Today

Miscellaneous Topics

- Alerts and Action Sheets
- Notifications
- Application Lifecycle
- Persistence
Alerts and Action Sheets

Two kinds of “pop up and ask the user something” mechanisms

Alerts
Action Sheets

Alerts
Pop up in the middle of the screen.
Usually ask questions with only two answers (e.g. OK/Cancel, Yes/No, etc.).
Can be disruptive to your user-interface, so use carefully.
Often used for “asynchronous” problems (“connection reset” or “network fetch failed”).
Can have a text field to get a quick answer (e.g. password)

Action Sheets
Usually slides in from the bottom of the screen on iPhone/iPod Touch, and in a popover on iPad.
Can be displayed from bar button item or from any rectangular area in a view.
Generally asks questions that have more than two answers.
Think of action sheets as presenting “branching decisions” to the user (i.e. what next?).
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system."
    preferredStyle: .actionSheet
)
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

```swift
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(...)
```
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(UIAlertAction(...))
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(UIAlertAction(
    title: String,
    style: UIAlertActionStyle,
    handler: (action:长大了AlertAction) -> Void
))
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(UIAlertAction(
    title: "Orbit Saturn",
    style: .default
) {
    (action: UIAlertAction) -> Void in
    // go into orbit around saturn
}
)
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(UIAlertAction(
    title: "Orbit Saturn",
    style: UIAlertActionStyle.default)
    { (action: UIAlertAction) -> Void in
        // go into orbit around saturn
    }
)

alert.addAction(UIAlertAction(
    title: "Explore Titan",
    style: .default)
    { (action: UIAlertAction) -> Void in
        if !self.loggedIn { self.login() }
        // if loggedIn go to titan
    }
)
```swift
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)

alert.addAction(UIAlertAction(
    title: "Closeup of Sun",
    style: .destructive)
    { (action: UIAlertAction) -> Void in
        if !loggedIn { self.login() }
        // if loggedIn destroy Cassini by going to Sun
    }
)

alert.addAction(UIAlertAction(
    title: "Cancel",
    style: .cancel)
    { (action: UIAlertAction) -> Void in
        // do nothing
    }
)
```
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)

present(alert, animated: true, completion: nil)
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: .actionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)

present(alert, animated: true, completion: nil)
```
var alert = UIAlertController(
    title: "Redeploy Cassini",
    message: "Issue commands to Cassini’s guidance system.",
    preferredStyle: .actionSheet
)

alert.addAction(/* orbit saturn action */)
alert.addAction(/* explore titan action */)
alert.addAction(/* destroy with closeup of sun action */)
alert.addAction(/* do nothing cancel action */)

alert.modalPresentationStyle = .popover

present(alert, animated: true, completion: nil)
```
var alert = UIAlertController(
    title: “Redeploy Cassini”,
    message: “Issue commands to Cassini’s guidance system.”,
    preferredStyle: .actionSheet
)
alert.addAction( /* orbit saturn action */)
alert.addAction( /* explore titan action */)
alert.addAction( /* destroy with closeup of sun action */)
alert.addAction( /* do nothing cancel action */)
alert.modalPresentationStyle = .Popover
let ppc = alert.popoverPresentationController
ppc?.barButtonItem = redeployBarButtonItem
present(alert, animated: true, completion: nil)
var alert = UIAlertController(  
    title: "Login Required",
    message: "Please enter your Cassini guidance system...",
    preferredStyle: .alert
)
```swift
var alert = UIAlertController(
    title: "Login Required",
    message: "Please enter your Cassini guidance system...",
    preferredStyle: .alert
)

alert.addAction(UIAlertAction(
    title: "Cancel",
    style: .cancel)
{ (action: UIAlertAction) -> Void in
 // do nothing
 })
```
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system…”,
    preferredStyle: .alert)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: “Login”,
    style: .default)
    { (action: UIAlertAction) -> Void in
    // get password and log in
    })
}
var alert = UIAlertController(
  title: "Login Required",
  message: "Please enter your Cassini guidance system...",
  preferredStyle: .alert
)

