iPhone Development Tutorial I
January 2011

Manish Chaturvedi
Parminder Saini

The following tutorial introduces XCode IDE and some basic concepts towards developing iOS SDK based applications using Objective C.

Contents:

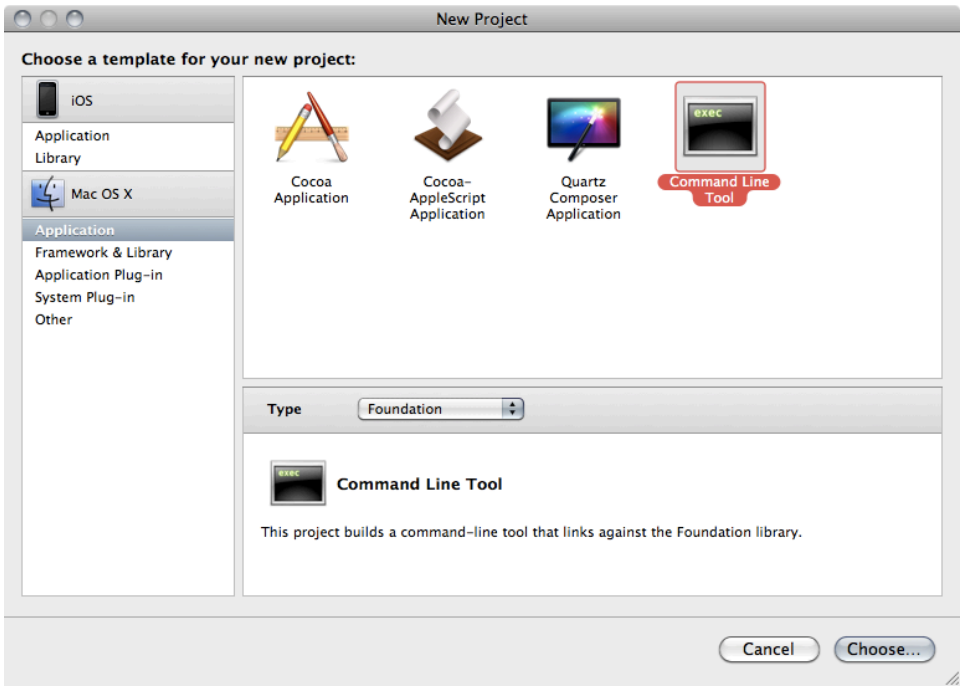
Part A
Objective C

This section steps through creation of Objective C source code using the XCode IDE, and using some Objective C programming constructs.

Creating a Objective C project within XCode

- **Open XCode**
- Create a Project Appropriate for Non GUI Source Code
  - "**File > New Project >MacOS>Application>Command Line tool**".
    - Pick "Foundation" from the Type Drop Down.

This creates an Objective C file with extension **.m,** and contains a **main** method

