Saving Data in iOS

Hands-On Challenges

Saving Data in iOS Hands-On Challenges

Challenge A: Tuning into Old Time Radio

Years before there was such a thing called the internet or even television, people gathered around their radios to receive the daily news or play some light music. But in the evenings after the dinner time dishes were washed and dried, families could listen to wealth of enacted stories from some of the most popoular actors.

Vincent Price, Humphry Bogart, and many other famous actors took their listeners on a multitude of adventures. From the depths of Amazon rain forests to the crime ridden streets of New York, no place was witout mystery or susepense.

Fifty years later, these shows still capture the imagination of the young and old alike, and now, due to their distribution over the internet, you can listen to them while working on your iOS apps. They are still as entertaining as when they were first produced.

In this challenge, you are going to work on Old Time Radio iOS app. This challenge will be the basis of several other challenges so it is important that you understand how this app works.

In essence, this app is a data driven app. It contains a listing of twenty five radio episodes from five different old time radio shows. It also contains a list of favorites as well as a listing of web sites that offer additional old time radio shows.

The app is in a skeleton state. The data is contained in a property list. Your first task is convert the data from the property list to actual objects.

Challenge 1: Saving the Graph

First, open **ViewController.m** and insde of videDidLoad:, get the stiring path to the actual data file, **old_time_radio.plist.**

```
NSString * plistPath = [[NSBundle mainBundle]
pathForResource:@"old_time_radio" ofType:@"plist"];
```

Next, convert this string path into an NSURL, and create a NSData from it.

```
NSURL * plistUrl = [NSURL fileURLWithPath:plistPath];
NSData * plistData = [NSData dataWithContentsOfURL:plistUrl
options:NSDataReadingMappedIfSafe error:nil];
```

Finally, convert the NSData into an actual property list

```
NSDictionary * oldTimeRadio = [NSPropertyListSerialization
propertyListFromData:plistData
mutabilityOption:NSPropertyListImmutable format:nil
errorDescription:nil];
```

The returned dictionary has four important keys: Sites, Shows, Episodes, and Favorites. There are four different kinds of model objects that correspond to each: OTRSite, OTRShow, OTREpisode, and OTRFavorite.

You'll be creating four different mutable arrays to store objects from those types.

Add the following:

```
NSMutableArray * sites = [@[] mutableCopy];
NSMutableArray * shows = [@[] mutableCopy];
NSMutableArray * episodes = [@[] mutableCopy];
NSMutableArray * favorites = [@[] mutableCopy];
```

As you create each object, place that object in the appropriate array. You will ultimately archive these arrays once you have finished creating all the objects.

Starting with the Sites directionary, loop through each dictionary and create an object based off property list like so:

```
for (NSDictionary * siteDictionary in oldTimeRadio[@"Sites"])
{
     OTRSite * otrSite = [[OTRSite alloc] init]
     otrSite.name = siteDictionary[@"Name"];
     ...
}
```

Some of these OTR objects contains references to other OTR objects. The relationships are documented in the property list.

For example, the OTRFavorite object has a reference to an OTREpisode object. The actual episode is designated by the key: Episode Id. Each episode has an episode id so you will have to associate the correct OTREpisode to an actual OTRFavorite.

Once you have created the object graph, your challenge actually starts.

Have each OTR object implement the NSCoding protocol by implementing initWithCoder: and encodeWithCoder:. Keep an eye out for weak references. If you see any, make sure to code the property as a conditional object.

To archive the entire graph, place the containing arrays into a dictionary.

```
NSDictionary * objectGraph = @{ @"Favorites" : favorites, @"Shows"
: shows, @"Episodes" : episodes, @"Sites" : sites };
```

Finally, archive the dictionary and save it in your Library directory. Save it under the name, **otr.bin**. Congratulations you archived your graph!

The final part of this challenge is to unarchive the graph.

Inside of **ViewController.m** at the top of **viewDidLoad:** (just above your code to convert your property list into OTR objects), create an instance of NSFileManager to see if the file **otr.bin** actually exists.

If it doesn't exist, then the code should read the property list, create the objects, then finally, archive the result.

If it doe exist, unarchive the file, retrieve each of the arrays, and then print out each OTR object.

Uber Haxx0r Challenge: Importing New Episodes

Having a fixed limit of files with your app can be a little limiting. Thankfully, you can import additional files for only twice the price! :]

Once you have all of the OTR objects parsed and saved to disk, you should then download an additional archive. You can find the archive here:

http://www.raywenderlich.com/downloads/challenges/saving-data/additional_otr.bin

Download the archive and un-archive it. It should contain a dictionary that contains a key of "Shows" and a key of "Episodes". These keys both reference arrays of OTR objects. Integrate these new objects with your current object graph, then archive the result.

Two things will happen will the user press Art-House-ify. First the image will turn black and white, and second, the text will reverse. Once the user presses the button again, the text will return to normal and the picture will return in color.

Once the app makes a change, you should save the change in the NSCachesDirectory, that way, if the user presses the button again, you can fetch the files from the Cache directory and present them to the screen without having do any additional processing.

To get started, open **ViewController.m**, and underneath the imports, make sure to add the following:

```
@import CoreImage;
```

Next, in your code, you're going to add the following Core Image code to turn your images black and white:

```
CIImage * tempImage = [CIImage
imageWithCGImage:self.imageView.image.CGImage];

CIImage * blackAndWhite = [CIFilter
filterWithName:@"CIColorControls" keysAndValues:kCIInputImageKey,
tempImage, @"inputBrightness", [NSNumber numberWithFloat:0.0],
@"inputContrast", [NSNumber numberWithFloat:1.1],
@"inputSaturation", [NSNumber numberWithFloat:0.0],
nil].outputImage;

CIImage * output = [CIFilter filterWithName:@"CIExposureAdjust"
keysAndValues:kCIInputImageKey, blackAndWhite, @"inputEV",
[NSNumber numberWithFloat:0.7], nil].outputImage;

CIContext * context = [CIContext contextWithOptions:nil];
CGImageRef cgiimage = [context createCGImage:output
fromRect:output.extent];
```

```
UIImage * newImage = [UIImage imageWithCGImage:cgiimage];

CGImageRelease(cgiimage);
```

Your challenge is to re-write the saveToDisk:. If the app is art-ifying the image and text, it will first look in the NSCacheDirectory to see if those elements already exist. If they do not exist, the app create them and save them into that directory.

At this point, the button name should change to “Normal-fy” at which point it should display the original contents.

When the app is loaded, the original contents should display as well.