Intro

We’re going to wire up our activity view which is our first viewwhere we go ahead and select activities and we click our start andour stop to log some time against some activity.So we’re going to make it interactive now and we’re going to get into IB outlets andIB actions.

IB Outlet

An **IB outlet** is an interface builder outlet. The outlet means you are connecting some UI element to the back end. In the code you’ve got a reference to that UI element and you can do different things with it. What we are going to do is create IB outlets for our picker view, for our label with zero on it and for our start button.

IB action

So let’s create some outlets for the label and for our button. For now you can hide the document outline and the debugger window.

Highlight zero on the label, click the `Ctrl` and then drag above viewDidLoad() - you will get an outlet. Give it a name of `timerLabel`.

Connecting the App

Now, if you go to the zero label, click on it and then on the Connections Inspector, you will see we’ve got a referencing outlet here. So now we will create one for the picker view by holding `Ctrl` and dragging right under our timer label. Call it `pickerViewConnect`.

Creating Button Actions

Now work on your button now and create an action. Once again use `Ctrl` and drag to have an outlet selected. Choose an action and call it `startButton`.

Button Event

The event should be called “Touch Up Inside”, which is really just a simple click event. The Argument is Sender with a type of “any object”.

For the button we want to check if the text on it is “start”. If it it, we’ll change it on a click to “stop”. Because we know what the state of the button is in, we can make these text changes. Therefore we will need another IB Outlet for the button. Call it `startButton`.

Creating Stop

Now let’s code the conditional statement:

```
@IBAction func startButton_Click(sender: AnyObject) {
    if(startButton.titleLabel!.text == "Start"){
        startButton.setTitle("Stop", forState: .Normal)
    }
    else {
        startButton.setTitle("Start", forState: .Normal)
    }
}
```

Build the app and check if it is working correctly.

Creating Variables

Now let’s create a few variables. We’re going to have a timer, which will increment when we click start. We will also need a timer counter that starts at zero. The last variable is the `currentActivity` that we’re dealing with.

```
var timer = NSTimer()
var timerCounter = 0
var currentActivity:Activity?
```

Calling a Function

Now code the actual function:

```
@IBAction func startButton_Click(sender: AnyObject) {
    if(startButton.titleLabel!.text == "Start"){
        timerCounter = 0
        timer = NSTimer.scheduledTimerWithTimeInterval(1, target: self, selector: Selector("proces
        startButton.setTitle("Stop", forState: .Normal)
    }
    else {
        startButton.setTitle("Start", forState: .Normal)
        timer.invalidate()
        currentActivity?.totalTime += timerCounter
    }
}
```

Adding Activities into Picker View

Now what we need is to put some activities into our picker view. Bring up the project navigator and click on the story board. We need to grab some delegates for our picker view and also create a data source, and to do that we're going to add some protocols to our class. This is similar to what we did with our table view which is going to be in our performance tab.

```
func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
    currentActivity = ActivityManager.activities[row]
}

func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> St
    var activity = ActivityManager.activities[row] as Activity
    return activity.activityName
}
```

Also code the data source:

```
func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    return ActivityManager.activities.count
}

func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
    return 1
}
```