alert.addAction( /* cancel button action */)

alert.addAction(UIAlertAction(UIAlertAction(
  title: "Login",
  style: .default
) { (action: UIAlertAction) -> Void in
    // get password and log in
    if let tf = self.alert.textFields?.first {
      self.loginWithPassword(tf.text)
    }
  }
))

alert.addTextField(configurationHandler: {
  textField in
    textField.placeholder = "Guidance System Password"
})
var alert = UIAlertController(
    title: “Login Required”,
    message: “Please enter your Cassini guidance system...”,
    preferredStyle: .alert
)

alert.addAction(/* cancel button action */)

alert.addAction(UIAlertAction(
    title: “Login”,
    style: .default
)
{ (action: UIAlertAction) -> Void in
    // get password and log in
    if let tf = self.alert.textFields?.first {
        self.loginWithPassword(tf.text)
    }
})

alert.addTextField(configurationHandler: {
    textField in
        textField.placeholder = “Guidance System Password”
})

present(alert, animated: true, completion: nil)
Demo

Yet more FaceIt!
Add an Alert to FaceIt
MVC

Controller

Model

View

Notification & KVO

Radio Station Communication
Notifications

The “radio station” from the MVC slides. For Model (or global) to Controller communication.

NotificationCenter

Get the default “notification center” via NotificationCenter.default
Then send it the following message if you want to “listen to a radio station” ...

```swift
var observer: NSObjectProtocol // a cookie to later “stop listening” with
observer = NotificationCenter.default.addObserver(
    forKey: NSNotification.Name, // the name of the radio station
    object: Any?, // the broadcaster (or nil for “anyone”)
    queue: OperationQueue? // the queue on which to dispatch the closure below
) {
    // closure executed when broadcasts occur
    let info: Any? = notification.userInfo
    // info is usually a dictionary of notification-specific information
}
```
Notification

What is NSNotificationCenter.Name?
Look this up in the documentation to see what iOS system radio stations you can listen to.
There are a lot.
You will see them as static vars on NSNotificationCenter.Name.
You can make your own radio station name with NSNotificationCenter.Name(String).
More on broadcasting on your own station in a couple of slides ...
Notification

Example of listening to “radio station broadcasts”

Watching for changes in the size of preferred fonts (user can change this in Settings) ...

```swift
let center = NotificationCenter.default
var observer = center.addObserver(
    forKeyName: NSNotification.Name.UIContentSizeCategoryDidChange
    object: UIApplication.shared,
    queue: OperationQueue.main
)
{ notification in
    // re-set the fonts of objects using preferred fonts
    // or look at the size category and do something with it ...
    let c = notification.userInfo?[UIContentSizeCategoryNewValueKey]
    // c might be UIContentSizeCategorySmall, for example
}

center.removeObserver(observer) // when you’re done listening
```
Notification

Posting a Notification

```swift
NotificationCenter.default.post(
    name: NSNotification.Name, // name of the “radio station”
    object: Any?,              // who is sending this notification (usually self)
    userInfo: [AnyHashable:Any]? = nil  // any info you want to pass to station listeners
)
```

Any closures added with addObserver will be executed. Either immediately on the same queue as post (if queue was nil). Or asynchronously by posting the block onto the queue specified with addObserver.
Application Lifecycle

Running your code, but no UI events.
Application Lifecycle

Running your code, receiving and processing UI events.
Application Lifecycle

Running your code for a limited time, no UI events.
Application Lifecycle

Your code not running. You could be killed.
Application Lifecycle

- Not running
- Foreground
  - Inactive
  - Active
- Background
  - Background
- Suspended
Application Lifecycle

Switch to another application
Application Lifecycle

- Not running
- Foreground
  - Inactive
  - Active
- Background
  - Background
- Suspended

Killed (notice no code runs between suspended and killed)
Your AppDelegate will receive ...

```swift
func application(_ application: UIApplication, will/didFinishLaunchingWithOptions: [UIApplicationLaunchOptionsKey: Any]? = nil)
```

... and you can observe ...

```swift
UIApplicationDidFinishLaunching
```

The passed dictionary (also in notification.userInfo) tells you why your application was launched.