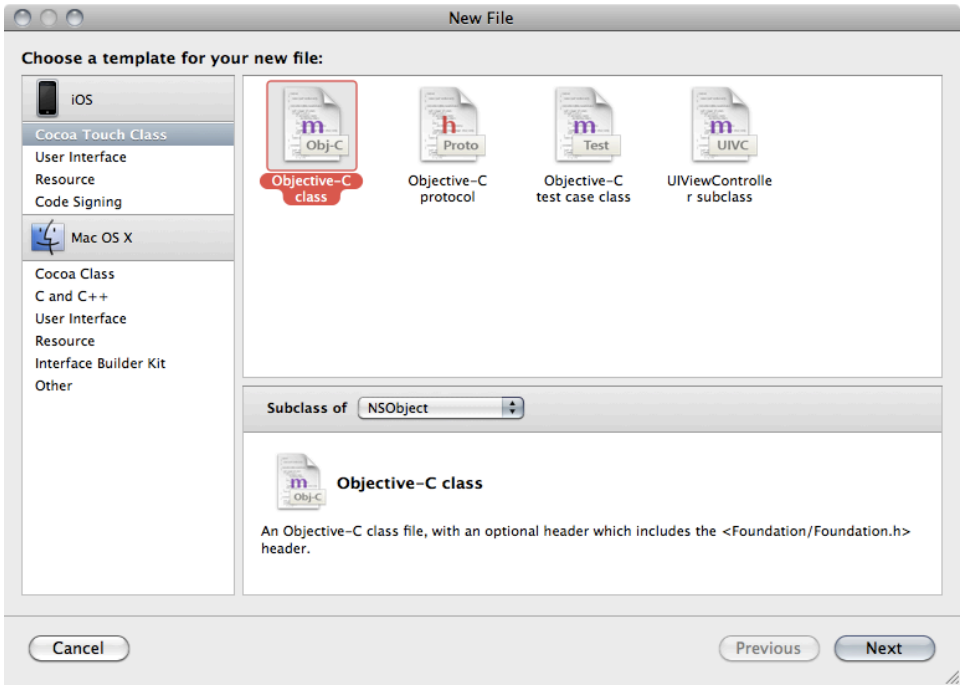


Adding Custom Class Files

We need to explore Object Oriented aspects of Objective C. This entails creating new classes in the existing project.

- "**File > New File > iOS > Cocoa Touch Class > Objective-Class**"
- Subclass it to **NSObject**.

This creates a Header and Implementation file (.h and .m) with the supplied name.
(e.g. CustomClass.m)



Exercise

- **Create a Class** called **Animal** which Inherits from **NSObject**
- Modify Animal header file
    - Add Instance Variables
        - Age (float)
        - Name (NSString)
    - Add Public Methods
        - void makeNoise
        - void setName
    - Make age a Property attribute
- Modify Animal Implementation file
    - Implement Class methods
    - Synthesize Age
- Create a **Sub Class** of **Animal**- Call it **Dog**
    - **Overide** makeNoise method, Ensure it makes Dog appropriate noises

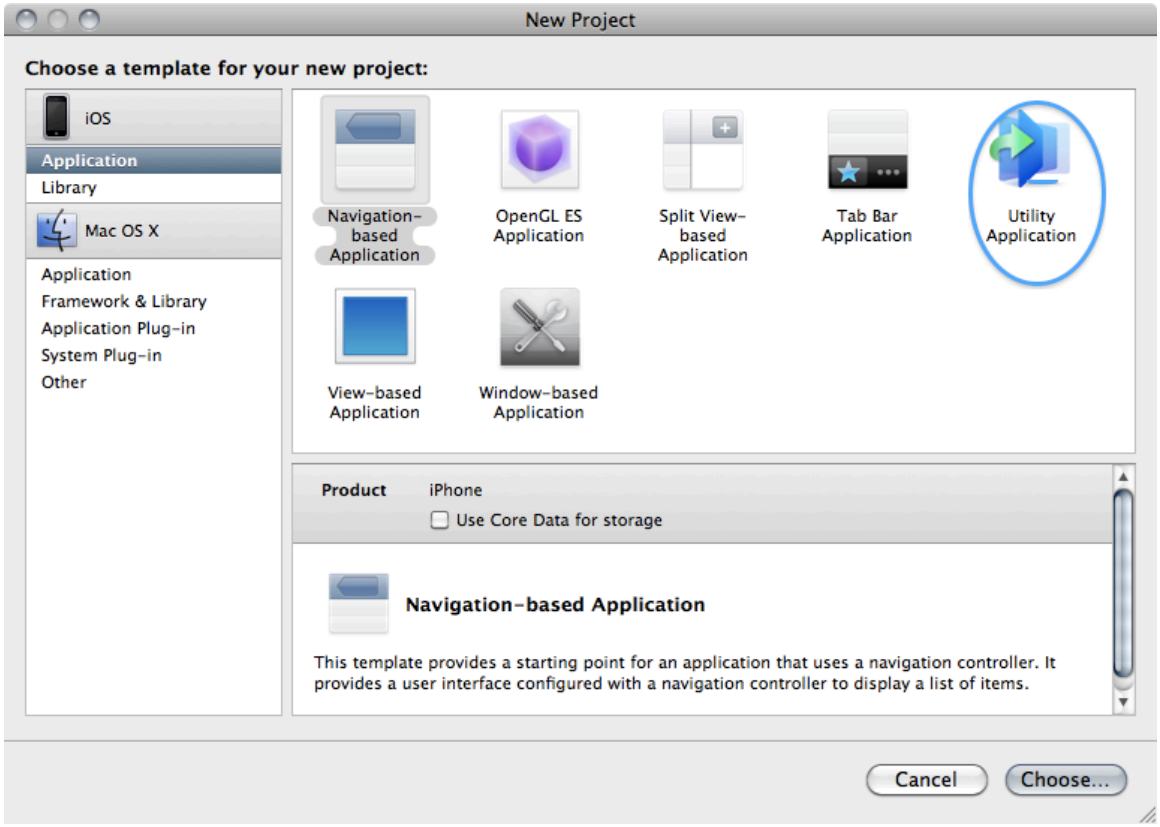
- **Modify the Main Class**
    - Access the Animal Object
        - Make Animal act like a Dog
        - Give it a Name
        - Set the Age
        - Make Dog like Noise (Use NSLog to Print to Console)

