MAD2 Diploma in IT Year 2 (2018) Semester 4	Week 3 **3 hours**
Practical 3: iOS User Interface I	

Objectives

At the end of this practical exercise, the students should be able to:

- Design an iOS user interface using Interface Builder
- Write and compile an iOS app with a simple user interface

IMPORTANT

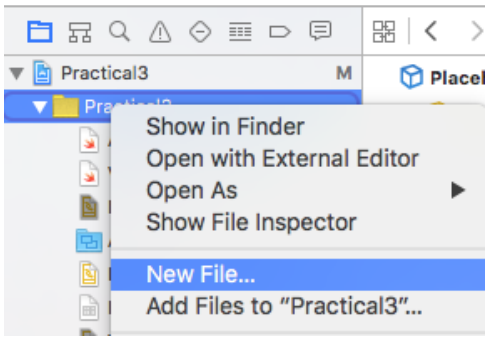
- At the end of the session, copy this folder (all your work) to MAD2 network folder so that your tutor may assess your work.
- The path of MAD2 network folder is **`\\ictspace.ict.np.edu.sg\MAD2`**

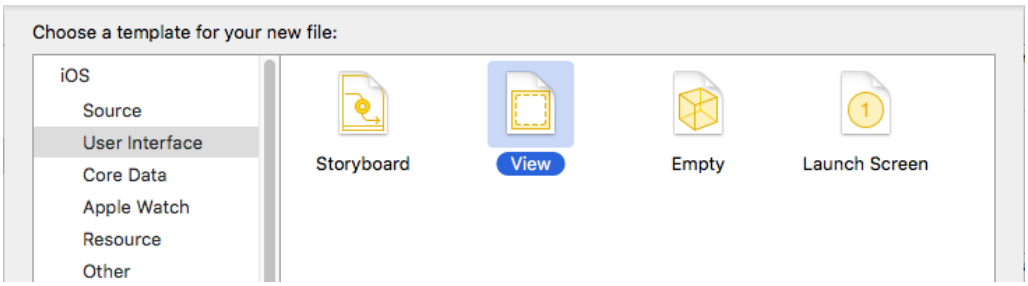
Example 1 – Creating the UI with Interface Builder

1. Launch Xcode and create a new project "Practical3"

2. Be familiar with the various Xcode interfaces.

3. Create a new User Interface and name it ViewController



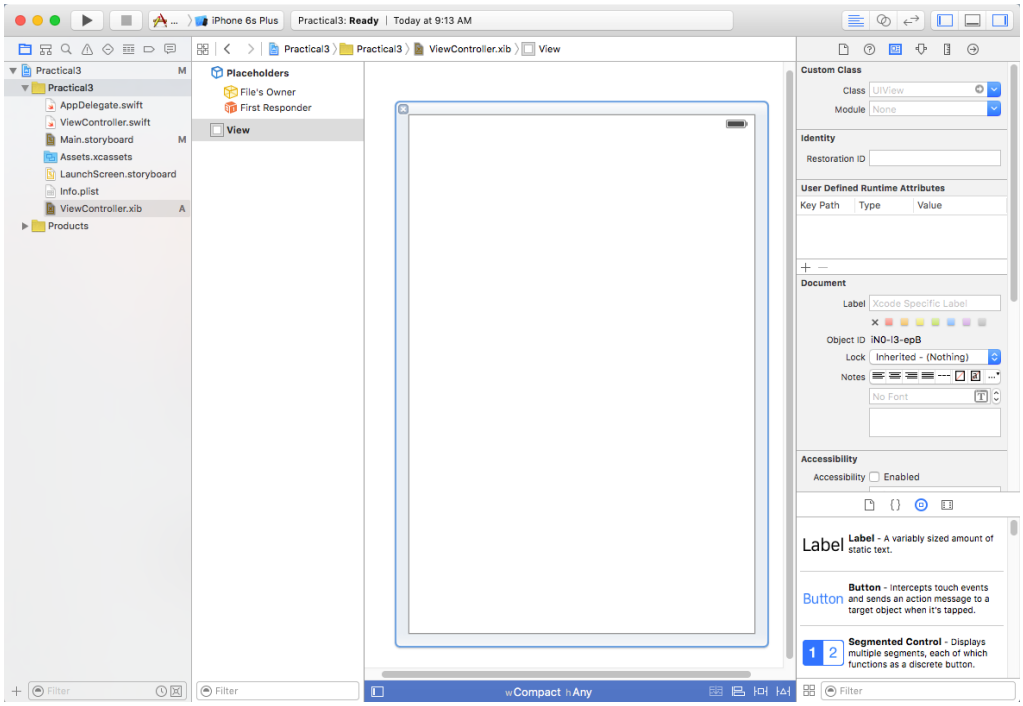


User interfaces in Xcode are designed using the Interface Builder. Interface Builder is an Xcode editor that provides a graphical interface for the creation of user interface files called a

nib file. Such a nib file has a filename with the extension `nib` or `xib`. There is another kind of UI resource in iOS known as Storyboard, which will be covered next week.

