Still busy!

- More on Segues
- Modal Alerts and Actions
- Animation
- Notification
- App Lifecycle
- Persistence
- Camera

... and once again plenty of demos!

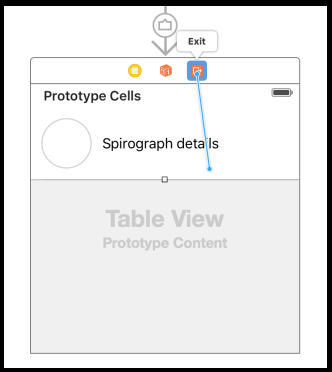
More on Segues

Unwind Segue

- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter

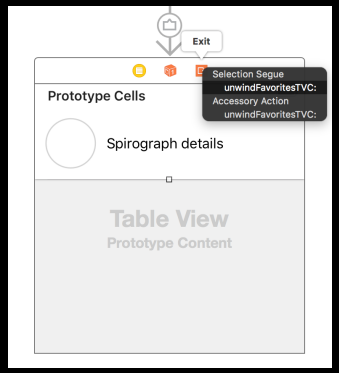
Unwind Segue

- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter
- **How does it work?**
  - Ctrl-drag to the "Exit" button in the same MVC (instead of ctrl-dragging to another MVC)



Unwind Segue

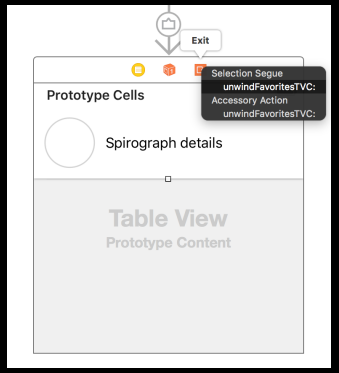
- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter
- **How does it work?**
  - Ctrl-drag to the "Exit" button in the same MVC (instead of ctrl-dragging to another MVC)
  - Then you can choose a special @IBAction method you've created in another MVC




```
@IBAction func unwindFavoritesTVC(unwindSegue: UIStoryboardSegue)
{
    print("\(spirographModel)")
}
```

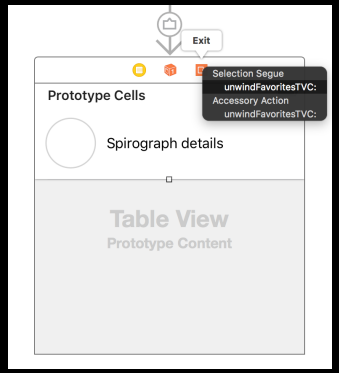
Unwind Segue

- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter
- **How does it work?**
  - Ctrl-drag to the "Exit" button in the same MVC (instead of ctrl-dragging to another MVC)
  - Then you can choose a special @IBAction method you've created in another MVC
  - This means "segue by exiting me and finding a presenter who implements that method".
  - If no presenter (directly or indirectly) implements that method, the segue will not happen



Unwind Segue

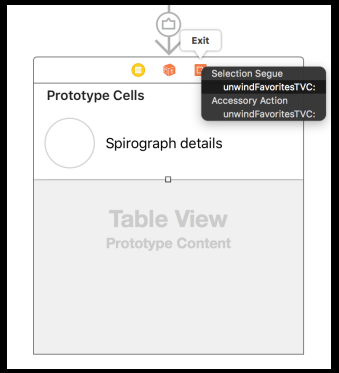
- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter
- **How does it work?**
  - If the @IBAction can be found, you (i.e. the presented MVC) will get to prepareForSegue as normal




```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
     switch segue.identifier! {
     case Identifiers.UnwindFavorites:
         let destinationVC = segue.destinationViewController as! LoggedSpirographVC
         let cell = sender as! SpirographCell
         destinationVC.spirographModel = cell.spirographModel!
     default:
         break
     }
 }
```

Unwind Segue

- **The only segue that does NOT create a new MVC**
  - It can only segue to other MVCs that (directly or indirectly) presented the current MVC
- **Where to use:**
  - Jumping up the stack of cards in a navigation controller (other cards are considered presenters)
  - Dismissing a Modally segued-to MVC while reporting information back to the presenter
- **How does it work?**
  - If the @IBAction can be found, you (i.e. the presented MVC) will get to prepareForSegue as normal
  - Then the special @IBAction will be called in the other MVC and that MVC will be shown on screen
  - MVC will be dismissed in the process (i.e. unpresented and thrown away)