Part B
iOS Utility and View Application

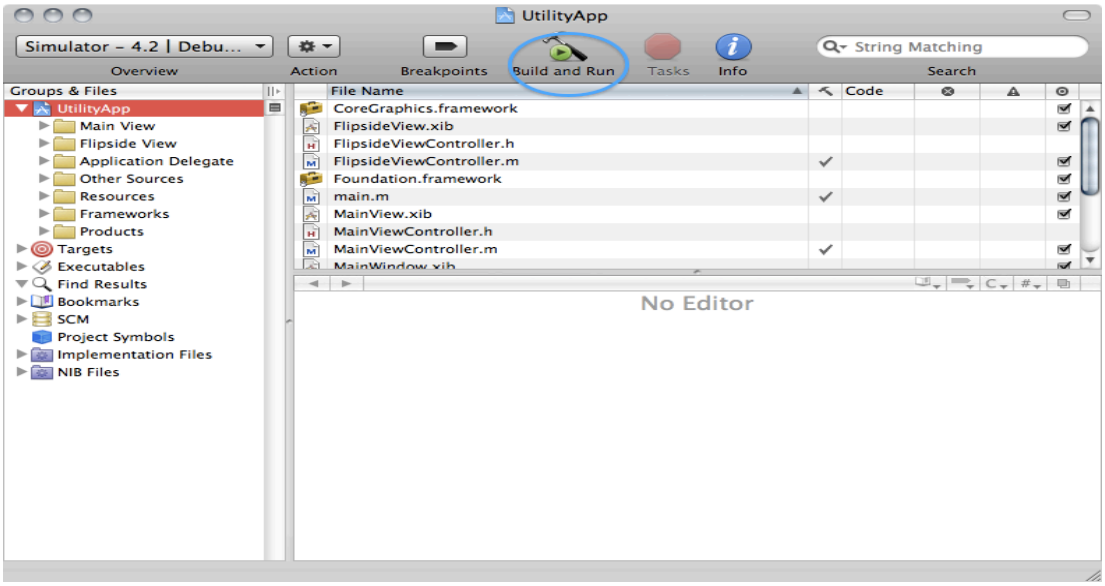
This section contains a tutorial exploring the Utility Application type in the iOS SDK. The stock application on iPhone is an example of a Utility Application type.

Creating a Project

Create an iPhone utility program view by selecting "**File > New Project**" and Selecting "**iOS > Application > Utility Application**".

