iPhone Programming
File Management / Undo Management

Tim Gegg-Harrison

Dealing with Data

- Data storage is critical in many apps
- Here we address local storage possibilities
- Several methods available
  - Standard file I/O
  - Using a relational database – SQLite
  - Using Core Data

File Access

- An application and its data files are typically stored together in the “home directory”
- If we utilize this home directory, we need not worry about actual path differences between the simulator and the device

File creation example

- Create a file manager

```
let fileManager: NSFileManager = NSFileManager.default
```

- Check if file/directory exists

```
let dirPath: String = "\(NSHomeDirectory())/tmp/directory"
  var isDir: ObjCBool = true
  if fileManager.fileExists(atPath: dirPath, isDirectory:&isDir) {
      if isDir.boolValue {
          NSLog("file exists and is a directory")
      }
      else {
          NSLog("file exists and is not a directory")
      }
  }
  else {
      NSLog("file does not exist")
  }
```

File creation example…

- Create a new directory

```
do {
        try fileManager.createDirectory(atPath: dirPath,
              withIntermediateDirectories: true, attributes: nil)
    }
    catch {
        NSLog("directory not created")
    }
```

- Create a new file

```
let data: Data? = NSKeyedArchiver.archivedData(withRootObject:
          myNSCodingDataObject)
if fileManager.createFile(atPath: filePath,
                          contents: data,
                          attributes: nil) {
    NSLog("File successfully created")
}
else {
    NSLog("Error creating file")
}
```

Storing/Retrieving data (Data)

- Archiving data:

```
NSKeyedArchiver.archivedData(withRootObject:)
```

- Unarchiving data:

```
NSKeyedUnarchiver.unarchiveObject(with:)
```

NSCoding

- The **`NSCoding`** protocol declares the two methods that a class must implement so that instances of that class can be encoded and decoded
- This capability provides the basis for archiving (where objects and other structures are stored on disk) and distribution (where objects are copied to different address spaces)
- An object being encoded or decoded is responsible for encoding and decoding its instance variables
- A coder instructs the object to do so by invoking `encode(with:)` or `init(coder:)`

NSCoding...

- `encode(with:)`
  - instructs the object to encode its instance variables to the coder provided; an object can receive this method any number of times
- `init(coder:)`
  - instructs the object to initialize itself from data in the coder provided; as such, it replaces any other initialization method and is sent only once per object
- Any object class that should be codable must adopt the `NSCoding` protocol and implement its methods

Undo Management

- Undo support provides a simple interface for managing undo and redo of user actions
- Recording undo actions is performed on a stack
  - User can shake the device to see the most recent action that can be reverted
  - If the user selects the action, the undo operation is executed
  - The undo operation records it counterpart operation so that the user can redo the original operation
  - In essence, there are 2 stacks: undo and redo

Undo Management …

- The undo/redo operations are managed by the `UndoManager` class.
- Any class that inherits from `UIResponder` has its own `UndoManager` instance (`self.undoManager`).

Undo Management …

- Registering an undo operation:

```
func registerUndo<TargetType : AnyObject>(
     withTarget target: TargetType,
     handler: (TargetType) -> Void)
```

- Set name of action:

```
func setActionName(_ actionName: String)
```

Undo Management …

- View controller needs to do the following:
  - Become first responder while its view is showing
  - Resign first responder when the view it is managing disappears