Modal Segues

- **A way of segueing that takes over the screen**
  - Should be used with care.
  - Example: contacts application.
- **Considerations**
  - The view controller we segue to using a Modal segue will take over the entire screen
  - This can be rather disconcerting to the user
- **How do we set a Modal segue up?**
  - Just ctrl-drag from a button to another View Controller
  - Pick segue type "Modal" Inspect the segue to set the style of presentation
  - If you need to present a Modal VC not from a button, use a manual segue ...

```
func performSegue(UIStoryboardSegue, withIdentifier: String)
```

  - ... or, if you have the view controller itself (e.g. Alerts or from instantiateViewController...) ...

```
func presentViewController(UIViewController, animated: Bool, completion: () -> Void)
```

  - In horizontally compact environments, this will adapt to be full screen!
  - In horizontally regular environments, modalPresentationStyle will determine how it appears ... .FullScreen, .OverFullScreen (presenter left underneath), .Popover, .FormSheet, etc.

Modal Segues

- **Preparing for a Modal segue**
  - You prepare for a Modal segue just like any other segue …

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject!) {
    if segue.identifier == "ShowModalVC" {
        let vc = segue.destinationViewController as ModalVC
        // set up the modal mvc
    }
}
```

- **Updating on return from a modally segued View Controller**
  - When the Modal View Controller is "done", how does it communicate results back to presenter?
  - If there's nothing to be said, just dismiss the segued-to MVC
  - To communicate results, generally you would Unwind (though delegation possible too).

Modal Segues

- **How to dismiss a view controller**
  - The presenting view controller is responsible for dismissing (not the presented)
  - If you send this message to the presented view controller, it will forward to its presenter (unless it has presented another MVC, in which case it is the presenter for that other MVC)

```
func dismissViewControllerAnimated(Bool, completion: () -> Void)
```

  - However, remember that unwind automatically dismisses

Modal Segues

- **How to dismiss a view controller**
  - The presenting view controller is responsible for dismissing (not the presented)
  - If you send this message to the presented view controller, it will forward to its presenter (unless it has presented another MVC, in which case it is the presenter for that other MVC)

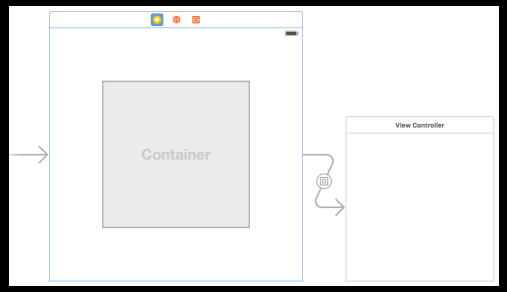
```
func dismissViewControllerAnimated(Bool, completion: () -> Void)
```

  - However, remember that unwind automatically dismisses
- **How is the modal view controller animated onto the screen?**
  - Depends on this property in the view controller that is being put up modally …

```
var modalTransitionStyle: UIModalTransitionStyle
.CoverVertical // slides up and down from bottom of screen (default)
.FlipHorizontal // flips the current view controller view over to modal
.CrossDissolve // old fades out as new fades in
.PartialCurl // only if presenter is full screen (and no more modal presentations coming)
```

Embed Segues

- **Putting a View Controllers's view in another View Controller's view hierarchy…**
  - This can be a very powerful encapsulation technique.
- **Create an Embed Segue in Xcode:**
  - Drag out a Container View from the object palette into the scene you want to embed it in.
  - Xcode automatically sets up an "Embed Segue" from container VC to the contained VC.
  - Embed Segue: works just like other segues e.g. prepareForSegue(sender:)
  - Remember embedded VC's outlets are not set at the time prepareForSegue(sender:) is called like any other segues



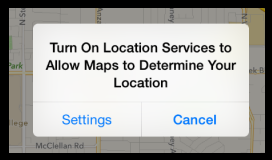
Alerts and Action Sheets

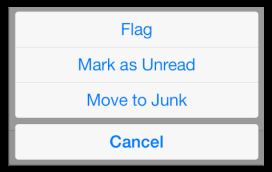
- **Two kinds of "pop up and ask the user something" mechanisms Alerts**
  - Action Sheets
  - Alerts
- **Alerts:**
  - Pop up in the middle of the screen.
  - Usually ask questions with only two (or one) answers (e.g. OK/Cancel, Yes/No, etc.).
  - Modal presentation: can be disruptive to your user-interface, 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?).

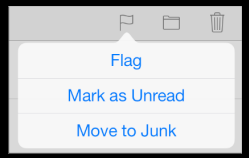
Alerts and Action Sheets

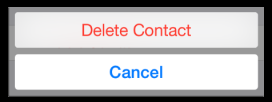
- **Create Alert or Action Sheets:**