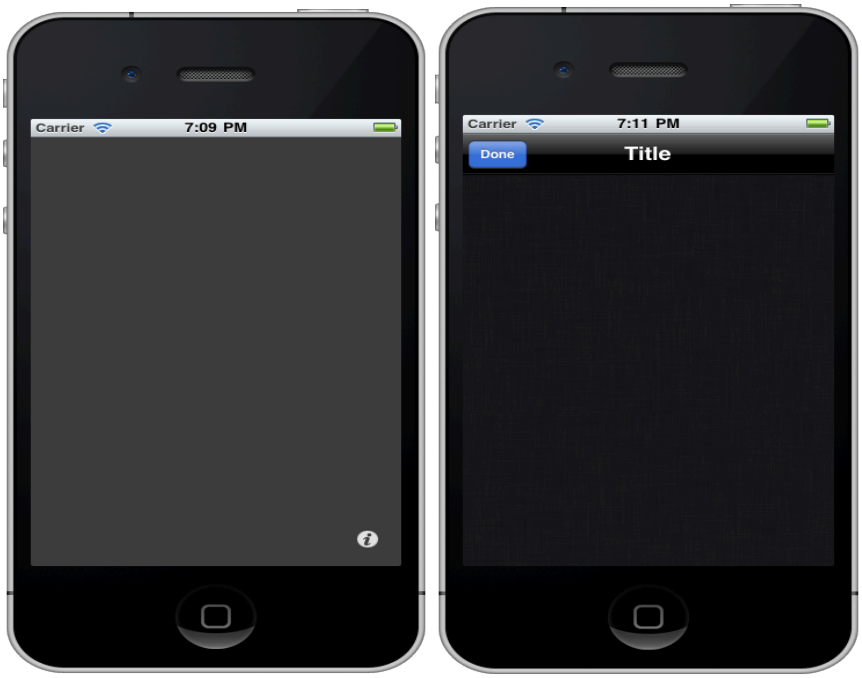


Run the application by Clicking "Build and Run" button as shown below:-



Basic Utility Application

Two views are presented: *i* button on lower right of the main view flips the page and "Done" button on the flipside view flips the page back.

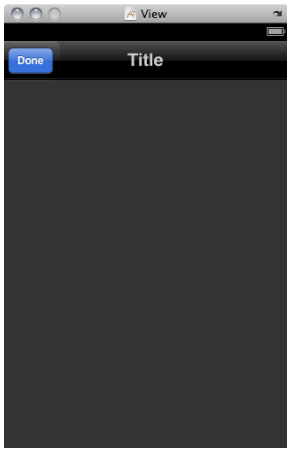


Time to **Explore the XCode Interface** –

The Interface Builder files are found under the "**NIB**" directory in the Groups and Files panels well as under Resources. Implementation Files folder contains the Classes.

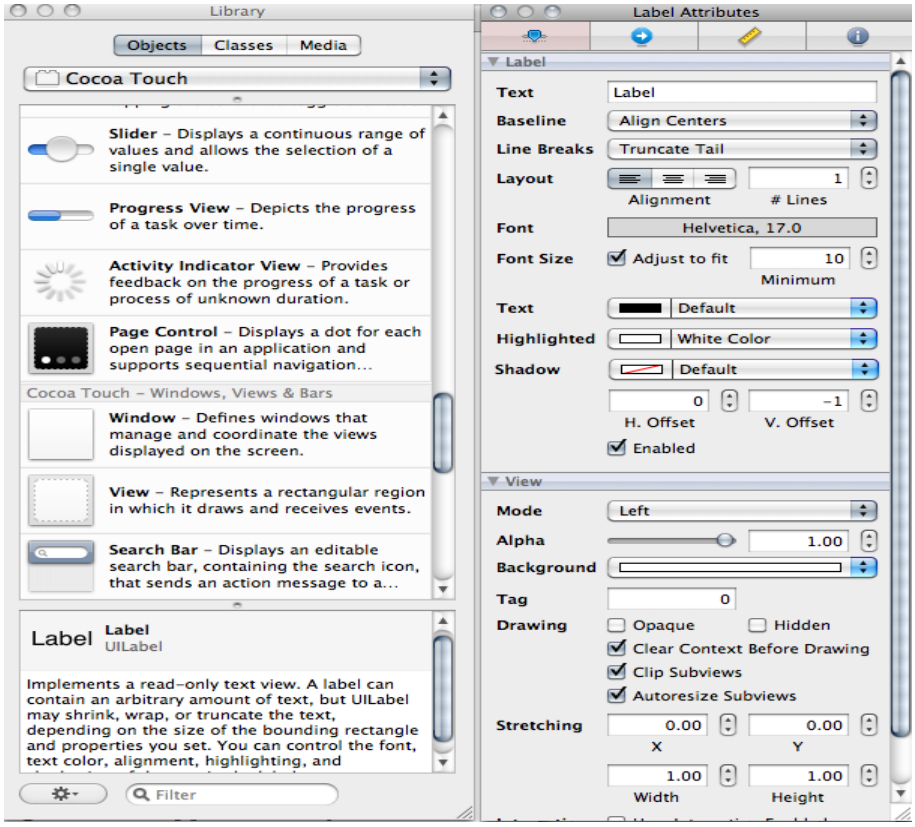
Adding Widgets to the Views

The two views can be brought up in the **Interface Builder** in order to add components –
Double Click FlipSideView.xib file.



You should get a set of window similar to the ones shown above.

