CSCI 4100 Laboratory Nine

Archiving

Introduction

In class we have learned multiple methods of storing data between application sessions. One such method is archiving. In this laboratory you will produce the simple application shown in figure 1. You will start with the basic application, the archive.zip file on Blackboard and add archiving to it.

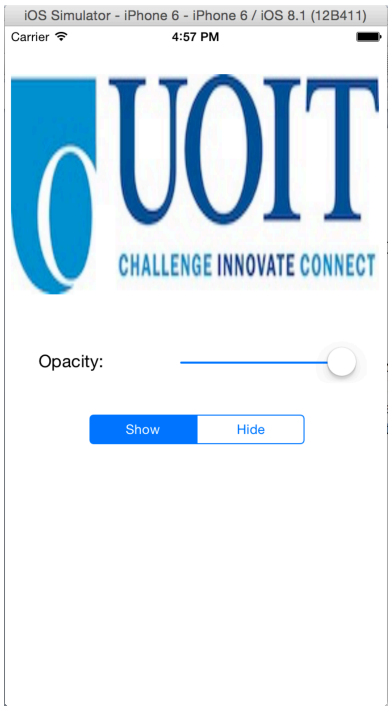

Figure 1 Simple Application

Model Class

You need to create a model class that holds the data to be archived. Create a new Swift class and name it Model. It must conform to both the NSCopying and NSCoding protocols.

The two properties that will be archived are alpha (a float value) and hidden (a BOOL value). You will need to create properties for both of them and while you are at it you can create string constants that will be used as the keys for storing and retrieving these values.

You also need to implement four methods, which are:

- init()
- init(decoder : NSCoder)

- encodeWithCoder(encoder : NSCoder)
- copyWithZone(zone :NSZone)

Note that three of these four methods are required by the two protocols that the object supports. The init method should provide reasonable initial values for the two properties and the other three methods are similar to the ones from the class list example presented in class.

At this point you will need to make some changes to the view controller to save the values entered by the user in the model.

Additions to View Controller

In the view controller we need to archive the data when the application terminates or resigns active and we need to unarchive the data when the application starts. We will start with the archiving side of the view controller.

We need to make the following addition to the start of the view controller code:

```
let kFileName = "archive"


var model = Model()

func dataPath() -> String {
    let paths =
NSSearchPathForDirectoriesInDomains(NSSearchPathDirectory.DocumentDirec
tory,
        NSSearchPathDomainMask.UserDomainMask, true)
    var path = (paths[0] as String)
    path = path + "/" + kFileName
    return path
}

func ApplicationWillTerminate(notification : NSNotification) {
    NSKeyedArchiver.archiveRootObject(model, toFile: dataPath())
}
```

The first function determines the directory where the archive will be stored and then adds the file name onto the end of the directory. The second function does that actual archiving by calling the archiveRootObject function.

We need to make sure that the archiving code is called when the application is about to terminate or resign the active status. To do this we add the following code to the viewDidLoad function:

```
    let app = UIApplication.sharedApplication()
    let center = NSNotificationCenter.defaultCenter()
    center.addObserver(self, selector: "ApplicationWillTerminate:",
         name: UIApplicationWillTerminateNotification, object: app)
```

```
    center.addObserver(self, selector: "ApplicationWillTerminate:",
        name: UIApplicationWillResignActiveNotification, object: app)
```

We first retrieve the current UIApplication object and then the default notification center. We then call addObserver to add notifications for when the application terminates or resigns active status.

Now that we can archive the data we need to consider how we are going to unarchive it. There are two cases to consider here. First, this could be the first time that we run the application. In this case there won't be an archive file and we will need to initialize the model with reasonable values (the init() method that you produced earlier should do this). Second, the archive could already exist. In this case we need to unarchive it and update the user interface. The file manager can assist us with determining the case we are currently handling. Its fileExistsAtPath returns true if the file exists. We can use this in an if statement to handle the two cases.

The code that we use for this is:

```
        let filePath = dataPath()
        let fileManager = NSFileManager.defaultManager()
        if fileManager.fileExistsAtPath(filePath) {
            model =
(NSKeyedUnarchiver.unarchiveObjectWithFile(filePath)as Model)
            slider!.value = model.opacity
            logo.alpha = CGFloat(model.opacity)
            if(model.hidden) {
                toggle.selectedSegmentIndex = 1
                logo.hidden = true
            } else {
                toggle.selectedSegmentIndex = 0;
                logo.hidden = false
            }
        } else {
            model = Model()
        }
```

After we unarchive the model we must update the user interface to reflect the current values in the model.

Report

Show your completed application to the TA. You should show that your application correctly save the current state of the application and restores it when it is restarted.