EE 443/ EE 593
Mobile Application Development

Robert Lykins

Lecture 12:
Chapter 15:Saving and Loading

Robert Lykins

On the Menu for Today

What We'll Discuss

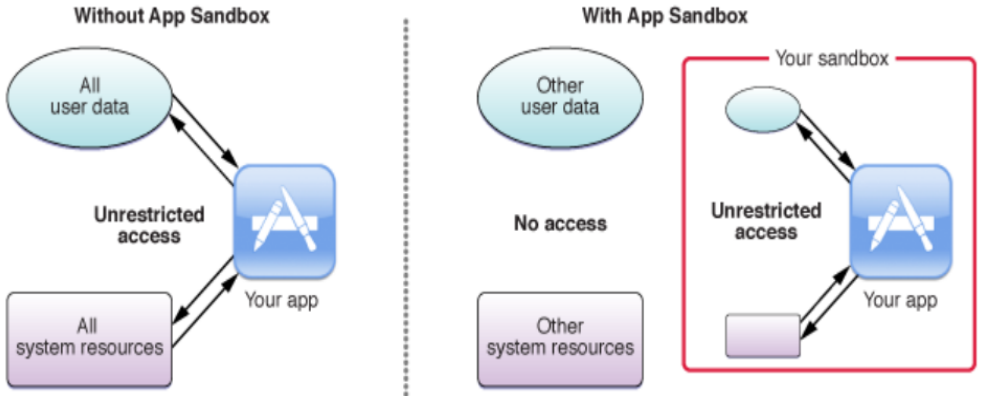
- The "Sandbox"
- Encoding/Decoding of Objects (Serialization) and Property Lists

What We're Programming

- A way to save checklist item objects to a plist
- A way to load stored checklist item objects from a plist

The “Sandbox”

- An environment through which app retrieves all its data in a closed environment
- Any data outside the sandbox the app needs it gets through entitlements (Permissions for data specified in)
- Attempts to contain the damage a compromised app can cause



https://developer.apple.com/library/content/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html

“Sandbox” Directory

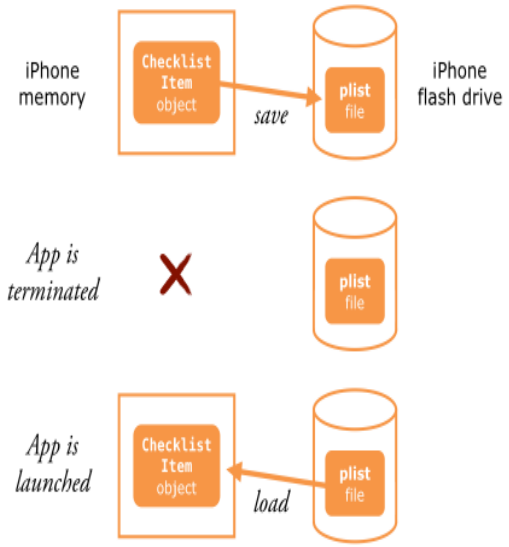
Consists of four main folders:

- Documents: where apps store data files. Use this for data regarding the user
- Library: cache and preference files managed by OS. Use this for data you don’t wish to expose to the user.
- Tmp: Temporary storage of files to be used by app get stored here. Any data that isn’t needed between subsequent launches of app should go here.
- SystemData: Stores system-level data relevant to current app, managed by OS

We’ll be using the Documents folder to store our data

Encoding/Decoding Objects(Serialization)

- Encoding is the process writing an object to a file in a format that can be recognized later
- Decoding is the process of recognizing this format and translating back to objects our app can use
- Think of it like freezing an object in time



We've used it before

- View Controllers in the storyboard are written into file using NSCoder
- This initializer ensures that static data is appended to items before NSCoder decodes the view controller from file

```
required init?(coder aDecoder: NSCoder) {
  items = [ChecklistItem]()                    // add this line

  let row0item = ChecklistItem()               // let
  row0item.text = "Walk the dog"
  row0item.checked = false
  items.append(row0item)                       // add this line
 let row1item = ChecklistItem()                // let
 row1item.text = "Brush my teeth"
 row1item.checked = true
 items.append(row1item)                        // add this line

 let row2item = ChecklistItem()                // let
 row2item.text = "Learn iOS development"
 row2item.checked = true
 items.append(row2item)                        // add this line

 let row3item = ChecklistItem()                // let
 row3item.text = "Soccer practice"
 row3item.checked = false
 items.append(row3item)                        // add this line

 let row4item = ChecklistItem()                // let
 row4item.text = "Eat ice cream"
 row4item.checked = true
 items.append(row4item)                        // add this line

 super.init(coder: aDecoder)
}
```