```
var alert = UIAlertController(title: "Alert", message: "Msg",
                              preferredStyle: .Action/.ActionSheet)
```









Alerts and Action Sheets

- **Create Alert or Action Sheets:**

```
var alert = UIAlertController(title: "Alert", message: "Msg",
                              preferredStyle: .Action/.ActionSheet)
```

- **Actions:**

```
alert.addAction(UIAlertAction(
    title: String,
    style: UIAlertActionStyle,
    handler: (action: UIAlertAction) -> Void
    ))
```

  - Action style: .Default, .Destructive, .Cancel
  - Also textfield e.g. login window

```
alert.addAction(UIAlertAction(title: "Login", style: .Default)
    { (action: UIAlertAction) -> Void in
        let tf = self.alert.textFields?.first as? UITextField
        if tf != nil { self.loginWithPassword(tf.text) }
} )
alert.addTextFieldWithConfigurationHandler { (textField) in
    textField.placeholder = "Enter Password"
}
```

Alerts and Action Sheets

- **Create Alert or Action Sheets:**

```
var alert = UIAlertController(title: "Alert", message: "Msg",
                              preferredStyle: .Action/.ActionSheet)
```

- **Presenting:**
  - Set presentation style e.g. .Popover or .Alert

```
alert.modalPresentationStyle = .Popover
let ppc = alert.popoverPresentationController
ppc?.barButtonItem = popoverBarButton

presentViewController(alert, animated: true, completion: nil)
```

Demo

Spirograph Artwork

NSTimer

NSTimer

- **Setting up a timer to call a method periodically**
  - You can set it up to go off once at at some time in the future, or to repeatedly go off
  - If repeatedly, the system will not guarantee exactly when it goes off, so this is not "real-time"
  - But for most UI "order of magnitude" activities, it's perfectly fine
  - We don't use it for "animation" (more on that later)
  - It's more for larger-grained activities
- **Run loops**
  - Timers work with run loops
  - So for our purposes, you can only use NSTimer on the main queue
  - Check out docs if you want to learn about run loops and timers on other queue
- **Firing off…**

```
class func scheduledTimerWithTimeInterval(
    _ seconds: NSTimeInterval,
    target: AnyObject,
    selector: Selector (String),
    userInfo: AnyObject?,
    repeats: Bool
)
```

NSTimer

- **Example**

```
let timer = NSTimer.scheduledTimerWithTimeInterval(8.0
    target: self, selector: “fire:”, // must have : at end of selector
    userInfo: nil,
    repeats: true
)
```

  - Every 8 seconds, the method run(NSTimer) will be invoked in self.

- **What does that run method look like?**

```
func run(timer: NSTimer) {
    // do whatever every 8 seconds
    // run on the main queue, don’t take too long
    let theTimersUserInfo = timer.userInfo // provides userInfo  set above
}
```

NSTimer

- **Example**

```
let timer = NSTimer.scheduledTimerWithTimeInterval(8.0
    target: self, selector: “fire:”, // must have : at end of selector
    userInfo: nil,
    repeats: true
)
```

  - Every 8 seconds, the method run(NSTimer) will be invoked in self.

- **Stopping a repeating timer**

```
func run(timer: NSTimer) {
    if doneWithTimer {
        timer.invalidate()
    }
}
```

- **Tolerance**
  - Helps system performance to set a tolerance for “late firing”
  - set timer tolerance property (seconds)
  - No drift: firing time is relative to the start of the timer (not the last time it fired)

Animations

UIView Animations

- **Changes to certain UIView properties can be animated over time e.g.**
  - geometry (e.g. frame)
  - transform (e.g. translation, rotation and scale)
  - opacity (alpha)
- **Achieved with UIView class method(s) using closures**
  - The class methods takes animation parameters and an animation block as arguments.
  - The animation block contains the code that makes the changes to the UIView(s).
  - The changes inside the block are made immediately (even though they will appear “over time”).
  - Most also have another “completion block” to be executed when the animation is done.
- **Animation class method in UIView**