To the main view we seek to add a Label. Select "**Tools > Library**" to bring up the library tool. Scroll down to "**Cocoa Touch - Inputs and Values**" section to locate the Label widget. Access "**Tools > Attributes Inspector**" to display and set Label properties.



Add a Label to the Flip View of the Interface.

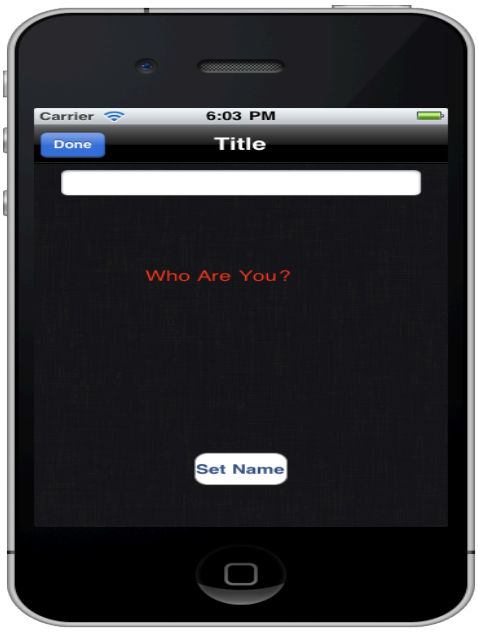
- Stretch label to fit the width of the view,
- Double click the label to Enter string “Who Are You”.
- Change the Text color from default of Black using the Attributes Inspector.

Open the **Flip View** nib if it is not open already –

- Add a Text Field (UITextField)
- Add a Button (UIButton).

Notice Guideline appear to help position the widgets.

The resulting views should be close to following image:-



Save the XIB files and launch “Build and Run” from the XCode editor.
Explore the widgets, particularly the Text edit box.

Outstanding Issues

On running the project following short comings can be noticed:-

- Keyboard for text entry is not dismissed as expected
- The Button on Flip view does not have any behavior attached to it. We want to modify the Label text using the text entry input when the button is pressed.

What is Delegation?

A Class is a delegate of a field –> The Class supports operations to manipulate the field. FlipsideViewController Class needs to be a delegate of the UITextField.

Modify **FlipsideViewController.h** interface to handle protocol UITextFieldDelegate

```
@interface FlipsideViewController : UIViewController
<UITextFieldDelegate>
```

`IBOutlet`

We need some way to connect the UI elements to the code in the ViewController. IBOutlet is a marker that allows us to do that. Further connections between the GUI and this class will be made using the Interface Builder.

- Replace default code between {} of the @interface with

```
IBOutlet UITextField *theField;
IBOutlet UILabel *theLabel;
```

For the Keyboard to disappear when done, we need for it to relinquish responding to the cursor being placed in the text field after Return is pressed .

- Modify **FlipsideViewController.m**

Add the following methods between @implementation and @end keywords

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

// Click Outside the Text Box to make KeyBoard Vanish

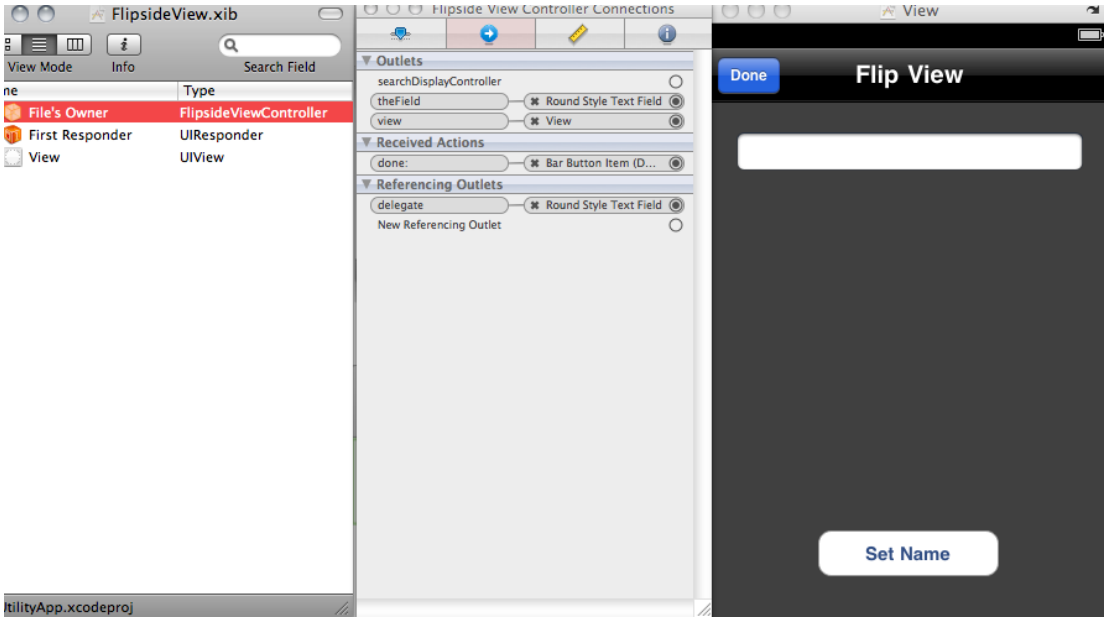
```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
[theField resignFirstResponder]; [super touchesBegan:touches
withEvent:event];
}
```

