GENERAL ASSEMBLY

MOBILE DEVELOPMENT
FILES & PERSISTENCE: PART 1

Rudd Taylor
Founder, SALT

LEARNING OBJECTIVES

- Recognize the different types of persistence and their pros/cons
- Implement user defaults, but recognize that storage here should be limited
- Create property lists and save/read data from property lists
- Discover the iOS folder structure and where we should store certain types of files
- Use folder search within our applications to read and write files
- Create, read, and write to flat files

INTRO TO PERSISTENCE

WHY PERSIST?

WHY PERSIST?

- Saving data across sessions
  - App high scores
  - App settings
  - User credentials
  - And lots, lots more
- Sharing data between screens

HOW TO PERSIST?

- There are many, many ways to persist data
- There is no ‘right’ way
- Choosing how to persist depends on several things
  - What kind of data am I writing?
  - What kind of data am I reading?
  - Am I storing relations between things? Ex; users and the people they are following.
  - How persistent does my data need to be? Ex, if just caching or saving things temporarily, maybe it doesn’t need to be that persistent.

HOW TO PERSIST?

- There are several built-in options for persisting data in iOS
  - This session:
    - Flat files
    - Property lists
    - User defaults
  - Next session (high level):
    - Core data
    - SQL

Flat files & Property Lists: two ways iOS give us to persist things across sections of our apps.

USER DEFAULTS

As app developers, we know there are directories and files underlying our apps; they are readable by other developers. Hidden to the user, but visible to anyone on the tech.side.

USER DEFAULTS

- A key/value store for storing app settings and other small, independent bits of data

Easy way to share keys and values (think dictionaries). Apple recommends that we store small pieces of data (app settings, flags for showing certain screens in the app) in the NS user defaults.

USER DEFAULTS

- What kind of data am I writing?
  - Small bits of data and an associated key, stored one at a time
  - e.g. A string, a number, a boolean, a dictionary, an array, a dictionary

String, number, date, NS Data, dictionary or array

- What kind of data am I reading?

Reading one at a time. Cant read big blobs of data. Good for app settings, bad for sensitive data. Users can click into Settings > Apps, and see what information is being pulled from their app.

  - Same as above, retrieved one at a time
- Am I storing relations between things?
  - No
- How persistent does my data need to be?

Persistent across app pages, but if the app is deleted, then all the defaults go away.

  - Persistent across app sessions, deleted when the app is deleted

USER DEFAULTS

- Good for:
  - App settings
  - App state
- Not good for:
  - Large data sets
  - Complex relations
  - Sensitive data
  - Caches

NSUSERDEFAULTS

- Storing data:
  - NSUserDefaults.standardUserDefaults().setBool(true, forKey: “mySetting”)
- Retrieving data:
  - NSUserDefaults.standardUserDefaults().boolForKey(“mySetting”) // true
- Very easy interaction with Apple settings menu settings

setBool, setNumber, setArray, setDictionary are other options

There always need to be a matching key for the bool Y / N

this is if you already have the bool code set up.

NSUSERDEFAULTS CODE ALONG

GROUP ASSIGNMENT

- Create an array of athlete names **when your application starts**, store this array in user defaults
- In the view controller, retrieve this list of names and print them out
- Bonus 1: Print these names in a table view
- Bonus 2: Add ability to add a persisted name to the list

PROPERTY LISTS

PROPERTY LISTS

- A bundle of text data (XML, specifically) stored in a text file.

FILES AND PERSISTENCE

PROPERTY LISTS

we can store an array of players, high scorers, list of settings, etc. Exact same kind of data stored in NSUserDefaults.

- What kind of data am I writing?
  - Small-ish sets of data (a few hundred Kb) retrieved file-at-a-time
  - e.g. A string, a number, a boolean, a dictionary, an array, a dictionary
- What kind of data am I reading?
  - Same as above, retrieved file-at-time
- Am I storing relations between things?
  - Not complicated ones
- How persistent does my data need to be?
  - Persistent across app sessions, deleted when the app is deleted

PROPERTY LISTS

- Good for:
  - Persisting lists,
  - Persisting collections of properties
- Not good for:
  - Very data sets
  - Complex relations
  - Sensitive data

PROPERTY LISTS

- Storing data:
  - let cocoaArray: NSArray = someSwiftArray
  - cocoaArray.writeToFile(fileUrlString, atomically: true) // If you have a URL String
  - cocoaArray.writeToURL(fileUrl, atomically: true) // If you have a URL
- Retrieving data:
  - let myArray = NSArray(contentsOfFile: fileUrlString) // If you have a URL String
  - let myArray = NSArray(contentsOfURL: fileUrl) // If you have a URL

Universal Resource Location = URL

below: if we're looking for something… use the line below. NSArray is the Objective C version.

PROPERTY LISTS

- NSUserDefaults is a fancy wrapper around an app-specific plist file
- plists can store:
  - Strings
  - Numbers
  - Date
  - Data
  - Dictionary
  - Array
- Or any combination of the above

PLIST CODE-ALONG

GROUP ASSIGNMENT

- Extend your app to add a plist for coaches with at least 2 coaches. Coaches should have name, years of experience, coach title.
- Print the name of each coach and their title.
- Bonus: Allow user to add coaches through user interface.
- Bonus 2: Display all players and coaches in their own respective table view.

FLAT FILES AND DIRECTORY STRUCTURES

PERSISTENCE AND FILES

DIRECTORY

iDirectory = an App in the App Store (free trial) that lets you look @ the file system for any app.

- Your app bundle contains a lot of useful information
  - Your app binary, and all of its supporting files
  - Your app's documents
  - Settings
  - Temporary files
  - These are stored in various directories

DIRECTORY

- **App/Documents:** User created content. Backed up.
- **App/Documents/Inbox:** for accessing outside entities such as opening email attachments. Backed up.
- **App/YourApp.app:** The app itself and its branding supporting documents. Ie: app icons, app binary, different branding images, fonts, sounds. Not backed up.
- **App/Library:** Most other files that are not user files. This folder itself usually does not contain files, but contains sub folders where content is stored. Backed up.
- **App/Library/Caches:** Temporary data that our app needs to re-create later. Don’t place any important application files in this folder, as system puts least priority on these files and clears out the folder if out of space. Not backed up.
- **App/Library/Preferences:** Preferences that app remembers between launches. Backed up.
- **App/tmp:** Temporary files that app creates and downloads. Used for storing downloaded files to increase app performance. The system may purge these files when app is not in use. Not backed up.

DIRECTORY

- We can get at these directories with NSFileManager
- if let documentPath = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask).first as? NSURL { /* my code*/ } // This gets the documents directory
- let filePath = documentPath.URLByAppendingPathComponent("file.plist", isDirectory: false) // This is for appending a file path onto the directory

If we're using the NSBundle way of loading a file, it only works with files already created on the app, pre-loaded. If we create our own file from scratch and add, it wont work.

FLAT FILES CODE ALONG

GROUP ASSIGNMENT

- Create a new file in the documents folder to store player notes
- The signatures will be in the form of an array of strings
- Append a new note to the end of the list (through user input from single text box) and save it to the signatures file
- Notes are global, not associated with a specific player
- Bonus: List all players, when one is clicked, give the ability to add notes to that player specifically. Those notes should be persisted and printed out when the ‘notes’ screen is navigated to
- Bonus 2: Display and remove player notes through a table view.