Plists

- We'll be serializing our objects with the help of Property Lists
- Plists is an XML file format that stores data in a structured list
- A list of settings and their corresponding values

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.
<plist version="1.0">
<dict>
	<key>WINDOW_VIEW</key>
	<dict>
		<key>visibleName</key>
		<string>high window</string>
		<key>examineAppearance</key>
		<string>This high window is simply a rough aperture in
		<key>appearance</key>
		<string>window set high in the concrete wall</string>
		<key>startLocation</key>
		<integer>507</integer>
	</dict>
	<key>BED_SIDE_VIEW</key>
	<dict>
		<key>startLocation</key>
		<integer>503</integer>
		<key>visibleName</key>
		<string>bedroom</string>
		<key>examineAppearance</key>
		<string>The sun is just coming up, lighting this sparse
		<key>appearance</key>
		<string>spare and modern bedroom</string>
	</dict>
	<key>WINDOW_SIDE_VIEW</key>
	<dict>
		<key>visibleName</key>
		<string>bedroom</string>
		<key>examineAppearance</key>
		<string>The spare furnishings make this room feel more
		<key>appearance</key>
		<string>sparely furnished room</string>
		<key>startLocation</key>
		<integer>504</integer>
```

Programming To-Do

- Identify the location of our App's Sandbox and create a plist for our Checklist items

- Utilize the PlistEncoder to create a function which will save our item objects to the plist

- Allow our Checklist items to be serialized by conforming them to Codable protocol

- Utilize the PlistDecoder to create a function which will load item objects from the plist

Locating the Filepath

- documentsDirectory() finds the location of the documents directory for this app
- dataFilePath() appends a reference to the plist we will create
- Documents located in 32-character ID. Think of it like an App Sandbox Identifier

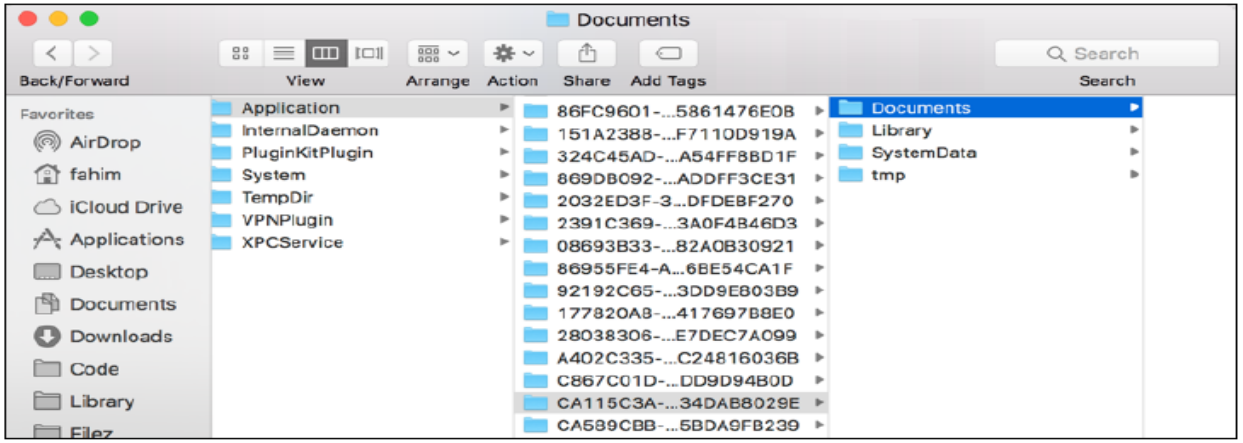
```
func documentsDirectory() -> URL {
  let paths = FileManager.default.urls(for: .documentDirectory,
                                       in: .userDomainMask)
  return paths[0]
}

func dataFilePath() -> URL {
  return documentsDirectory().appendingPathComponent(
                                  "Checklists.plist")
}
```