Some examples ...

- Someone wants you to open a URL
- You entered a certain place in the world
- You are continuing an activity started on another device
- A notification arrived for you (push or local)
- Bluetooth attached device wants to interact with you
Application Lifecycle

Your AppDelegate will receive ...
func application(UIApplication,
will/didFinishLaunchingWithOptions:
    [UIApplicationLaunchOptionsKey:Any]? = nil)
... and you can observe ...
UIApplicationDidFinishLaunching

It used to be that you would build your UI here. For example, you’d instantiate a split view controller and put a navigation controller inside, then push your actual content view controller. But nowadays we use storyboards for all that. So often you do not implement this method at all.
Application Lifecycle

Your AppDelegate will receive ...

```swift
func applicationWillResignActive(UIApplication)
```

... and you can observe ...

```swift
UIApplicationWillResignActive
```

You will want to “pause” your UI here.
For example, Asteroids would want to pause the asteroids.
This might happen because a phone call comes in.
Or you might be on your way to the background.
Application Lifecycle

Your AppDelegate will receive ...
func applicationDidBecomeActive(UIApplication)

... and you can observe ...
UIApplicationDidBecomeActive

If you have “paused” your UI previously here’s where you would reactivate things.
Application Lifecycle

Your AppDelegate will receive ...

```swift
func applicationDidEnterBackground(UITabBarController)
```

... and you can observe ...

```
UIApplicationDidEnterBackground
```

Here you want to (quickly) batten down the hatches. You only get to run for 30s or so. You can request more time, but don’t abuse this (or the system will start killing you instead). Prepare yourself to be eventually killed here (probably won’t happen, but be ready anyway).
Application Lifecycle

Your AppDelegate will receive ...

```swift
func applicationWillEnterForeground(UICollectionView)
```

... and you can observe ...

`UIApplicationWillEnterForeground`

Whew! You were not killed from background state!
Time to un-batten the hatches.
Maybe undo what you did in `DidEnterBackground`.
You will likely soon be made `Active`.
UIApplicationDelegate

Other AppDelegate items of interest ...

State Restoration (saving the state of your UI so that you can restore it even if you are killed).
Data Protection (files can be set to be protected when a user’s device’s screen is locked).
Open URL (in Xcode’s Info tab of Project Settings, you can register for certain URLs).
Background Fetching (you can fetch and receive results while in the background).
UIApplication

Shared instance
There is a single UIApplication instance in your application
let myApp = UIApplication.shared
It manages all global behavior
You never need to subclass it
It delegates everything you need to be involved in to its UIApplicationDelegate
However, it does have some useful functionality ...

Opening a URL in another application
func open(URL)
func canOpenURL(URL) -> Bool

Registering to receive Push Notifications
func (un)registerForRemoteNotifications()
Notifications, both local and push, are handled by the UNNotification framework.
UIApplication

Setting the fetch interval for background fetching
You must set this if you want background fetching to work ...
func setMinimumBackgroundFetchInterval(TimeInterval)
Usually you will set this to UIApplicationBackgroundFetchIntervalMinimum

Asking for more time when backgrounded
func beginBackgroundTask(withExpirationHandler: ((() -> Void)?) -> UIBackgroundTaskIdentifier
Do NOT forget to call endBackgroundTask(UIBackgroundTaskIdentifier) when you’re done!

