GENERAL ASSEMBLY

MOBILE DEVELOPMENT
FILES & PERSISTENCE: PART 2

Rudd Taylor
Founder, SALT

LEARNING OBJECTIVES

- Practice persisting data with plists, flat files, and user defaults
- Breakdown the Core Data stack at a high level
- Create, read, update, delete (CRUD) data with Core Data
- Evaluate when to/not to use Core Data over plists/flat files/user defaults

REVIEW OF PERSISTENCE

USER DEFAULTS

dictionaryForKey, arrayForKey, boolForKey, etc.
Set var for key

- Storing small bits of data
- Backed by an plist shared by the entire app
- A **key/value** store for use in storing app settings, small bits of app-wide data
- How we interact with the Settings app
- NSUserDefaults.standardUserDefaults().boolForKey(“someKey”)
- NSUserDefaults.standardUserDefaults().setBool(true, forKey: “someKey”)

NSUserDefaults = Class. standardUserDefaults
setBool are the keys used to get information

FILES AND PERSISTENCE

PLISTS

When to use PLISTS = same information distributed to all the same users; everyone who opens the app sees the same, fixed thing. plists have a nice graphic visualizer, and we can avoid doing clunky editing in code. Example, our magazines!

- Storing small-ish bits of data that are only applicable to certain parts of your application
- Reading and writing to plists happens all-at-once, and can be atomic (i.e. not able to fail mid-write)

PLISTS = Way to serialize arrays and dictionaries. How I want to save it to disc and populate in the app.

- Can store the same types as user defaults: String, Date, Data, Number, Array, Dictionary
- Backed by XML, though thanks to Xcode's graphical editor and plist serialization methods, we don't really need to both with actual XML
- Almost always, the top-level object in a plist is either an array or a dictionary

PLISTS

- Reading and writing:
  - (["test"] as NSArray).writeToFile(somePathString, atomically: true)
  - let arrayFromFile = NSArray(contentsOfFile: somePathString)
    - somePathString here should include the file name and extension

Swift array as NS array, and then we can pull NS commands.

PATHS

- Apps within iOS are sandboxed from each other
  - They do not share a filesystem
- Even though they aren't usually displayed to the user, apps do have their own directories, files, etc. Each top-level directory has rules about whether it's backed up, whether it can periodically be purged, etc
- As app developers, we are concerned with where files go within our apps (even though users don't ever see the actual directory structure underlying them)

FILES AND PERSISTENCE

PATHS

network error, user typed in wrong password == developer has no control over it.
Logic errors within code, those are things the developer can resolve.

- Getting paths for a built-in file (e.g. plist, PDF, document, etc):
  - NSBundle.mainBundle().pathForResource(“myFileName”, ofType: “plist”)
    - Returns a path (as a String) for the bundled plist file myFileName.plist
    - Returns nil for files that don’t exist
- Getting paths for programmatically generated files:
  - NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory, inDomains: .UserDomainMask)[0]

    Class.singleton.whatToDoToThatDirectory
    - Returns the URL for user’s documents directory
  - Note that if you want to write to a file, you must append the file name
    - let fileUrl = url.URLByAppendingPathComponent("myFile.plist")

BUILDING A TODO APP

TODO APP

- Pair up (groups of 2-3, your choosing)
- User needs to be able to save their name from the settings screen.
- Build a notes app that allows the user to view all notes saved in a table view.
- The user can create a new note and set the file name.
- Users can select and edit the note.
- Recommended structure:
  - Save file names in a plist.
  - Save contents of notes as either dictionaries or flat files (see NSString's initializer with contentsOfFile and encoding)

TableView. When click at the file name, see the note and edit the note for what you have to do.
TODOProject
UITAbleViewControl, UITableViewDataSource
return 1
Link New Referencing Outlet to the main Table View
Create new outlet for it.
Create an array of strings
tableView function return self.items.count

func tableView
var cell = tableView
dequeueReuseableCellwithIdentfier("Mycell") as UITableViewCell!
if cell == nil {
cell = UITableViewCell(style: .Default, reuseIdentifier: "MyCell")

cell.textlabel.text = self.items[indexPath.row]
return cell

Add UINavigationController
Set Navigation Controller 1st thing you see in the app
Create Bar Button
> Sent ACtions L Selector

IBAction func didTapAdd (sender:AnyObject) {
var alert = UIAlertView(title: "Item Name?" message: "Enter an item name",
delegate: self, cancelButtonTitle:"Dismiss", otherButtonTitles: "Add") alert.show

CORE DATA

CORE DATA

- An object persistance framework
- Very powerful, very complicated
- Lots of boilerplate

CORE DATA

- Managed object model (MOM): a file that represents the data model, essentially the database schema.
- Entity: essentially a class definition in Core Data
- Attribute: a property of an entity (a member variable)
- Relationship: link between two entities. This is where entity (table) relationships are defined.
- Managed object: an entity that we want to store in Core Data. Its instances are placed in managed object context.
- Managed object context: this is the virtual representation of our data. This instance of our data can be manipulated as we like and saved when we are ready.

CORE DATA

- We always work on the managed object context
- A series of operations are performed on the MOC (insert, fetch & update, delete), then saved when we want them to persist

CORE DATA CODE ALONG

GROUP ASSIGNMENT

- Clone your first app, use Core Data to store note information instead of files
- Bonus: Add title to notes
- Bonus: To the best of your ability, try and duplicate the UX of the iOS notes app