UCI Extension

I&C SCI X402.37
Intro to Mobile Development for Apple iPhone and iPad
2015 Lesson Plans

Week 4: Phone features and Storyboards

- **Topics covered:**
    - Camera (BNR Ch. 11)
    - Touch Events (BNR Ch. 12)
    - UIGestureRecognizer (BNR Ch. 13)
    - Storyboards (BNR Ch. 28)
- **In-class:**
    - I will review the code behind the camera in the sample Homepwner application from Week 2.
    - I will go over how I converted the Homepwner sample application I have been showing in class to use storyboards and the review the different techniques we can use to reduce the amount of code by using UIStoryboard and UIStoryboardSegue.
        - **Implementation notes:**
            - Will need to convert data passes to use segues
            - Need to come up with a way to show UITouch, UIGestureRecognizer, etc.
                - Touches – Maybe a little game tutorial?
                - GRs - maybe swiping can display a CRUD menu?

Homework:

- **Reading:**
- **Coding:** Students will convert their basic SwiftConvrxns project to run on storyboards. Also, we will use a UIGestureRecognizer to allow users to adjust the inputValue for the conversion by “panning” and switching between “type-to-enter” mode and “pan-to-adjust” with a double tap.

Implementation Instructions:

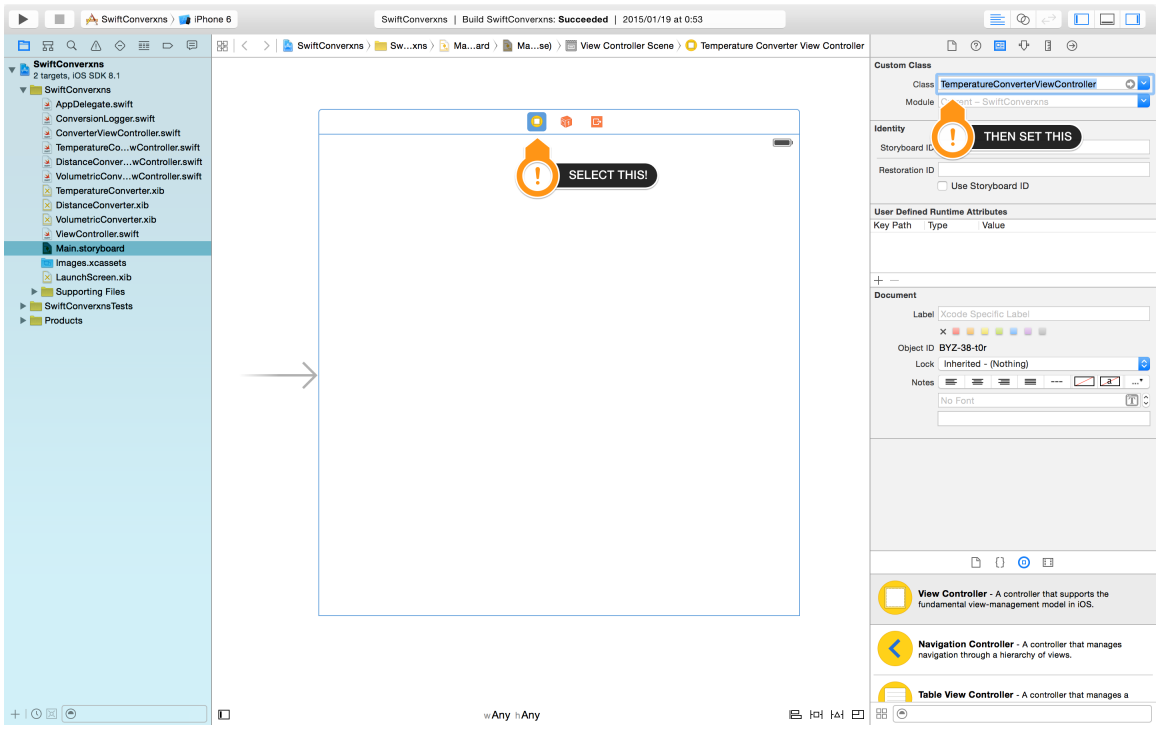
1. **Project Setup**
   a. Create a copy of your project from Week 3 and rename the project folder in your file system. (You do not have to rename the actual project.)
   b. Open the new project in Xcode.
2. **Create your storyboard.**
   a. You should have a file in your project called Main.storyboard. If you do not, create a new file and select 'Storyboard' from the 'iOS' > 'User Interface' section in the wizard that appears and name this file Main.storyboard.
   b. Click on the project and select the 'General' tab. Make sure the 'Main Interface' property says 'Main' (this is the name of the storyboard which will serve as the main interface of the app – Main.storyboard). Next, in AppDelegate.swift, remove the code from the application:didFinishLaunchingWithOptions: function that sets up the view hierarchy but leave the code that sets up the NSUserDefaults. When you are done with this step, your function should appear as follows.

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

        if let data: NSData =
NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as? NSData {

        }
        else
        {
            let defaultFavorites = NSKeyedArchiver.archivedDataWithRootObject([])

            NSUserDefaults.standardUserDefaults().registerDefaults(["FavoriteConversions" :
defaultFavorites])
            NSUserDefaults.standardUserDefaults().synchronize()