Associate GUI to ViewController Class

- Switch to the Interface Builder tool
- Select "**File's Owner**" icon in the **FlipsideView.xib** window.
- Bring up the Controller Connections window from "**Tools > Connections Inspector**".

You should see an entry for **theField** Outlet created earlier.

- Drag a line from theField to the Text Field widget on the Flip view.
- Next Select the Text Field
- Connect "Delegate" in Controller Connections to the "File's Owner"



Save the Interface Builder changes with "Ctrl-S". Switch to Xcode > Build and Run.

Now pressing return after text entry or clicking outside the text box will hide the Keyboard.

Add Behavior to Set Name Button

We want to alter the Label on the Main View when the button on Flip View is Pressed. Start by adding Outlet attribute for the Label to the **`FlipsideViewController.h`**

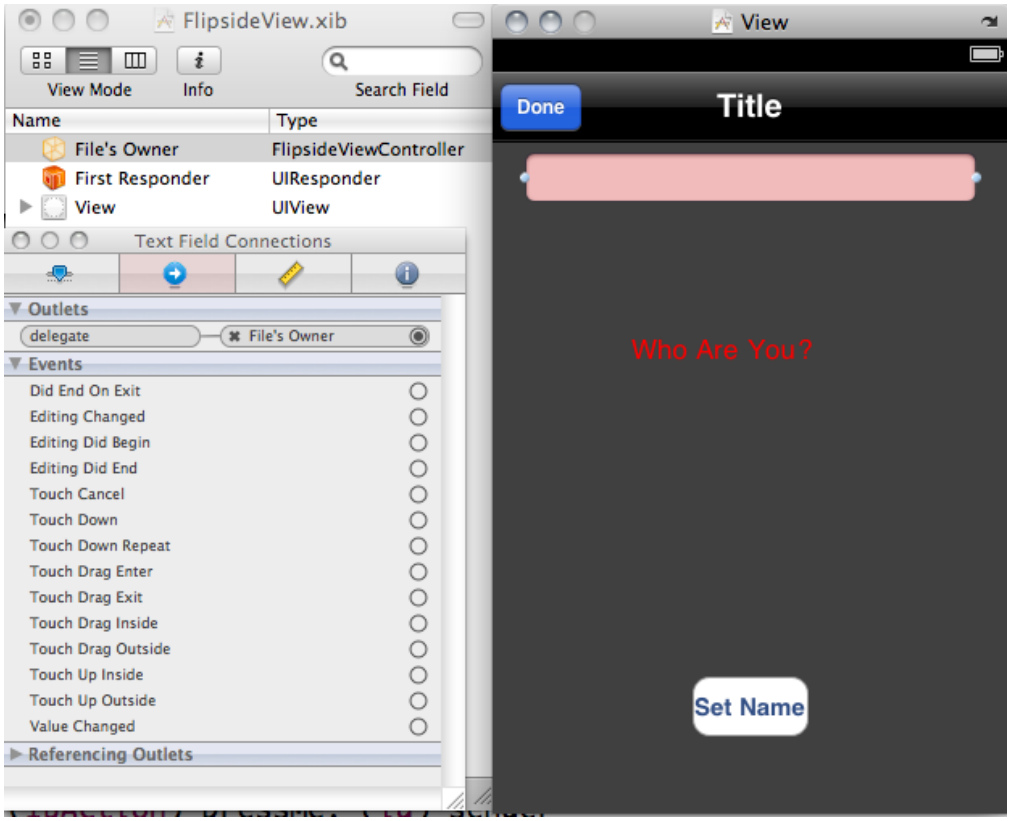
```
IBOutlet UILabel *theLabel;
```

Provide method signature for code to handle the Button event in **`FlipsideViewController.h`**

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

Switch to Interface Builder to connect these attributes and Actions to GUI widgets. (Remember to Save/Build Xcode).