Turning on the “network in use” spinner (status bar upper left)
var isNetworkActivityIndicatorVisible: Bool // unfortunately just a Bool, be careful

Finding out about things
var backgroundTimeRemaining: TimeInterval { get } // until you are suspended
var preferredContentSizeCategory: UIContentSizeCategory { get } // big fonts or small fonts
var applicationState: UIApplicationState { get } // foreground, background, active
Info.plist

Many of your application's settings are in Info.plist
You can edit this file (in Xcode's property list editor) by clicking on Info.plist
Many of your application's settings are in Info.plist
You can edit this file (in Xcode's property list editor) by clicking on Info.plist.
Or you can even edit it as raw XML!
Info.plist

Many of your application’s settings are in Info.plist
You can edit this file (in Xcode’s property list editor) by clicking on Info.plist
Or you can even edit it as raw XML!
But usually you edit Info.plist settings by clicking on your project in the Navigator …
Capabilities

Some features require enabling
These are server and interoperability features
Like iCloud, Game Center, etc.

Switch on in Capabilities tab
Inside your Project Settings

Not enough time to cover these!
But check them out!
Many require full Developer Program membership
Familiarize yourself with their existence

<table>
<thead>
<tr>
<th>Feature</th>
<th>On/Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>iCloud</td>
<td>Off</td>
</tr>
<tr>
<td>Game Center</td>
<td>Off</td>
</tr>
<tr>
<td>Passbook</td>
<td>Off</td>
</tr>
<tr>
<td>Apple Pay</td>
<td>Off</td>
</tr>
<tr>
<td>In-App Purchase</td>
<td>Off</td>
</tr>
<tr>
<td>Maps</td>
<td>Off</td>
</tr>
<tr>
<td>Personal VPN</td>
<td>Off</td>
</tr>
<tr>
<td>Background Modes</td>
<td>Off</td>
</tr>
<tr>
<td>Keychain Sharing</td>
<td>Off</td>
</tr>
<tr>
<td>Inter-App Audio</td>
<td>Off</td>
</tr>
<tr>
<td>Associated Domains</td>
<td>Off</td>
</tr>
<tr>
<td>App Groups</td>
<td>Off</td>
</tr>
<tr>
<td>Data Protection</td>
<td>Off</td>
</tr>
<tr>
<td>HomeKit</td>
<td>Off</td>
</tr>
<tr>
<td>HealthKit</td>
<td>Off</td>
</tr>
<tr>
<td>Wireless Accessory Configuration</td>
<td>Off</td>
</tr>
</tbody>
</table>
Persistence

- **User Defaults**
  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, Date, 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 encode(with aCoder: 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 URL
   “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 Data or property list components.

4. Manage the filesystem with FileManager
   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 ...
File System

Getting a path to these special sandbox directories
FileManager (along with URL) is what you use to find out about what's in the file system.
You can, for example, find the URL to these special system directories ...

```swift
let urls: [URL] = FileManager.default.urls(
    for directory: FileManager.SearchPathDirectory.documentDirectory, // for example
    in domainMask: .userDomainMask
)
```
There will only be one URL in the returned Array in iOS (different than on Mac).

Examples of SearchPathDirectory values
.documentDirectory, .cachesDirectory, .documentationDirectory, etc.
See documentation for more.
**URL**

**Building on top of these system paths**

URL methods:

- `func appendingPathComponent(String) -> URL`
- `func appendingPathExtension(String) -> URL`  // 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 resourceValues(for keys: [URLResourceKey]) throws -> [URLResourceKey:Any]`
- `Example keys: .creationDateKey, .isDirectoryKey, .fileSizeKey`
File System

Data

Reading/writing binary data to files

init?(contentsOf: URL)

func write(to url: URL, atomically: Bool) -> Bool // atomically means “safe write”
File System

FileManager
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:
func createDirectory(at url: URL,
    withIntermediateDirectories: Bool,
        attributes: [String:Any]? = nil // permissions, etc.
) -> Bool throws
func isReadableFile(atPath: 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.