```
class func animateWithDuration(duration: NSTimeInterval, delay: NSTimeInterval,
    options: UIViewAnimationOptions,
    animations: () -> Void,
    completion: ((finished: Bool) -> Void)?)
```

View Animation Example

```
if spirographView.alpha == 1.0 {
    UIView.animateWithDuration(1.0,
        delay: 2.0,
        options: UIViewAnimationOptions.CurveEaseInOut,
        animations: { spirographView.alpha = 0.0 },
        completion: { if $0 { spirographView.removeFromSuperview() } })
    print("spirographView.alpha = \(spirographView.alpha)")
}
```

- This would cause spirographView to fade out over 1 second (starting in 2 seconds).
- Then it would remove spirographView from the view hierarchy (but only if the fade completed).
- If, within the 3s, someone animated the alpha to non-zero, the removal would not happen.
- The output on the console would be … spirographView.alpha = 0.0
- … even though the alpha on the screen won't be zero until the 3 seconds have elapsed

Animation Options

```
UIViewAnimationOptions

.BeginFromCurrentState // interrupt other, in-progress animations of these properties
.AllowUserInteraction // allow gestures to get processed while animation is in progress
.LayoutSubviews // animate the relayout of subviews with a parent's animation
.Repeat // repeat indefinitely
.Autoreverse // play animation forwards, then backwards
.OverrideInheritedDuration // if not set,use duration of any in progress animation
.OverrideInheritedCurve // if not set, use curve (e.g. ease-in/out) of in-progress animation
.AllowAnimatedContent // if not set, just interpolate between current and end
.CurveEaseInOut // slower at the beginning, normal throughout, then slow at end
.CurveEaseIn // slower at the beginning, but then constant through the rest
.CurveLinear // same speed throughout
```

Animation

- **Sometimes you want to make an entire view modification at once I**
  - In this case you are not limited to special properties like alpha, frame and transform
  - Flip the entire view over UIViewAnimationOptionsTransitionFlipFrom{Left,Right,Top,Bottom}
  - Dissolve from old to new state UIViewAnimationOptionsTransitionCrossDissolve
  - Curling up or down UIViewAnimationOptionsTransitionCurl{Up,Down}
- **Use closures again with this UIView class method**

```
UIView.transitionWithView(view: UIView,
    duration: NSTimeInterval,
    options: UIViewAnimationOptions,
    animations: () -> Void,
    completion: ((finished: Bool) -> Void)?)
```

- **Example: flipping a playing card over**
  - Assume cardView draws itself face up or down depending on boolean property faceUp
  - This will cause the card to flip over (from the left edge of the card)

```
UIView.transitionWithView(view: cardView,
    duration: 0.2,
    options: UIViewAnimationOptions.TransitionFlipFromLeft,
    animations: { faceUp = !faceUp },
    completion: nil)
```

Animation of View Hierarchy

- **Animating changes to the view hierarchy is slightly different**
  - In other words, you want to animate the adding/removing of subviews (or (un)hiding them)
  - UIViewAnimationOptionShowHideTransitionViews if you want to use the hidden property otherwise it will actually remove fromView from the view hierarchy and add toView.

```
UIView.transitionFromView(fromView: UIView,
    toView: UIView,
    duration: NSTimeInterval,
    options: UIViewAnimationOptions,
    completion: ((finished: Bool) -> Void)?)
```

Dynamic Animations

Dynamic Animation

- **A little different approach to animation than last week**
  - Set up physics relating animatable objects and let them run until they reach stability
  - Very possible to set it up so this never occurs, but that could be performance problem
- **Steps**
  - Create a UIDynamicAnimator
  - Add UIDynamicBehaviors to it (gravity, collisions, etc.)
  - Add UIDynamicItems (usually UIViews) to the UIDynamicBehaviors
    UIDynamicItem is an protocol which UIView happens to implement

→Things will start animating immediately!

Dynamic Animation

- **Create a UIDynamicAnimator**

```
var animator = UIDynamicAnimator(referenceView: UIView)
```

  - If animating views, all views must be in a view hierarchy with referenceView at the top.
- **Create and add UIDynamicBehavior instances e.g.**