```
Documents folder is file:///var/mobile/Applications/
FDD50B54-9383-4DCC-9C19-C3DEBC1A96FE/Documents
```

Locating the Filepath

- Command + N opens Finder
- Command + Shift + G and Go to Folder allows you to type in folder paths
- Typing in the document file path shows something like below
- Here we see again the four different folders



Creating a save function

- PropertyListEncoder takes items and converts it to binary data that can be written to file, we also need the path which is provided by dataFilePath()
- Do-try-catch is new block of code that allows for error handling in Swift
- If the line after a try fails, the flow of execution is directed to the catch block, to “catch” the error
- Where do you think we should put this function? (Hint: 4 places)

```
func saveChecklistItems() {
  // 1
  let encoder = PropertyListEncoder()
  // 2
  do {
    // 3
    let data = try encoder.encode(items)
    // 4
    try data.write(to: dataFilePath(),
                   options: Data.WritingOptions.atomic)
    // 5
  } catch {
    // 6
    print("Error encoding item array!")
  }
}
```

Why it won’t work.. yet

- We haven’t told our code that our objects can be encoded and decoded. To do this we must conform to the Codable protocol(includes both Encodable and Decodable)

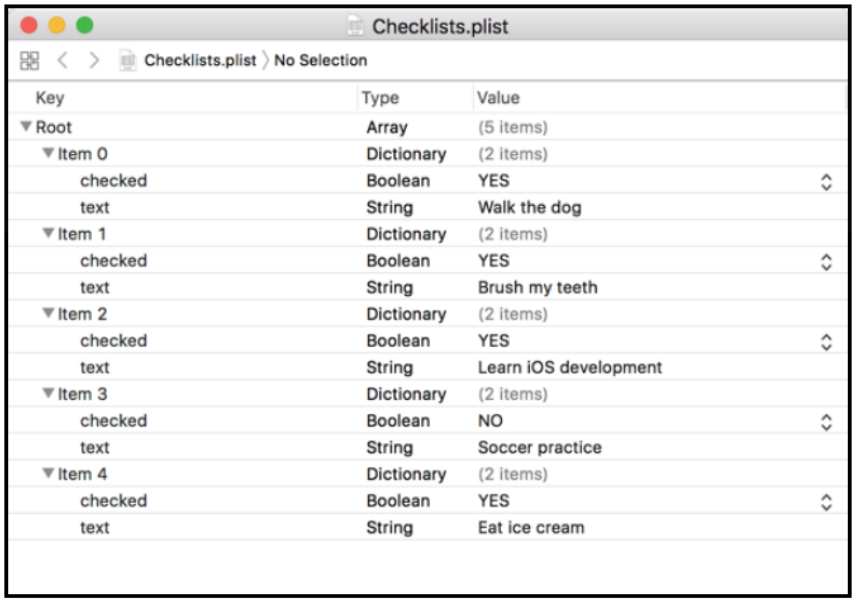
Testing the save

- After testing it, we should look to see if Checklists.plist does indeed store our data
- With the filepath, right-click and

open with xcode and look at the file,

you can also edit the values by clicking

on them



Implementing the Load Function

- Very similar to the Save function except now we're using PropertyListDecoder and decode methods.
- The try? keyword will attempt the code that follows it, on success, the value of data will be as expected, on failure, try? will return nil.
- Why is try? needed here?

```
func loadChecklistItems() {
  // 1
  let path = dataFilePath()
  // 2
  if let data = try? Data(contentsOf: path) {
    // 3
    let decoder = PropertyListDecoder()
    do {
      // 4
      items = try decoder.decode([ChecklistItem].self,
                                 from: data)
    } catch {
      print("Error decoding item array!")
    }
  }
}
```

Implement Load and some Refactoring

- Change items to initialize an array of Checklist items

```
var items: [ChecklistItem]
```

To:

```
var items = [ChecklistItem]()
```

- init?(coder:) is no longer necessary as it loads only static data and because viewDidLoad() executes right after controllers are instantiated which provides the same functionality. Delete it
- viewDidLoad() is executed regardless of whether the controller is in the storyboard or the code
- Add load function to the end of viewDidLoad()

Quick Note about Initializers

- When an item is created such as item = Item(), a method known as init() is called that creates the item.
- More than one init() method can exist for a class so long as the parameter list is different.
- Init?() first checks to see if the object is nill before proceeding, if it is, nothing happens.

```
init() {
  // Put values into your instance variables and constants.

  super.init()

  // Other initialization code, such as calling methods, goes here.
}
```