        }


        return true
    }
```

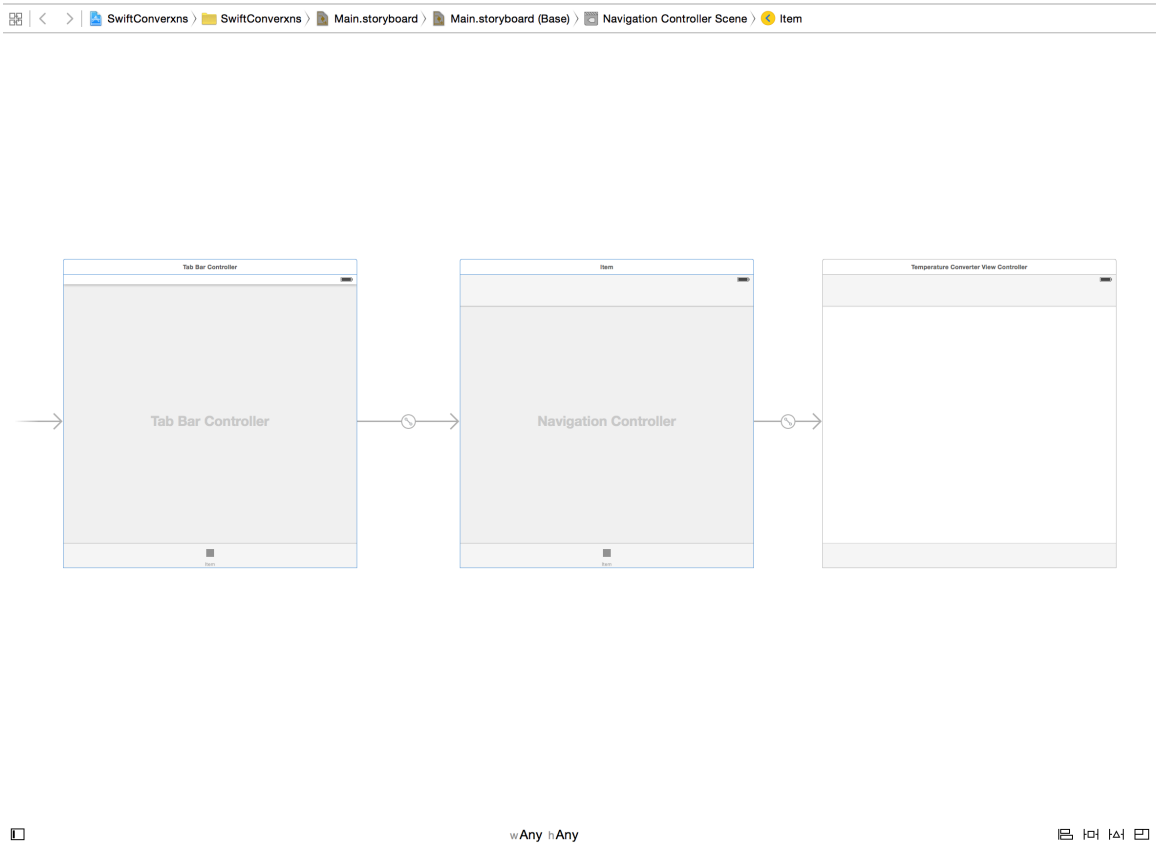
Now, the application will use the storyboard to provide the views of the ViewControllers.

c. Open the storyboard in Interface Builder (just click the Main.storyboard file once; do not double click. This will open the file in the main window in Xcode and allow you to access the inspector panels. If you double click, it will open in a separate window and you will not be able to access the inspectors).

You should have one empty view controller in your storyboard. Provide TemperatureConverterViewController as the custom class of this view controller. (You do this in a similar way to how you provide the custom class for the File's Owner in a XIB).



d. Now that we have set the custom class, we can create the UI. But before we do that, let's finish setting up the UI flows using the storyboard. With the TemperatureConverterViewController selected, select 'Editor' > 'Embed In' > 'Navigation Controller' to place our converter view controller inside a navigation controller. Click that navigation controller and repeat the processs, but this time, embed it in a tab bar controller as was shown in class. When you are done your storyboard should appear as follows.



e. Now our application should look familiar—we have a converter view controller inside a navigation controller and that is in a tab bar controller. But our view is blank – if we ran our application now, we would see that

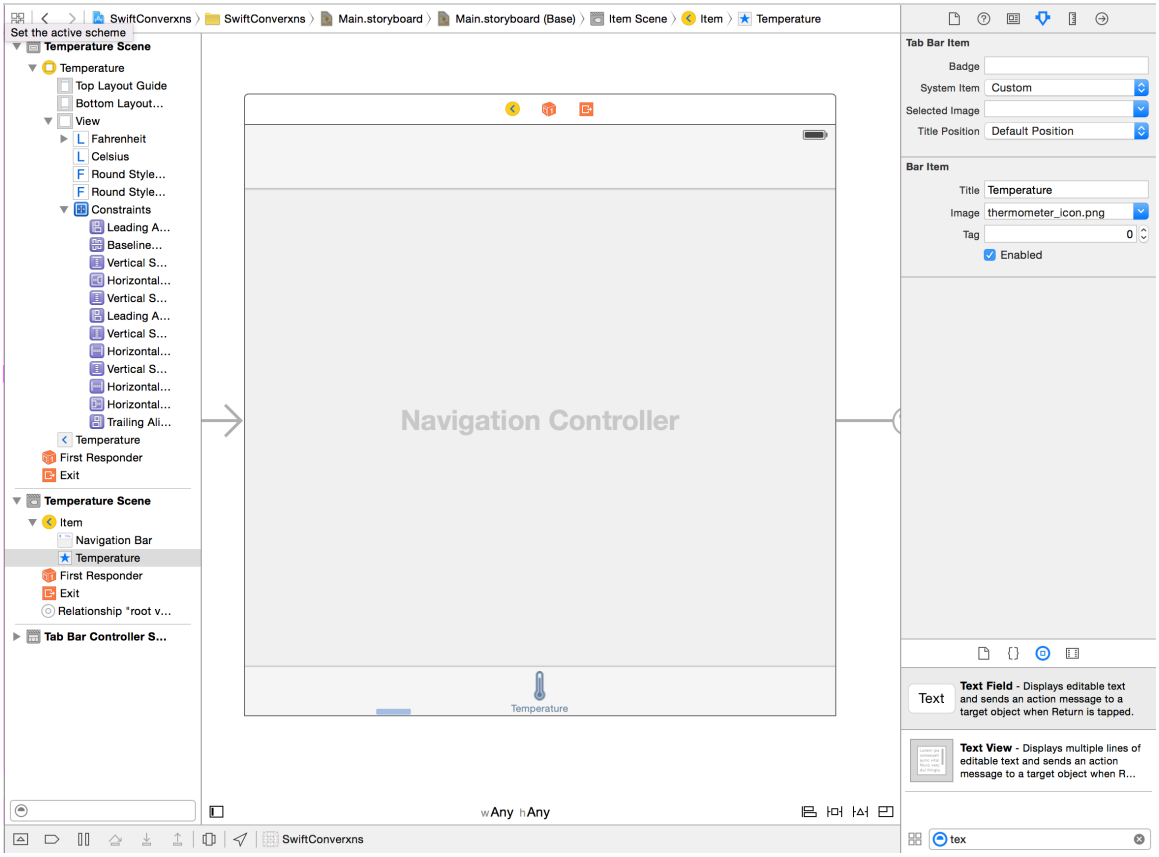
tab bar and an empty, white view on screen. Put in the UILabels and UITextFields needed to make your temperature converter's UI look as expected. (This is exactly the same as when we used a XIB – drag and drop the subviews onto the canvas and set the IBOutlets. Don't forget to set your textFields' delegate outlet!)

f. Run the application. If you have done everything correctly, you should see your temperature converter inside the tab bar with the textFields and labels as expected.