```
let gravity = UIGravityBehavior()
animator.addBehavior(gravity)
let collider = UICollisionBehavior()
animator.addBehavior(collider)
```

- **Add UIDynamicItems to a UIDynamicBehavior**

```
let item1: UIDynamicItem = ... // usually a UIView
let item2: UIDynamicItem = ... // usually a UIView
gravity.addItem(item1)
collider.addItem(item1)
gravity.addItem(item2)
```

  - item1 and item2 will both be affect by gravity
  - item1 will collide with collider's boundaries, but not with item2

Dynamic Animation

- **UIDynamicItem protocol**
  - Any animatable item must implement this ...

```
protocol UIDynamicItem {
    var bounds: CGRect { get } // size cannot be animated
    var center: CGPoint { get set } // position can
    var transform:CGAffineTransform { get set } // also rotation
}
```

  - UIView implements this protocol
  - If you change center or transform while the animator is running, you must call this method in UIDynamicAnimator …

```
func updateItemUsingCurrentState(item: UIDynamicItem)
```

Dynamic Animation Behaviours

- **UIGravityBehavior**

```
var angle: CGFloat // in radians – 0  right, positive increment -> clockwise
var magnitude: CGFloat // 1.0 is 1000 points/s^2
```

- **UIAttachmentBehavior**

```
init(item: UIDynamicItem, attachedToAnchor: CGPoint)
init(item: UIDynamicItem, attachedToItem: UIDynamicItem)
init(item: UIDynamicItem, offsetFromCenter: CGPoint, attachedToItem:Anchor...)
var length: CGFloat // distance between attached things even while animating
var anchorPoint: CGPoint // can be set at any time even while animating
```

  - The attachment can oscillate (i.e. like a spring) and you can control frequency and damping

Dynamic Animation Behaviours

- **UICollisionBehavior**

```
var collisionMode: UICollisionBehaviorMode // .Items, .Boundaries, or .Everything
```

  - If .Items, then any items you add to a UICollisionBehavior will bounce off of each other
  - If .Boundaries, then you add UIBezierPath boundaries for items to bounce off

```
func addBoundaryWithIdentifier(identifier: NSCopying, forPath: UIBezierPath)
func removeBoundaryWithIdentifier(identifier: NSCopying)
var translatesReferenceBoundsIntoBoundary: Bool // referenceView's edges
```

  - How do you find out when a collision happens?

```
var collisionDelegate: UICollisionBehaviorDelegate
```

  - ... this delegate will be sent methods like ...
  - func collisionBehavior(behavior: UICollisionBehavior, began/endedContactForItem: UIDynamicItem, withBoundaryIdentifier: NSCopying) // withItem:atPoint: too
  - The withBoundaryIdentifier is the one you pass to addBoundaryWithIdentifier()
  - It is an NSCopying (NSString & NSNumber are both NSCopying, so String & Int/Double work)
  - In this delegate method you'll have to cast (with as or as?) the NSCopying to what you want

Dynamic Animation Behaviours

- **UISnapBehavior**
  - init(item: UIDynamicItem, snapToPoint: CGPoint)
  - Imagine four springs at four corners around the item in the new spot.
  - You can control the damping of these "four springs" with var damping: CGFloat
- **UIPushBehavior**

```
var mode: UIPushBehaviorMode // .Continuous or .Instantaneous
var pushDirection: CGVector
```

  - ... or ...

```
var angle: CGFloat // in radians and ...
var magnitude: CGFloat // magnitude 1.0 moves a 100x100 view at 100 points/s^2
```

  - Interesting aspect to this behaviour, if you push .Instantaneous, what happens after it's done?
  - It just sits there wasting memory, needs to be cleared up

Dynamic Animation Behaviours

- **UIDynamicItemBehavior**
  - Sort of a special "meta" behavior.
  - Controls the behavior of items as they are affected by other behaviors.
  - Any item added to this behavior (with addItem) will be affected by ...

```
var allowsRotation: Bool
var friction: CGFloat
var elasticity: CGFloat
```

  - See docs for others
  - Can also get information about items with this behavior ...