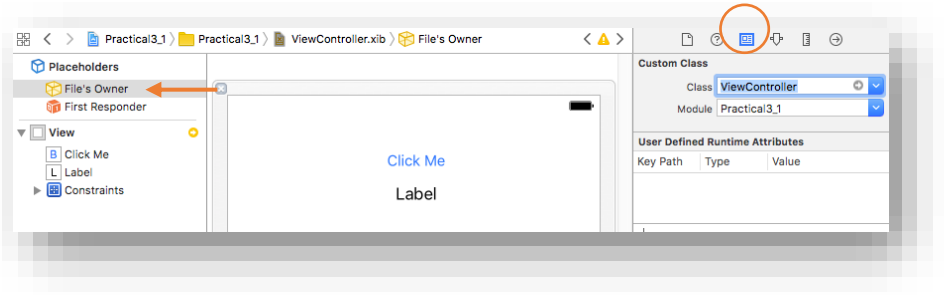
You will now create a `UIButton` and a `UILabel` and place them on the ViewController's `UIView`. Refer to the lecture notes for the definition of these terms.

Thereafter, you will create and define the handler to perform a specific task when the button is clicked by the user.

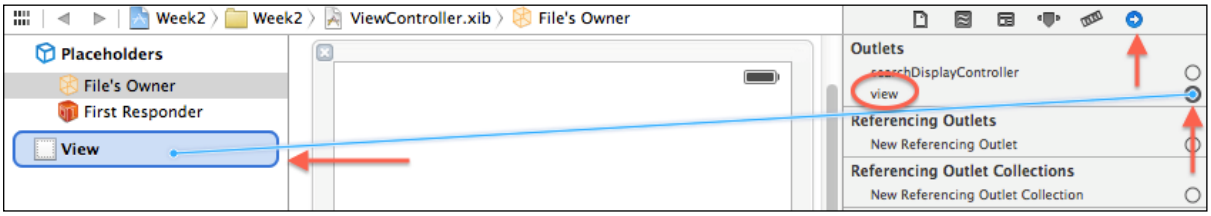


4. With the ViewController.xib selected, expand the Utilities panel at the top-right corner:

5. We now need to associate this `xib` file with a controller file. Under Placeholders, click File's Owner. Then choose the Identity Inspector on the right panel. From the Custom Class dropdown box, scroll to select ViewController (which matches ViewController.xib file you just created).

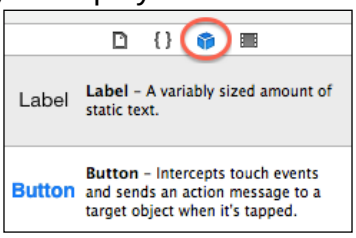


6. Next, assign the view property using the Connections Inspector. Click-drag from the Outlets (right-side) to the left-side:

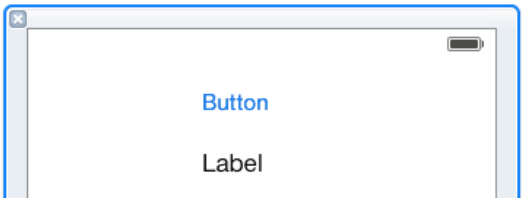


When the mouse is released, your screen should look like:

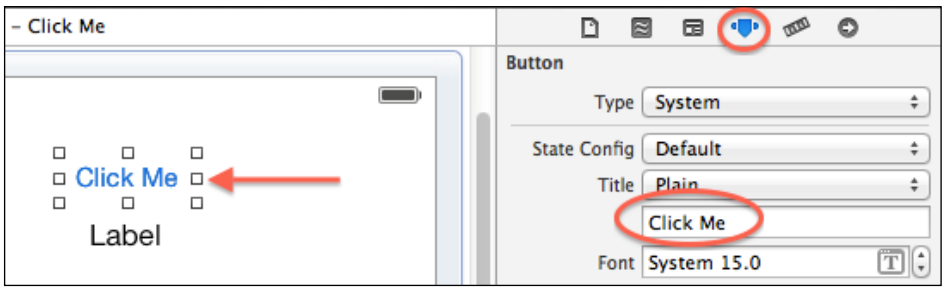
7. Next, click the Object library to display a list of user interface controls:



8. Drag a Button and a Label to the ViewController's view:



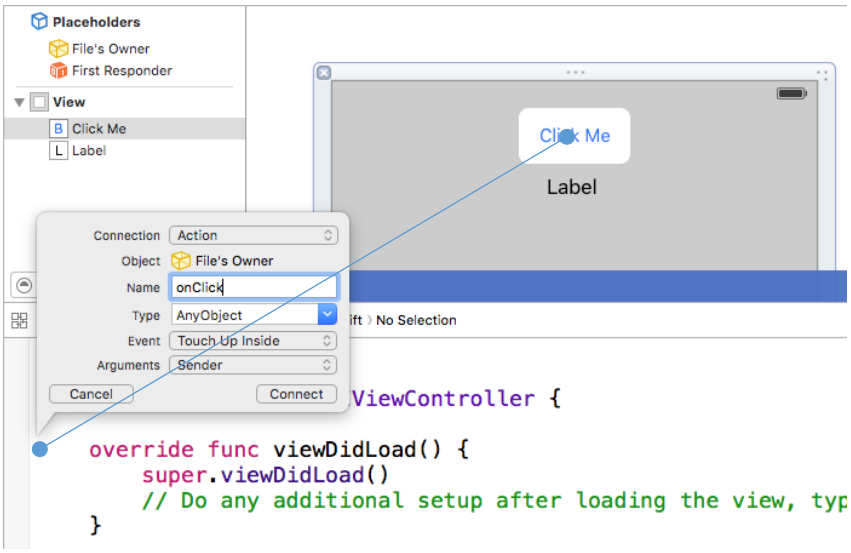
9. Select the currently untitled button. Click the Attributes inspector and change its title to "Click Me":



You may need to adjust the size of the button by dragging the handles surrounding the button so that the whole title is visible.

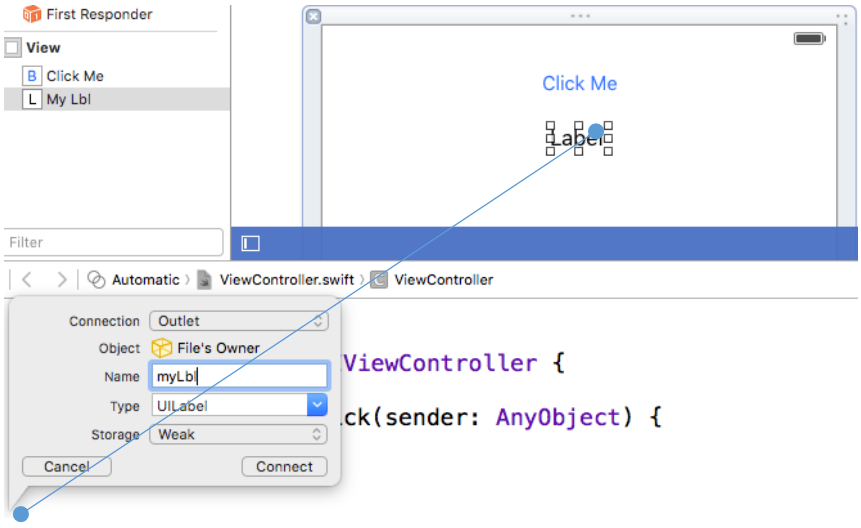
You will now define a `UIButton` instance variable and a `UILabel` instance variable to hold these controls.

10. Click-drag from the Button control to the viewController code to look like the following:



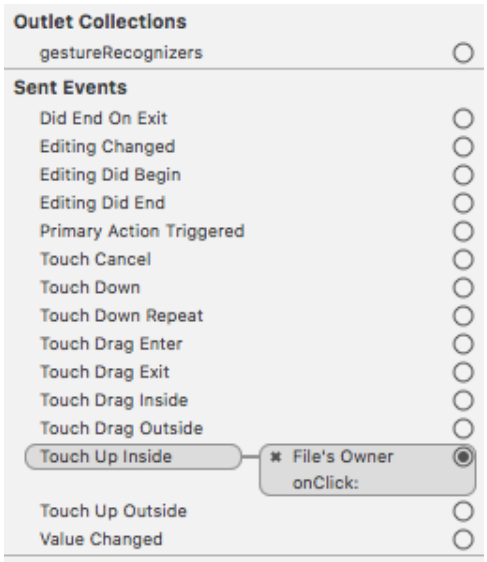
add a code `print("I'm Clicked!")` into the onClick button function.