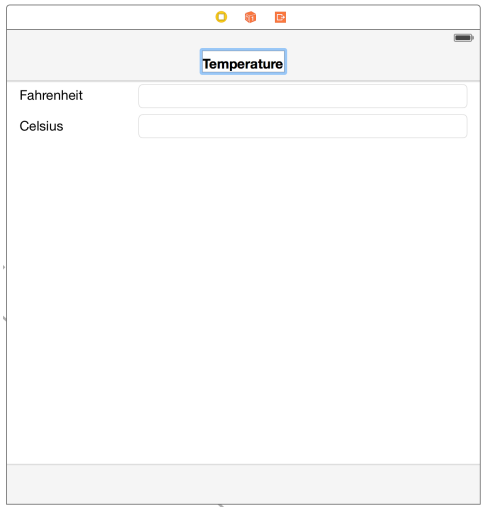
g. Let's add some polish to the application.

i. In your storyboard, locate the Navigation controller we added in Step 2d. Since it is embedded in a tab bar controller, it has a tab bar item button. Click on this, then in the 'Attributes Inspector', set the 'Title'

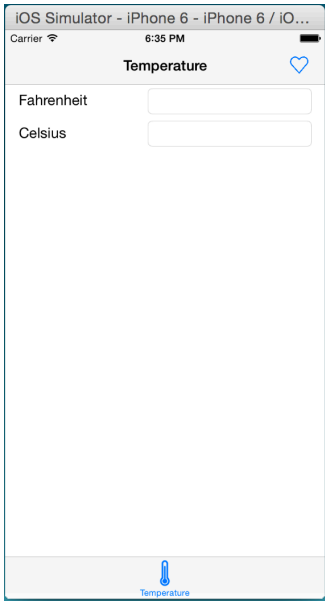
property to “Temperature” and set the ‘Image’ property to “thermometer_icon.png”. (Make sure to download the image from the class homepage and add it to your project first.)



ii. Find the TemperatureConverterViewController on your storyboard. There should be a simulated navigation bar. Double-click in the middle of it and set the title of the view controller to “Temperature” as well.

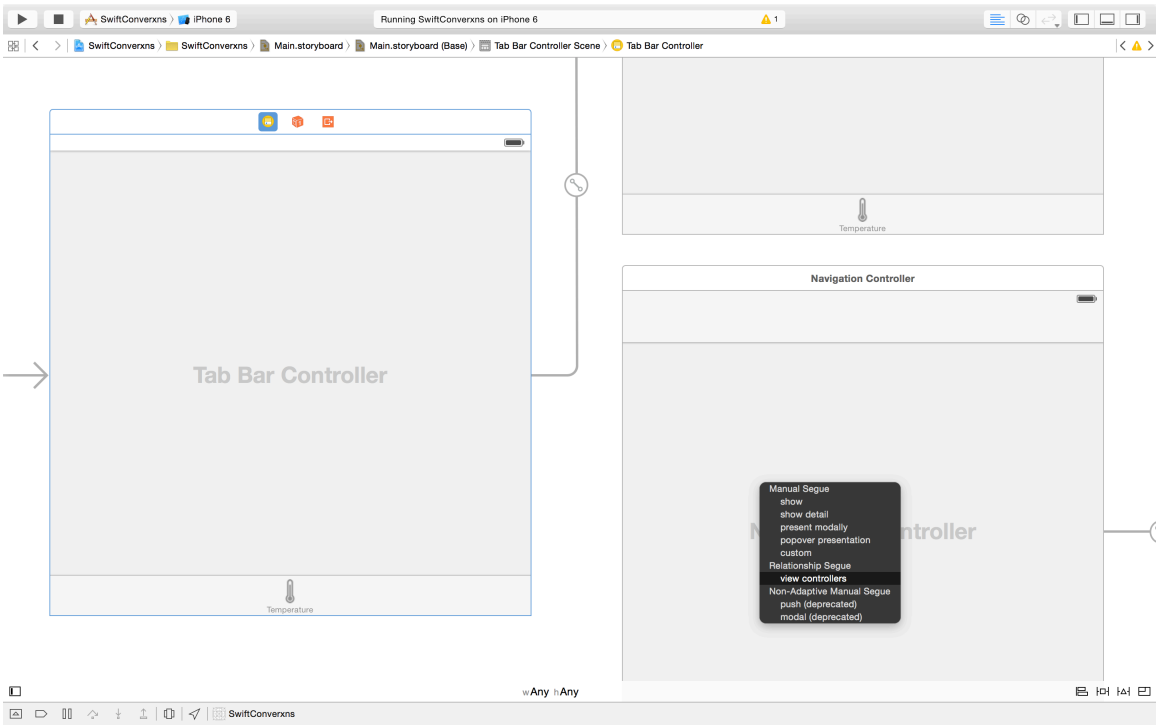


Now when you run the application, your tab bar and navigation bar should look a little more descriptive.

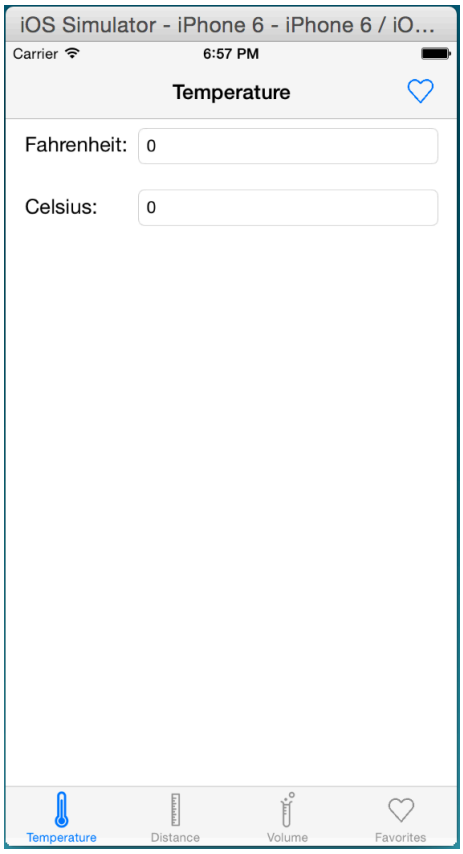


h. Practice creating your UI flow by dragging more view controllers onto the storyboard canvas so that you have all three converter controllers and the Favorites table view controller on your storyboard with the appropriate UI. There are icon images for all three converters on the course homepage. (You should have a heart icon in your project from Week 3. Use this for the icon for that favorites tab as well.)

You can embed your converters in navigation controllers as in Step 2d, but when it comes time to place those navigation controllers into your tab bar controller, we have to do things a littlee differently. Using the 'Editor' menu to embed the new navigation controllers would result in multiple tab bar controllers on our storyboard. To embed them in our existing tab bar controller, Control-drag from the tab bar controller to the navigation controller and select the 'view controllers' relationship segue.

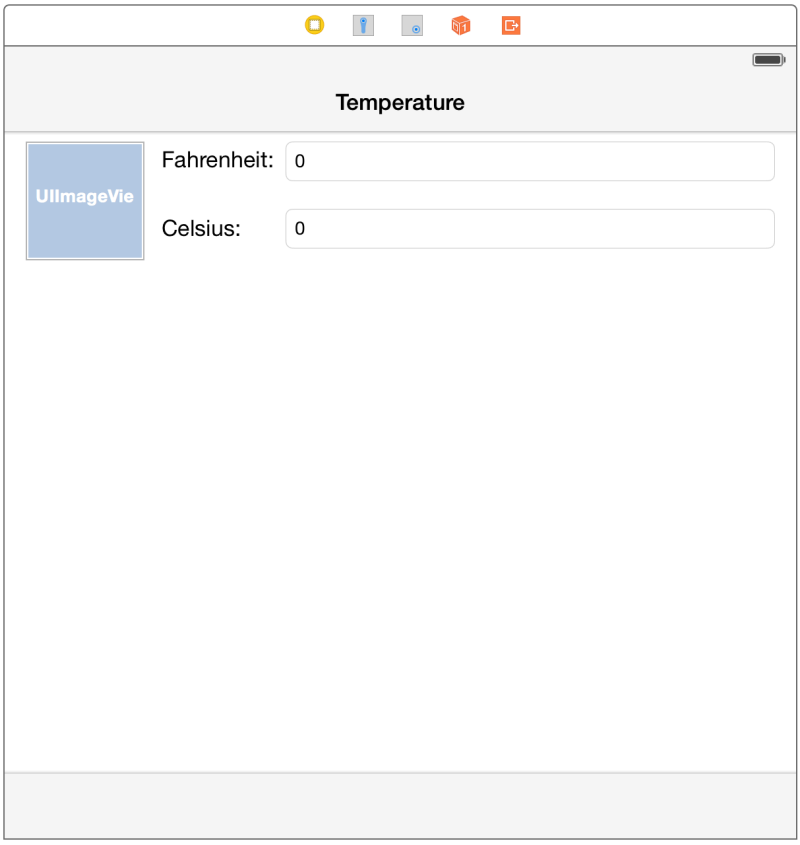