```
func linearVelocityForItem(UIDynamicItem) -> CGPoint
func addLinearVelocity(CGPoint, forItem: UIDynamicItem)
func angularVelocityForItem(UIDynamicItem) -> CGFloat
```

Dynamic Animation Behaviours

- **UIDynamicBehavior**
  - Superclass of behaviors.
  - You can create your own subclass which is a combination of other behaviours.
  - Usually you override init method(s) and addItem and removeItem to call …

```
func addChildBehavior(UIDynamicBehavior)
```

  - This is a good way to encapsulate a physics behaviour that is a composite of other behaviours.
  - You might also have some API which helps your subclass configure its children.
- **All behaviors know the UIDynamicAnimator they belong to**
  - They can only be part of one at a time.

```
var dynamicAnimator: UIDynamicAnimator { get }
```

  - And the behavior will be sent this message when its animator changes ...

```
func willMoveToAnimator(UIDynamicAnimator)
```

Dynamic Animation Behaviours

- **UIDynamicBehavior's action property**
  - Every time the behaviour acts on items, a block of code called action (takes no arguments and returns nothing) can be executed ...

```
var action: (() -> Void)?
```

  - You can set this to do anything you want.
  - But it will be called a lot, so make it very efficient.
  - If the action refers to properties in the behavior itself, watch out for memory cycles.

Dynamic Animation Stasis

- **UIDynamicAnimator's delegate tells you when animation pauses**
  - Just set the delegate …

```
var delegate: UIDynamicAnimatorDelegate
```

  - ... and you'll find out when stasis is reached and when animation will resume ...

```
func dynamicAnimatorDidPause(UIDynamicAnimator)
func dynamicAnimatorWillResume(UIDynamicAnimator)
```

Dynamic Animation Caveat

- **Memory Cycle**
  - When setting an action, be careful to avoid a memory cycle
- **Example: .Instantaneous UIPushBehavior**
  - When push is done acting on its items, it would be nice to remove it from its animator
  - We can do this with the action method, but we must be careful to avoid a memory cycle ...

```
if let pushBehavior = UIPushBehavior(items: [...], mode: .Instantaneous) {
    pushBehavior.magnitude = // ...
    pushBehavior.angle = // ...
    pushBehavior.action = {
       pushBehavior.dynamicAnimator!.removeBehavior(pushBehavior)
    }
    animator.addBehavior(pushBehavior) // will push right away
}
```

  - The above has a memory cycle because the action captures a pointer back to itself
  - So neither the action closure nor the pushBehavior can ever leave the heap

Dynamic Animation Caveat

- **Memory Cycle**
  - When setting an action, be careful to avoid a memory cycle
- **Example: .Instantaneous UIPushBehavior**
  - When push is done acting on its items, it would be nice to remove it from its animator
  - We can do this with the action method, but we must be careful to avoid a memory cycle ...

```
if let pushBehavior = UIPushBehavior(items: [...], mode: .Instantaneous) {
    pushBehavior.magnitude = // ...
    pushBehavior.angle = // ...
    pushBehavior.action = { [unowned pushBehavior] in
       pushBehavior.dynamicAnimator!.removeBehavior(pushBehavior)
    }
    animator.addBehavior(pushBehavior) // will push right away
}
```

  - Now it no longer captures pushBehavior
  - This is safe to mark unowned because if the action closure exists, so does the pushBehavior
  - When the pushBehavior removes itself from the animator, the action won't keep it in memory
  - So they'll both leave the heap because the animator no longer points to the behaviour

Demo

Gravity Bubbles

Demo

- **Dynamic Animation**

Notification

NSNotification

- **Notifications**
  - The radio station from the MVC slides.
  - For Model (or global) to Controller communication.
- **NSNotificationCenter**
  - Get the default "notification center" via NSNotificationCenter.defaultCenter()
  - Then send it the following message if you want to "listen to a radio station" ...

