We are all done. Save the changes that you have made in Interface Builder, go back to Xcode and do a Build and Go to try out your very first app. You should see something like this. 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.



If you want to try this app on you iPhone or iPod Touch, all you need to do is connect your iPhone to your computer. From the Xcode project window, select Device – iPhone OS 2.2.1 from the drop-down menu at the top left corner of the window, and then do a Build and Go again<sup>2</sup>. If the download is successful, you should see a white icon on your iPhone with the name MyFirstApp. You can now tap it to run it.

Select Device – iPhone OS 2.2.1.



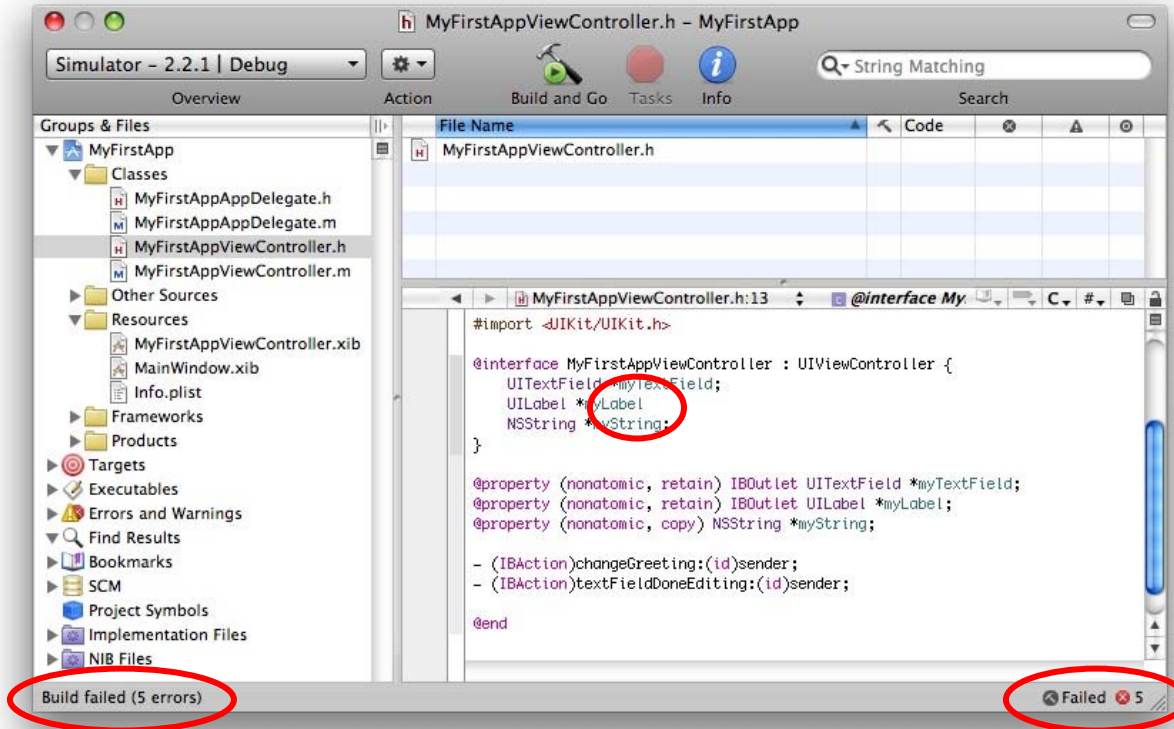
## 7. Exercises

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.
2. Modify the app so that the text displayed in the Label is red instead of black.
3. When you type in a name that is long, the font size in the Label is automatically made smaller. Modify the app so that the font size for the Label does not change.
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.
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”.

---

<sup>2</sup> In order for this to work, you’ll need to have the paid version of the SDK and have followed the instructions for installing it to your computer.

- In the MyFirstAppViewController.h file where you typed in the label name declaration line `UILabel *myLabel;`, remove the semicolon at the end of the line. Save the file and then do a Build and Go. What happens? First, the iPhone Simulator did not start. More importantly, look at the status line at the bottom of the Xcode project window, it says that the Build has failed with 5 errors as shown here.



Click on the red  $\otimes$  at the bottom right corner. A window comes up and tells you what and where the error is. Notice that in the first error message, it does not say that something is wrong with the `myLabel` line. Instead, it says that the syntax error is before the `NSString` keyword. Another thing to realize is that missing one little semicolon can cause 5 errors!

Experiment by deleting other characters and see what error messages you get.

- Remove all unnecessary code in the `changeGreeting` method and the `textFieldDoneEditing` method. You should be able to remove three lines in the `changeGreeting` method, and two lines in the `textFieldDoneEditing` method.
- Reverse the two connections to the two methods so that when you click on the Click Me button it will execute the `textFieldDoneEditing` method instead of the `changeGreeting` method, and when you press the Done key on the keyboard it will execute the `changeGreeting` method instead of the `textFieldDoneEditing`.
- Change the label name `myLabel` with `myFirstLabel`. Note that there are several places that you will have to make the change, some in the `MyFirstAppViewController.h` file and some in the `MyFirstAppViewController.m` file. Furthermore, you will also have to redo the connection for the Label in the Connection Inspector.

10. Add a second Label to the app that will display the same thing as for the first Label, but in a different color.
11. 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.