When you are finished, you should have all three converters and the favorites view controller on your storyboard and they should display in four tabs when you run the application. We are able to do this because we are reusing our view controller classes and we are just using the storyboard to provide the views.



3. **Enhance the model.** Let's get some practice with UIImagePicker, UIImageView and UIImage by allowing users to take a photo or selecting an image from their device's photo library. To do this, we need to add a little more to our UI and also to our Favorite model class. Finally, we need to modify the functions we created to save our favorite conversions.

a. Let's just focus on the temperature converter for the rest of this assignment. (Later, you can go and repeat these steps for your other converters). In the Main.storyboard, drag a UIImageView onto the main view of the TemperatureConverterViewController and resize your subviews so it looks as follows. You will need to clear all the constraints for all views in View and then 'Add Missing Constraints' for all views in View. (We will learn more about constraints in Week 5).

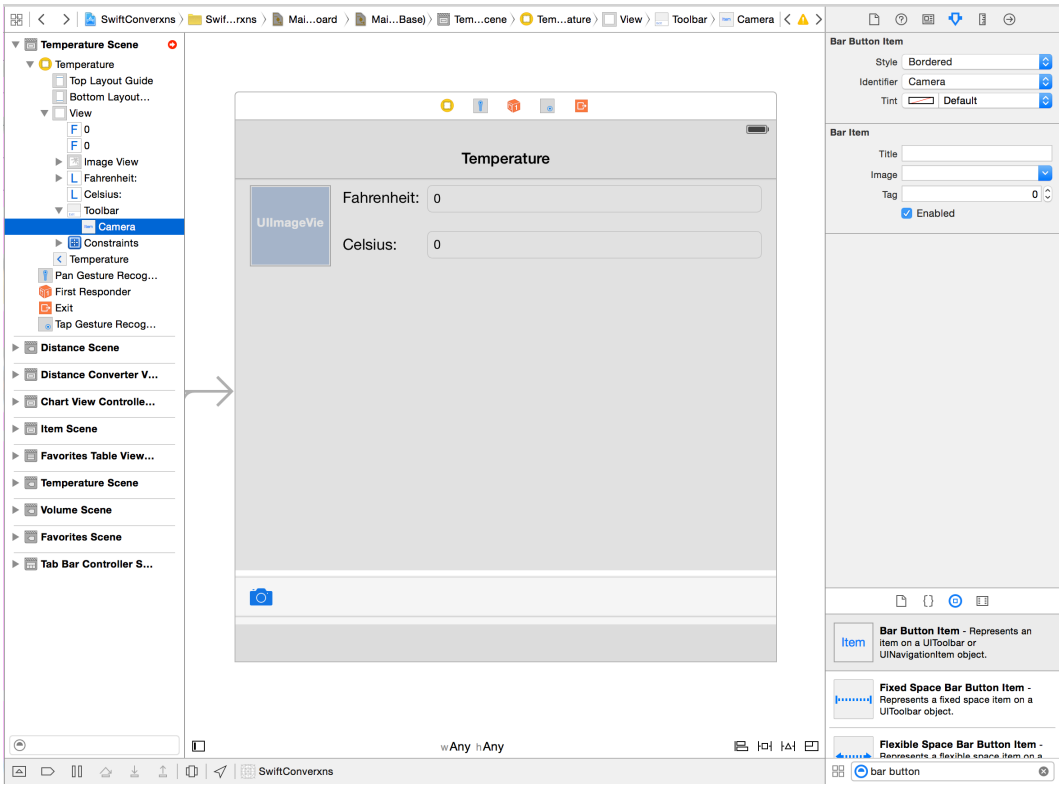


Create an outlet for this image view and name it "imageView".

b. Drag a 'Tool Bar' onto the temperature converter and position it at the bottom. Then, with the toolbar selected, select 'Editor' menu > 'Resolve

Auto Layout Issues' > 'Add Missing Constaints' under the 'Selected Views' section.

c. Next, drag a 'Bar Button Item' onto the left hand side of the tool bar (it should just say "Item" at this point). Select it and from the 'Attributes Inspector' change its 'Identifier' property to be 'Camera'. Your bar button will now have a camera icon.



d. Now, create an IBAction function for this button in TemperatureConverterViewController (using the control-drag method we are used to) and name it “cameraButtonPressed”. Then implement that method as follows.