```
func addObserverForName(String, // the name of the radio station
                        object: AnyObject?, // the broadcaster (or nil for "anyone")
                         queue: NSOperationQueue?) // the queue to execute the closure on
{ (notification: NSNotification) -> Void in
   let info: [NSObject:AnyObject]? = notification.userInfo
   // info is a dictionary of notification-specific information
}
```

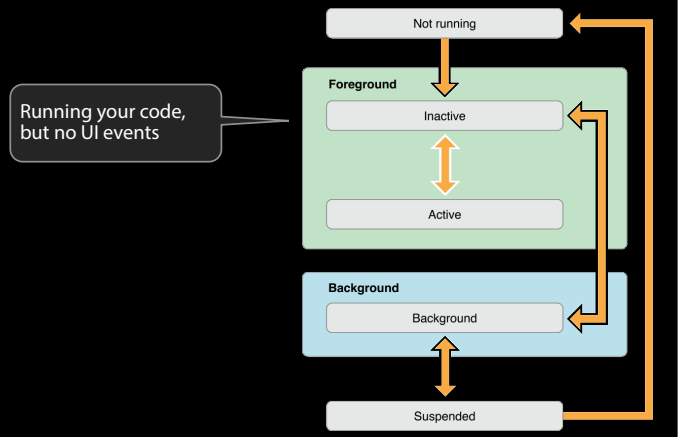
NSNotification

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

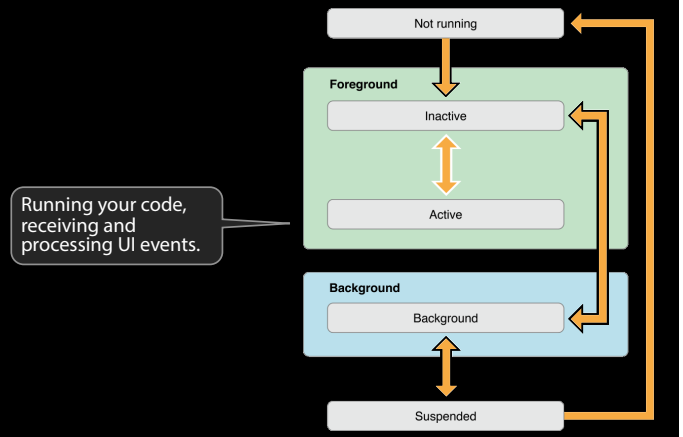
```
let center = NSNotificationCenter.defaultCenter()
center.addObserverForName(UIContentSizeCategoryDidChangeNotification,
    object: UIApplication.sharedApplication(),
    queue: NSOperationQueue.mainQueue())
{ notification in
    // re-set the fonts of objects using preferred fonts
    // or look at the size category and do something with it ...
    let c = notification.userInfo?[UIContentSizeCategoryNewValueKey]
    // c might be UIContentSizeCategorySmall
}
```

App Life Cycle

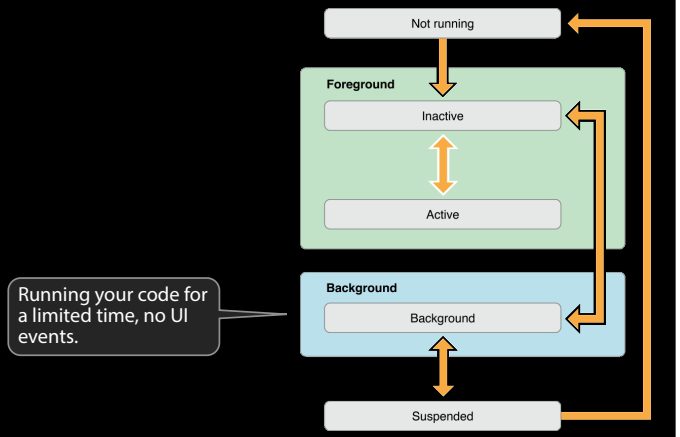
State changes in an iOS app



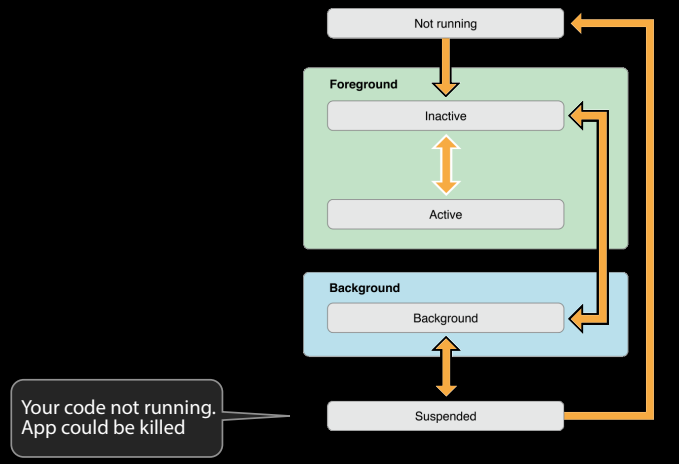
State changes in an iOS app



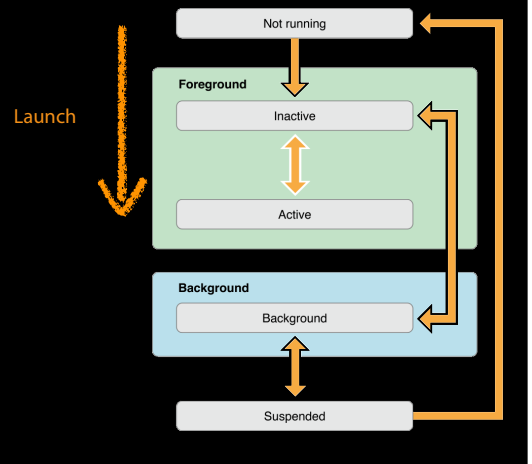
State changes in an iOS app



State changes in an iOS app

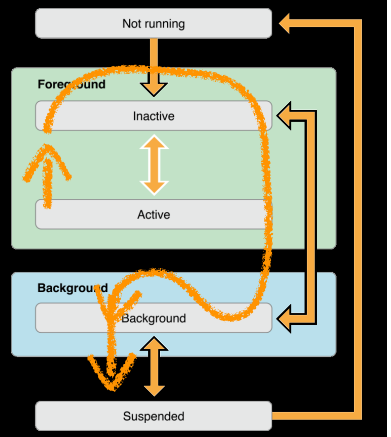


State changes in an iOS app



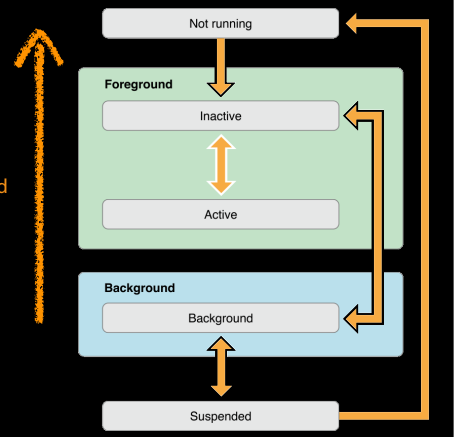