11. Repeat the previous step for the Label control.



12. Return to ViewController.xib. Select the button and then the Connections inspector:

13. Check that all outlets are “wired” properly:



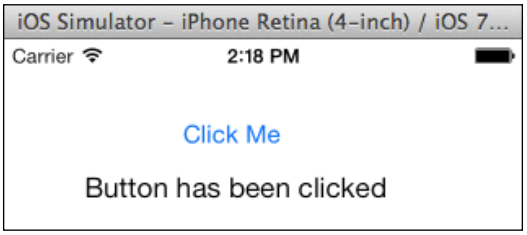
14. Add the following to AppDelegate.swift

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        self.window = UIWindow(frame: UIScreen.main.bounds)
        self.window?.makeKeyAndVisible()
        let viewController = ViewController(nibName: "ViewController", bundle: nil)

        self.window?.rootViewController = viewController
        self.window?.makeKeyAndVisible()

        return true
    }
```

15. Click Run and observe the output in the Debugging window when the simulator is launched.

Exercise 1 Hook up the `UILabel` control to the `myLbl` instance variable.

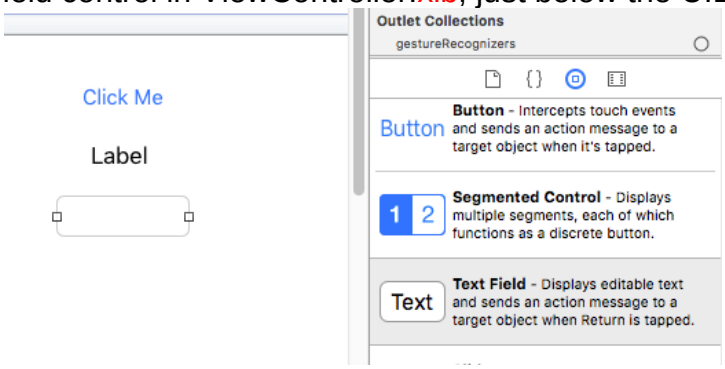
Exercise 2 Besides displaying the message “Button has been clicked” through `print`, display the same message using the `UILabel` created in Exercise 1. Your output might look something like this on the iOS Simulator:



Example 2 – Adding UI Delegate

1. Continue with Exercise 2.

2. Add a UITextField control in ViewController.xib, just below the UILabel.



3. Wire up the text Field control as a @IBOutlet called txtField.

4. Add the following function, a UITextField delegate, anywhere in ViewController.swift:

```
@IBOutlet weak var txtField: UITextField!

func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    txtField.resignFirstResponder()
    MyLbl.text = textField.text
    return true
}
```

Note: make sure the class inherit the UITextFieldDelegate class

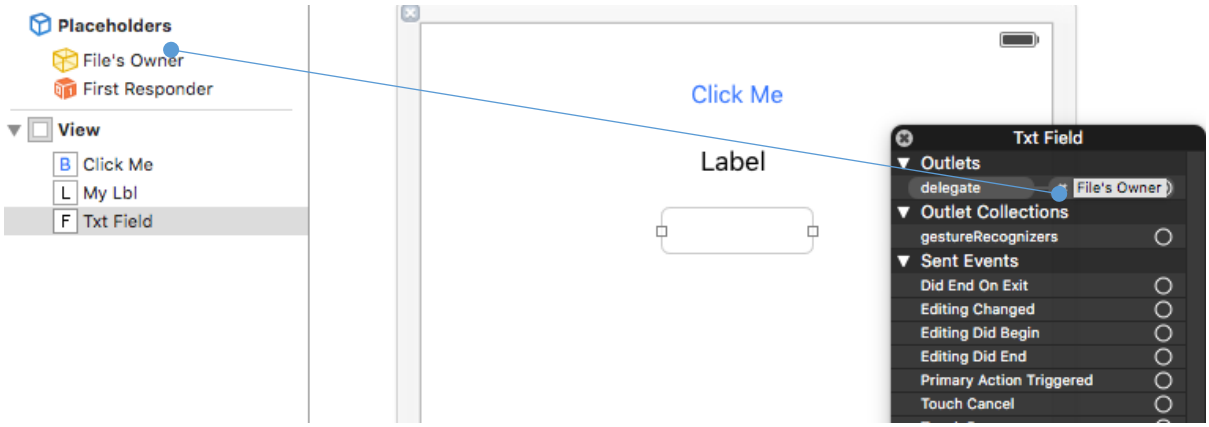
i.e `class ViewController: UIViewController,UITextFieldDelegate {`

5. Return to ViewController.xib, select the UITextField control.

6. The next step is to specify the location of the UITextField delegate method. Although it has been defined in ViewController.swift, you need to hook it up to the UITextField control. You can do this in two ways:

(a) Interface Builder

Click on “delegate” and drag to File’s Owner as shown below:



(b) programmatically

To use this method, you need to first remove the delegate assigned through Interface Builder, by clicking its 'x' icon. Then, add the following statement at the end of `viewDidLoad` method in ViewController.swift

```
txtField.delegate = self // the delegate is define in this class
```

7. Click Run. Type something in the UITextField and press Return.