```
@IBAction func cameraButtonPressed(sender: AnyObject) {

        let imagePicker = UIImagePickerController()

        imagePicker.delegate = self

        if (UIImagePickerController.isSourceTypeAvailable(.Camera))
        {
            imagePicker.sourceType = UIImagePickerControllerSourceType.SavedPhotosAlbum
        }
        else
        {
            imagePicker.sourceType = .SavedPhotosAlbum
        }

        presentViewController(imagePicker, animated: true, completion: nil)

    }
```

Notice how we check to see if the camera is available and set the imagePicker’s source type accordingly. Then we display the imagePicker by calling the presentViewController() function.

Also notice we did not call `self.presentViewController()`. We can call this because self, in this case, is a (“*isa*”) UIViewController and presentViewController() is an instance method of UIViewController.

e. We have an error because self doesn’t conform to the UIImagePickerDelegate protocol. If we ran the application at this point, pressing the camera button will display the image picker (camera or photo album), but nothing will happen when we select and image or shoot a photo. To handle that, we need to utilize the delegate functions in the UIImagePickerDelegate protocol.

We set the imagePicker.delegate in the previous step, now implement

the following functions inside of TemperatureConverterViewController.swift.

```
func imagePickerController(picker: UIImagePickerController!, didFinishPickingImage image: UIImage!,
editingInfo: [NSObject : AnyObject]!) {

        imageView.image = image
        picker.dismissViewControllerAnimated(true, completion: nil)

}

func imagePickerController(picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[NSObject : AnyObject]) {

        imageView.image = info["UIImagePickerControllerOriginalImage"] as? UIImage
        picker.dismissViewControllerAnimated(true, completion: nil)
}

func imagePickerControllerDidCancel(picker: UIImagePickerController) {

      picker.dismissViewControllerAnimated(true, completion: nil)

}
```

We created our imagePicker in Interface Builder and created an outlet for it. Now, in these callbacks we set imageView.image and dismiss the imagePicker once the user selects an image or takes a photo. If they don't want to do either of those, tapping the 'Cancel' button provided by the SDK will dismiss the imagePicker as well. Setting imageView.image takes care of actually adding the image to the image view on screen.

f. Now that we've provided the UI for selecting an image, let's work on enhancing our model class, "Favorite" and to hold the image for us. Remember, we are using NSCoding to save our Favorites into NSUserDefaults. This only supports saving plist objects (basically only NSObject or its subclasses). Luckily, we can convert the UIImage into NSData to accomplish this. (Not the most efficient for storage space, but it will serve our purposes without making things too complicated for this class). We will also give users a change to name their favorite later on.

Modify Favorites.swift as follows on the next page.

```
class Favorite: NSObject, NSCoding {

    var conversionType: String

    var inputValue: Double

    var inputUnits: String

    var outputValue: Double

    var outputUnits: String

     var imageData: NSData?

     var favoriteTitle: String?

    required init(conversionType: String, inputValue: Double, inputUnits: String, outputValue:
Double, outputUnits: String) {

        self.conversionType = conversionType

        self.inputValue = inputValue

        self.inputUnits = inputUnits

        self.outputValue = outputValue

        self.outputUnits = outputUnits

        super.init()

    }

    required init(coder aDecoder: NSCoder) {

        self.conversionType = aDecoder.decodeObjectForKey("conversionType") as String
        self.inputValue = aDecoder.decodeDoubleForKey("inputValue")
        self.inputUnits = aDecoder.decodeObjectForKey("inputUnits") as String
        self.outputValue = aDecoder.decodeDoubleForKey("outputValue")
        self.outputUnits = aDecoder.decodeObjectForKey("outputUnits") as String
         self.favoriteTitle = aDecoder.decodeObjectForKey("favoriteTitle") as? String
         self.imageData = aDecoder.decodeObjectForKey("imageData") as? NSData

    }

    func encodeWithCoder(aCoder: NSCoder) {

        aCoder.encodeObject(self.conversionType, forKey: "conversionType")
        aCoder.encodeDouble(self.inputValue, forKey: "inputValue")
        aCoder.encodeObject(self.inputUnits, forKey: "inputUnits")
        aCoder.encodeDouble(self.outputValue, forKey: "outputValue")
        aCoder.encodeObject(self.outputUnits, forKey: "outputUnits")
         aCoder.encodeObject(self.favoriteTitle?, forKey: "favoriteTitle")
         aCoder.encodeObject(self.imageData?, forKey: "imageData")

    }

}
```

g. The model can now support saving images for each of our Favorites. We need to modify the logic that saves our Favorites to actually store the image. In TemperatureConverterViewController.swift, modify favoriteButtonPressed() as follows.

```
@IBAction func favoriteButtonPressed(sender: AnyObject) {

        var alert = UIAlertController(title: "Name this favorite?", message: "Enter a name to help remember
this conversion.", preferredStyle: UIAlertControllerStyle.Alert)

var nameTextField = UITextField()

        alert.addTextFieldWithConfigurationHandler { (nameTextField) -> Void in

        nameTextField.delegate = self

        let save = UIAlertAction(title: "Save", style: UIAlertActionStyle.Default, handler: { (action) ->
Void in

                (self.conversions.last! as Favorite).favoriteTitle = nameTextField.text as String?
                self.saveFavorite()

        })

        alert.addAction(save)

        let cancel = UIAlertAction(title: "Cancel", style: UIAlertActionStyle.Cancel, handler: { (action) ->
Void in
                alert.dismissViewControllerAnimated(true, completion: nil)
        })

        alert.addAction(cancel)
      }

     self.presentViewController(alert, animated: true, completion: nil)
}
```

We moved the actual logic that saves the favorite into the new saveFavorite() function (next page) and modified favoritesButtonPressed() to allow users to enter a title for their favorite (e.g. “Boiling Point of Water”). We haven’t reviewed UIAlertController in class, but I wanted to give you some experience working with this new, iOS 8 way to present alerts as UIAlertView and UIAlertViewDelegate are deprecated in iOS 8.

h. Below favoriteButtonPressed(), add the following new method.

```
func saveFavorite()
{
      // save the associated image for the Favorite
      var image = imageView.image

      let imageData = UIImageJPEGRepresentation(image, 1.0)

      (conversions.last as Favorite).imageData = imageData

      // save the favorite instance here by getting the last favorite we prepped and store it
      if let data: NSData = NSUserDefaults.standardUserDefaults().objectForKey("FavoriteConversions") as?
NSData
      {

      var savedFavorites: Array<AnyObject> = NSKeyedUnarchiver.unarchiveObjectWithData(data) as Array

        if self.conversions.count > 0
        {
                savedFavorites.append(conversions.last!)

                let updatedFavorites = NSKeyedArchiver.archivedDataWithRootObject(savedFavorites)