- Connect the Label as before (**File's Owner>theLabel Outlet>Label on Flip View**)



To the Button we need to associate an Action.

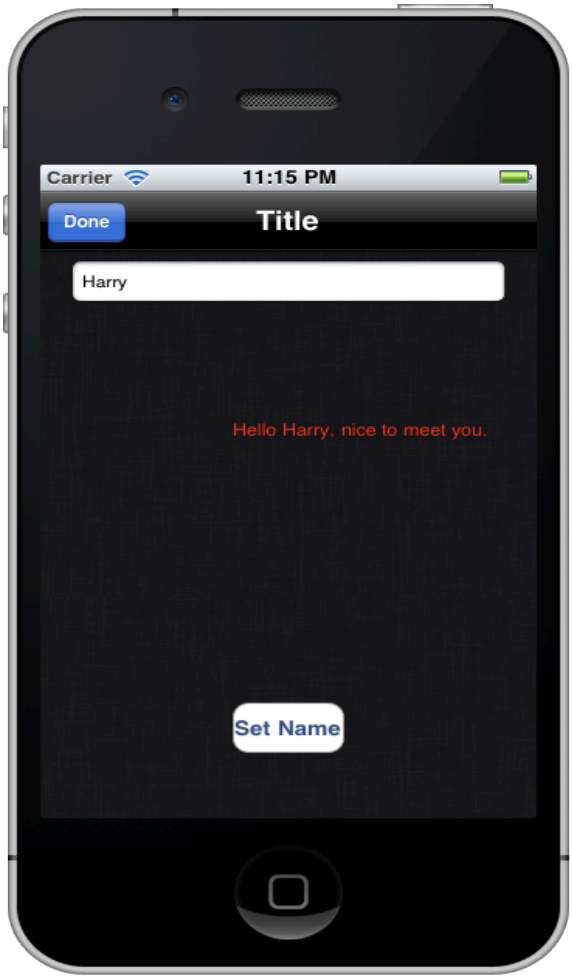
- Select the button, find the Event Action "**Touch Up Inside**"
- Connect to "File's Owner". A PopUp is displayed with "**PressMe**" – Select it.

Add Implementation code for "PressMe" method. Build and Run from Xcode

```
- (IBAction) PressMe:(id)sender

{
     NSString *txt = [[NSString alloc] initWithFormat: @"Hello
%@, nice to meet you.", [theField text]];
     [theLabel setText:txt];
     NSLog(@"The button was clicked");}
```

The Final Output should look like the screenshot below:-



Exercise

- Add a RESET Button on Flip View to restore the Original Label Text
- Modify the Label on Pressing of the Return Key, in addition to pressing the “Set Name” button

APPENDIX

Objective C (A Short Guide)

Creating Objects

```
NSString* myString = [[NSString alloc] init];
```

The *alloc* method is called on NSString itself. The call to *init* is on the newly created object. Objects created in this manner must be released. `[mystring release];`

Calling Methods

```
[object method];
[object methodWithInput:input];
```

Code within square brackets means you are sending a Message to an Object or Class.

All Objective-C variables are Pointer types.

```
NSString* myString = [NSString string];
```

Nested Messages

If result of function2 is to be passed as input to function1

```
[NSString stringWithFormat:[prefs format]];
```

The Class Interface

Is stored in the .h file, and defines instance variables and public methods.

```
@interface Shape : NSObject {
      //instance variables
      float x;
      float y;
}
      //Public Methods
- (void) findArea;//Instance Level

@property float x;
…
```

Properties

Properties are a feature in Objective-C that allow to automatically generate getters and setters. This should be specified in the Interface.

```
@property (retain) NSString* name;
```

The Implementation should contain the @synthesize directive

```
@synthesize name;
```