Stanford CS193p

Developing Applications for iOS
Spring 2016

Today

- Trax Demo Continued
  - Segues (including Adaptive Presentation)
  - Visual Effects (e.g. Blur)
- Persistence (time permitting)
  - Storing things permanently on the device

Demo

- Add User Waypoint in Trax
  - Segueing from an `MKMapView`
  - Draggable `MKAnnotation`
  - Modal segue to a new `EditWaypointViewController`
  - `UITextField` `NSNotifications`
  - Dismissing a modally-presented MVC
  - Unwinding from a modally-presented MVC
  - Popover
  - Controlling Adaptive Presentation Behavior
  - Visual Effects (e.g. Blur)

Persistence

- NSUserDefaults
  - Only for little stuff
- Core Data
  - You're very familiar with this one!
- Archiving
  - Very rarely used for persistence, but it is how storyboards are made persistent
- SQLite
  - Also rarely used unless you have a legacy SQL database you need to access
- File System
  - iOS has a Unix filesystem underneath it
  - You can read and write files into it with some restrictions

Archiving

- There is a mechanism for making ANY object graph persistent
  - Not just graphs with `Array`, `Dictionary`, `NSDate`, etc. in them.
- For example, the view hierarchies you build in Xcode
  - Those are obviously graphs of very complicated objects.
- Requires all objects in the graph to implement `NSCoding` protocol

```
func encodeWithCoder(encoder: NSCoder)
init(coder: NSCoder)
```

- It is extremely unlikely you will use this in this course
  - Obviously we did not in the homework assignments.
  - But almost certainly not in your Final Project either.
  - There are other, simpler, (or more appropriate), persistence mechanisms.

SQLite

- SQL in a single file
  - Fast, low memory, reliable.
  - Open Source, comes bundled in iOS.
  - Not good for everything (e.g. not video or even serious sounds/images).
  - Not a server-based technology
    (not great at concurrency, but usually not a big deal on a phone).
  - API is "C like" (i.e. not object-oriented).
  - Is used by Core Data.

File System

- Accessing files in the Unix filesystem
  1. Get the root of a path into an NSURL
     "Documents" directory or "Caches" directory or ...
  2. Append path components to the URL
     The names of your files (and the directories they reside in)
  3. Write to/read from the files
     Usually done with `NSData` or property list components.
  4. Manage the filesystem with `NSFileManager`
     Create directories, enumerate files in directories, get file attributes, delete files, etc.

File System

- Your application sees iOS file system like a normal Unix filesystem
  - It starts at /.
  - There are file protections, of course, like normal Unix, so you can't see everything.
- And you can only write inside your application's "sandbox"
- Why?
  - Security (so no one else can damage your application)
  - Privacy (so no other applications can view your application's data)
  - Cleanup (when you delete an application, everything it has ever written goes with it)
- So what's in this "sandbox"?
  - Application bundle directory (binary, .storyboards, .jpgs, etc.). This directory is NOT writeable.
  - Documents directory — This is where you store permanent data created by the user.
  - Caches directory — Store temporary files here (this is not backed up by iTunes).
  - Other directories (check out `NSSearchPathDirectory` in the documentation).

File System

- How do you get a path to these special sandbox directories?

  `NSFileManager` (along with `NSURL`) is what you use to find out about what's in the file system.

  You create an `NSFileManager` then find system directories ...

```
let fileManager = NSFileManager()
let urls: [NSURL] = fileManager.URLsForDirectory(NSSearchPathDirectory,
                                                 inDomain: NSUserDomainMask)
```

  There will only be one `NSURL` in the returned `Array` in iOS (different than on Mac).

- Examples of `NSSearchPathDirectory` values

  `.DocumentDirectory`, `.CachesDirectory`, `.DocumentationDirectory`, etc.

  See documentation for more.

NSURL

- Building on top of these system paths

  NSURL methods:

  ```
  func URLByAppendingPathComponent(String) -> NSURL
  func URLByAppendingPathExtension(String) -> NSURL  // e.g. "jpg"
  ```

- Finding out about what's at the other end of a URL

  ```
  var isFileURL: Bool  // is this a file URL (whether file exists or not) or something else?
  func resourceValuesForKeys([String], error: NSErrorPointer) -> [NSObject:AnyObject]?
  ```

  Example keys … NSURLContentAccessDateKey, NSURLIsDirectoryKey, NSURLFileSizeKey

File System

- NSData

  Reading/writing binary data to files

  ```
  init?(contentsOfURL: NSURL)
  func writeToURL(NSURL, atomically: Bool) -> Bool // atomically means "safe write"
  ```

File System

- NSFileManager
  - Provides utility operations
  - Check to see if files exist; create and enumerate directories; move, copy, delete files; etc.
  - Thread safe (as long as a given instance is only ever used in one thread)
  - Examples:

```
let manager = NSFileManager()
func createDirectoryAtURL(NSURL,
  withIntermediateDirectories: Bool,
                   attributes: [NSObject:AnyObject]? // permissions, etc.
) -> Bool throws
func isReadableFileAtPath(String) -> Bool
```

  - Also has a delegate with lots of "should" methods (to do an operation or proceed after an error)
  - And plenty more. Check out the documentation.