                NSUserDefaults.standardUserDefaults().setValue(updatedFavorites, forKey:
"FavoriteConversions")
                NSUserDefaults.standardUserDefaults().synchronize()
        }
      }
}
```

Most of this code should look familiar to you from last week (we took most of it from favoritesButtonPressed()). When a conversion is performed, we pre-save it into an instance of our Favorite class. But we are not guaranteed that the user will have chosen an image at that point. And since we are giving the user the chance to provide a title for the favorite, we needed to delay when we actually save the Favorites to NSUserDefaults, which is why we had to refactor that code into this new method. Notice that we added logic in the first three lines of this method to update our last pre-saved Favorite with our image. We get the image from the imageView, convert it into a value of type NSData, then save that data into the imageData property of Favorite that we added in Step 3e.

i. Run the application. You should be able to select an image or shoot a photo, see it in the UI, then hit the Favorites button and see the conversion in the Favorites table. But what about the image and title? It would be nice for a user to see the image and title along with the calculation in the Favorites table. Modify the following method in FavoritesTableViewController.swift.

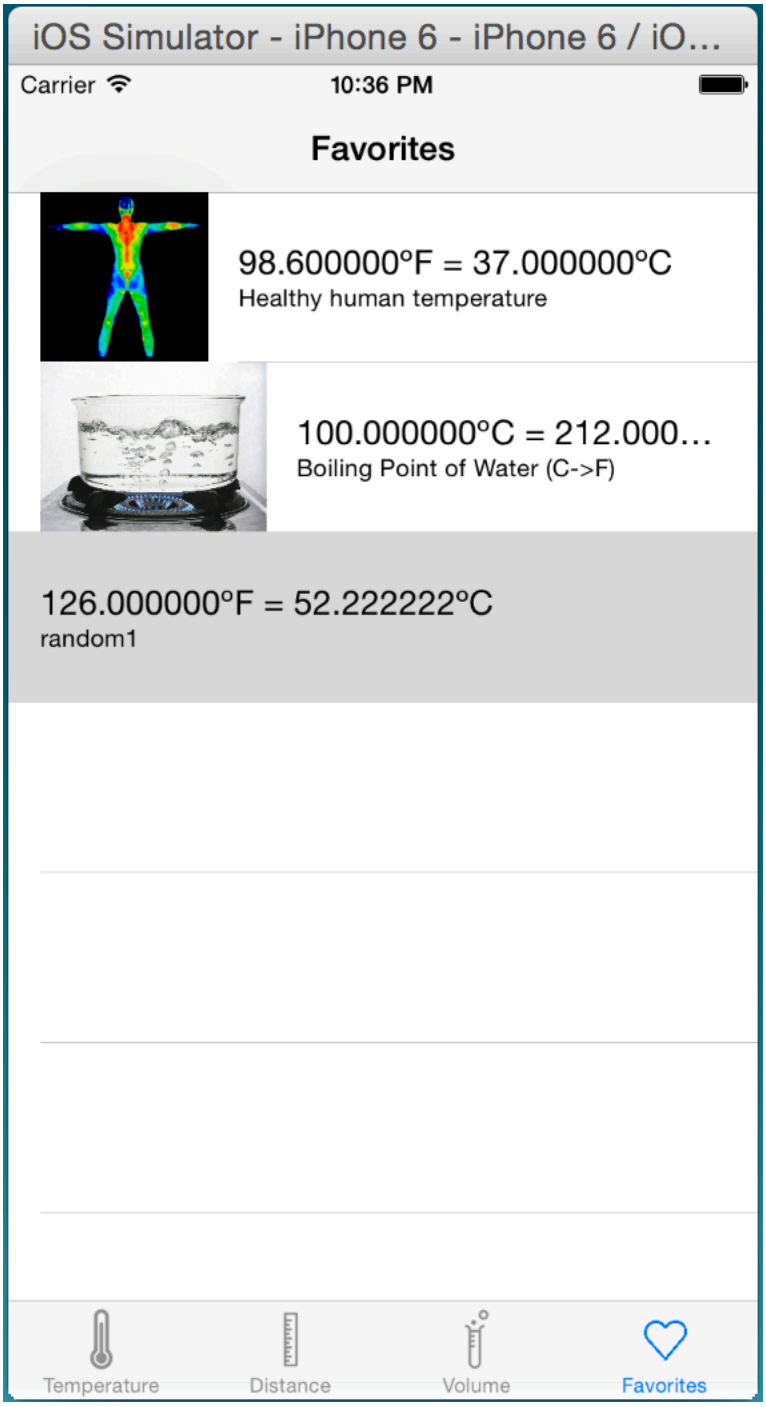
```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {

        let cell = tableView.dequeueReusableCellWithIdentifier("Cell", forIndexPath: indexPath) as
UITableViewCell

        let cell = UITableViewCell(style: UITableViewCellStyle.Subtitle, reuseIdentifier: "Cell")

        // Configure the cell...

        let favorite: Favorite = favorites[indexPath.row] as Favorite

        cell.textLabel?.text = NSString(format: "%f%@ = %f%@", favorite.inputValue, favorite.inputUnits,
favorite.outputValue, favorite.outputUnits)

         cell.detailTextLabel?.text = favorite.favoriteTitle

         if let imgData = favorite.imageData
         {
             cell.imageView?.image = UIImage(data: imgData, scale: 1.0)
         }

        return cell
    }
```

The style UITableViewStyleSubtitle gives us a cell with two UILabels (called textLabel and detailTextLabel) and an optional UIImageView called imageView. This is perfect for us, as we want to show the conversion, the title and the image view. Notice that to get the imageData back to a UIImage, we use the initWithData: method of UIImage.

Run the application. Your favorites page should appear as follows on the next page.

UCI Extension

I&C SCI X402.37
Intro to Mobile Development for Apple iPhone and iPad
2015 Lesson Plans



smilefish
Mobile and Social Media Applications

This material was written and developed by Justin-Nicholas Toyama.

4. **Enhance the UI.** Now that we can save images and a title for our favorite conversion, wouldn't it be cool if there were a faster way to input the temperature we want to convert? Let's practice using gesture recognizers to let users increase or decrease the input value by sliding (aka panning) their finger up or down on the screen as shown in class.
   a. Create a property at the top of TemperatureConverterViewController called currentValue as follows.

```
var currentValue: CGFloat?
```

   b. Let's create the extra UI item – a "handle" that will let us see what happens as we slide (pan) our finger around and also indicate to the user "slide this handle up to raise the temperature, slide down to lower the temperature".
      i. Add a dark gray view that is 100 points wide by 100 points tall and use the blue dotted layout guides to center it vertically and horizontally in the main View of the temperature converter. Create an outlet for this to TemperatureConverterViewController and call it "panHandleHolder". We now have to take care of its constraints (we will learn more about Auto Layout and size constraints in Week 5, so just follow along and treat this as a preview for now.)
         1. Select the panHandleHolder then select 'Editor' menu > 'Align' > 'Horizontal Center In Container'.
         2. Select the panHandleHolder then select again then select 'Editor' menu > 'Align' > 'Vertical Center In Container'.

3. Select the panHandleHolder again then select ‘Editor’ menu > ‘Pin > ‘Width’.
4. Select the panHandleHolder again then select ‘Editor’ menu > ‘Pin > ‘Width’.

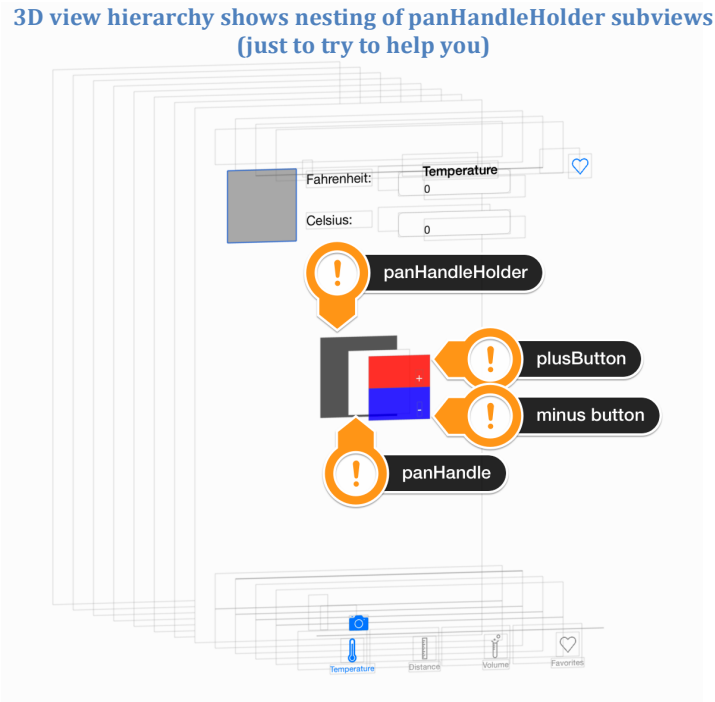
ii. Add another view ***inside*** the panHandleHolder that is 80 points wide by 80 points tall. The color does not matter. Use the layout guides to position it in the center of the panHandleHolder. Create an outlet for this called “panHandle”. Select the panHandle, then repeat Steps 4.b.i.1 – 4 for the panHandle.

iii. Lastly, add a red “+” button and a blue “-“ button ***inside*** of the panHandle. Each button should be 80 points wide by 40 points tall. Thankfully, we do not need to worry about constraints just place them one on top of the other (see image below). Create outlets for these called “plusButton” and “minusButton”. Then create IBAction functions for them and implement as follows.

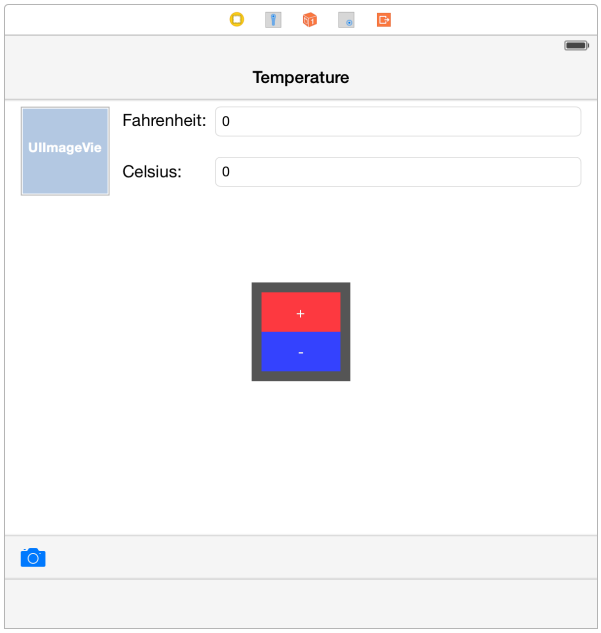
```
@IBAction func plusButtonPressed(sender: AnyObject) {
    let currentValue: Float = (fahrenheitTextField.text as NSString).floatValue
    let newValue = currentValue + 0.5
    fahrenheitTextField.text = "\(newValue)"
    celsiusTextField.text = calculateCelsius(Double(newValue))
}
@IBAction func minusButtonPressed(sender: AnyObject) {
    let currentValue: Float = (fahrenheitTextField.text as NSString).floatValue
    let newValue = currentValue - 0.5
    fahrenheitTextField.text = "\(newValue)"
    celsiusTextField.text = calculateCelsius(Double(newValue))
}
```

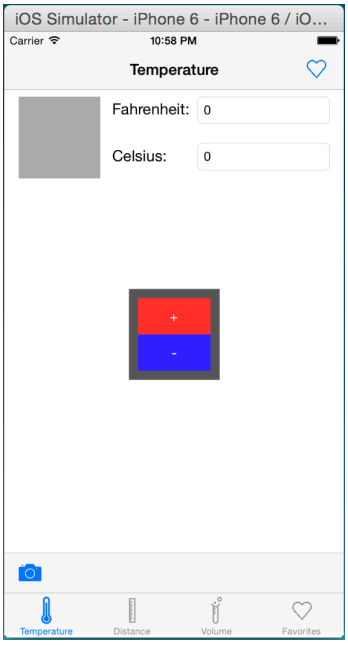
UCI Extension

I&C SCI X402.37
Intro to Mobile Development for Apple iPhone and iPad
2015 Lesson Plans



When you are finished, your converter should appear as follows.

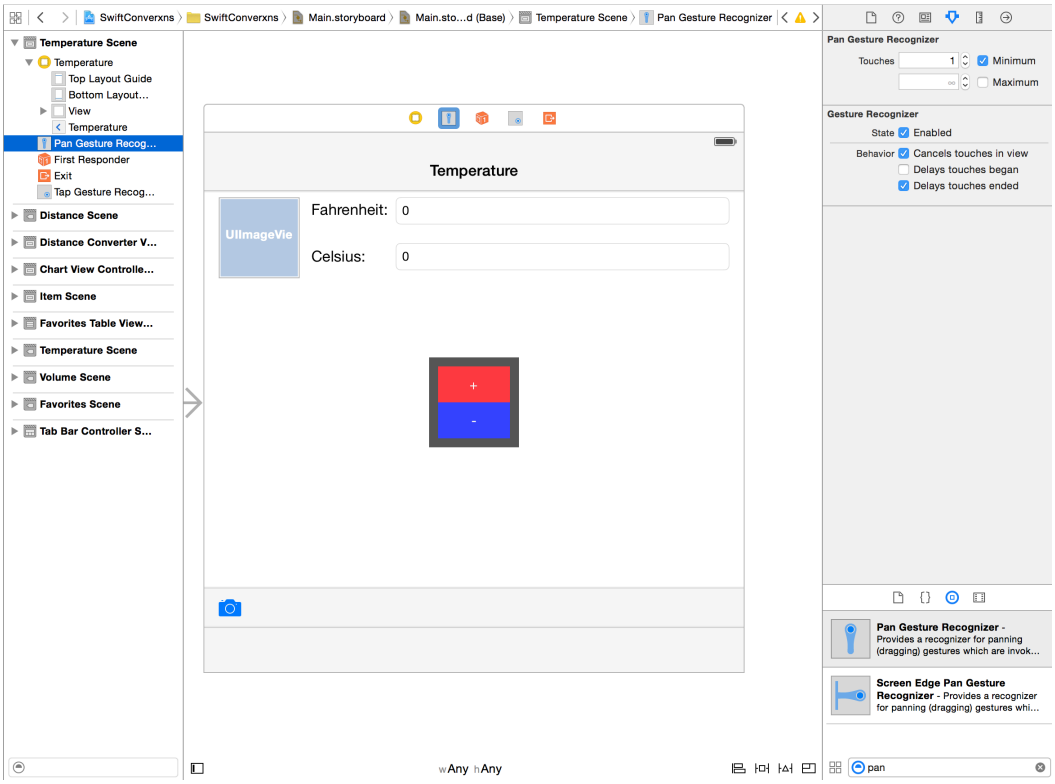




smilefish
Mobile and Social Media Applications

This material was written and developed by Justin-Nicholas Toyama.

c. In Interface Builder, drag a 'Pan Gesture Recognizer' onto the main view in the Temperature Converter controller on the storyboard. If you look at the dock, you should see it there.



Create an IBOutlet to TemperatureConverterViewController and call it "panGR". Also create an IBAction function and call it "paned". Implement the function as follows on the next page.

```
@IBAction func panned(sender: AnyObject) {

        if panGR.state == .Began
        {

            currentValue = CGFloat((fahrenheitTextField.text as NSString).floatValue)

        }
        else
        if panGR.state == .Changed
        {

            let translationTransform = CGAffineTransformMakeTranslation(panGR.translationInView(self.view).x,
panGR.translationInView(self.view).y)

            panHandle.transform = translationTransform

            var newValue: CGFloat?

            if panGR.translationInView(self.view).y < 0 && panGR.locationInView(self.view).y <
self.view.frame.size.height/2
            {

                newValue =  currentValue! - panGR.translationInView(self.view).y

                println("UP: current: \(currentValue) translation.y: \(panGR.translationInView(self.view).y)
newValue:\(newValue)")

            }
            else
            if panGR.translationInView(self.view).y > 0 && panGR.locationInView(self.view).y >
self.view.frame.size.height/2
            {

                newValue = currentValue! - panGR.translationInView(self.view).y

                println("DOWN: current: \(currentValue) translation.y:
\(panGR.translationInView(self.view).y) newValue:\(newValue)")

            }
            else
            {

                newValue = currentValue

            }

            fahrenheitTextField.text = "\(newValue!)"

            celsiusTextField.text = calculateCelsius((fahrenheitTextField.text as NSString).doubleValue)

        }
        else
        if panGR.state == .Ended
        {

            panHandle.transform = CGAffineTransformIdentity

        }
    }
```

Here, we have an if statement with three conditions:

- `if panGR.state == .Began`
- `else if panGR.state == .Changed`
- `else if panGR.state == .Ended`

Inside of the first case, the user has began dragging their finger and the state of panGR becomes “Began”. In here we get the current value from the fahrenheitTextField and store that into the property we created called currentValue.

Inside the second case, the user is continuing dragging their finger and the state of panGR becomes “Changed”. This is where the magic really happens. I explained this code in class, but briefly, we get the translationInView() of the panGR. If `panGR.translationInView(self.view).y < 0 && panGR.locationInView(self.view).y < self.view.frame.size.height/2`, then the user is moving their finger up in the top half of the screen. We count this as the “up” motion for users to increase the input temperature (newValue). If `panGR.translationInView(self.view).y > 0 && panGR.locationInView(self.view).y > self.view.frame.size.height/2`, then the user is moving their finger down in the bottom half of the screen. We count this as the down motion for the user to decrease the input temperature. If neither of these conditions is satisfied, then the user is moving their finger horizontally, so we do not increase or decrease the input temperature. Lastly, we update fahrenheitTextField to be the newValue and set the celsiusTextField to be the result of our conversion function using fahrenheitTextField’s value. We also create a CGAffineTransform to make the panHandle follow our finger. This whole thing happens over and over again as we keep dragging our finger up and down the screen, so we get a live update as we continue to

pan.

In the third case, the user has lifted their finger off the screen, so we just move the panHandle back to where it was originally.

d. Now, let's make sure that the panGR only starts if the user placed their finger in the panHandle (otherwise there wouldn't be much point of having a handle.) Implement the following method in TemperatureConverterView Controller.swift. (Make sure to conform to the UIGestureRecognizerDelegate protocol!)

```
    func gestureRecognizer(gestureRecognizer: UIGestureRecognizer, shouldReceiveTouch
touch: UITouch) -> Bool {

        var shouldReceive = true

        if gestureRecognizer == panGR
        {
            if touch.view == self.view
            {
                shouldReceive = false
            }
        }

        return shouldReceive
    }
```

This ensures that if I touch somewhere in the main view (i.e. outside of the panHandle) and start panning, the handle will not follow us. This makes sense—I should have to touch the handle in order to drag it. (Actually, if you touch *outside* of the panHandle, but *inside* the panHandleHolder, the handle will still move, but that's unlikely.)

e. Run the application. You should be able to increase and decrease the temperature by sliding the handle around or pressing plus or minus buttons.

5. **Get ready for iPad.** Next week, we will start learning about iPad design paradigms. You have already heard of Master-detail design. iPad makes

greater use of this design with split view controllers. This week, let's stay focused on UIStoryboard. We should be pretty comfortable with storyboards now. But there is one thing we haven't practiced using—UIStoryboardSegue. We will use this to create a master-detail system for our favorites. When we tap on a row in the favorites table, we will see the full details of the conversion in the appropriate converter.

a. In Interface Builder, controlDrag *from* the favorites table view controller *to* the temperature converter view controller on the storyboard, then select the 'Show' manual segue. Select the segue (the arrow) and give it the 'Identifier' of "`showTemperatureDetails`" in the 'Attributes Inspector' (capitalization counts!)
b. We will use this segue to pass the values of the text fields, etc. in the converter when the tap on a row in the favorites table view. In FavoritesTableViewController.swift, implement the following methods.

```
    override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath:
NSIndexPath) {

        let favorite: Favorite = favorites[indexPath.row] as Favorite

        if favorite.conversionType == "Temperature"
        {

            performSegueWithIdentifier("showTemperatureDetails", sender: self)

        }

    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

        if segue.identifier == "showTemperatureDetails"
        {

            let index = tableView.indexPathForSelectedRow()?.row

            var tempVC = (segue.destinationViewController as
TemperatureConverterViewController)

            tempVC.favorite = favorites[index!] as? Favorite

        }

    }
```

The real magic here is in the prepareForSegue() function. We may have multiple segues, so we first check the segue.identifier. Then, since we know where that segue points to on the storyboard, we can know the class of the segue.destinationViewController. Now, we set the favorite property of that destination view controller (tempVC). Inside of TemperatureConverterViewController, let's go add that property now and add the code to populate the textFields based on the instance of Favorite that gets passed through the segue.

d. In TemperatureConverterViewController.swift, add a property to the top of the class as follows.

```
var favorite: Favorite?
```

We make this optional because TemperatureConverterViewController can be initialized without a value for this property (i.e. the way we've been using it until now). If we don't make it optional, we have to provide a default value or initialize it in an init function, which is unnecessary and potentially dangerous.

e. Now, modify viewDidLoad() to use the Favorite if one was provided (next page).

This week, this detail view will be read only, so we need to disable the favorite button and the camera button if we are coming from the Favorites table.

```
    override func viewDidLoad() {

        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        let favoriteBarButton = UIBarButtonItem(image: UIImage(named:
"favoritesButton_60x60"), style:
            UIBarButtonItemStyle.Plain, target: self, action:
"favoriteButtonPressed:")

            self.navigationItem.rightBarButtonItem = favoriteBarButton

        if let theFavorite = favorite
        {
            if theFavorite.inputUnits == "ºF"
            {
                fahrenheitTextField.text = "\(theFavorite.inputValue)"
                celsiusTextField.text = "\(theFavorite.outputValue)"
                if theFavorite.imageData != nil
                {
                    imageView.image = UIImage(data: theFavorite.imageData!, scale:
1.0)
                }
            }
            else
            if theFavorite.inputUnits == "ºC"
            {
                celsiusTextField.text = "\(theFavorite.inputValue)"
                fahrenheitTextField.text = "\(theFavorite.outputValue)"
                if theFavorite.imageData != nil
                {
                    imageView.image = UIImage(data: theFavorite.imageData!, scale:
1.0)
                }
            }

            self.title = theFavorite.favoriteTitle

            favoriteBarButton.enabled = false

            cameraBarButton.enabled = false

        }

    }
```

Build and run the application. You should be able to create a favorite with a title and image, view it in the favorites table, and then tap on it to see it in the converter again. (You can change the ºF or ºC values if you want, but you will not be able to save a new Favorite from here or update an existing favorite.)