State changes in an iOS app

Switch to another App



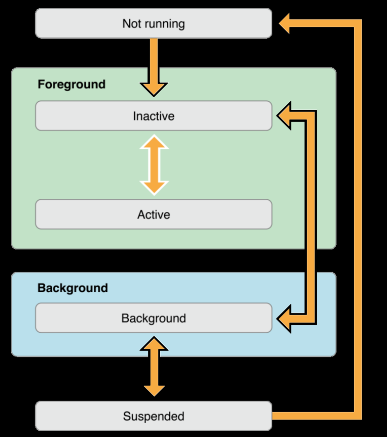
State changes in an iOS app

App Killed



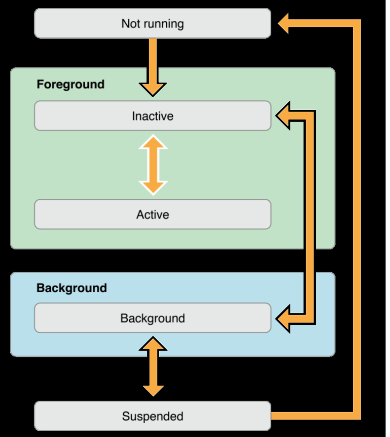
State changes in an iOS app

- **Your AppDelegate will receive**
  - application:willFinishLaunchingWithOptions:
    This method is your app's first chance to execute code at launch time.
  - application:didFinishLaunchingWithOptions:
    This method allows you to perform any final initialization before your app is displayed to the user.
  - applicationDidBecomeActive:
    Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.
  - applicationWillResignActive:
    Lets you know that your app is transitioning away from being the foreground app. Use this method to put your app into a quiescent state.



State changes in an iOS app

- **Your AppDelegate will receive**
  - applicationDidEnterBackground:
    Lets you know that your app is now running in the background and may be suspended at any time.
  - applicationWillEnterForeground:
    Lets you know that your app is moving out of the background and back into the foreground, but that it is not yet active.
  - applicationWillTerminate:
    Lets you know that your app is being terminated. This method is not called if your app is suspended.



UIApplication

- **Shared instance**
  - There is a single UIApplication instance in your application

```
let myApp = UIApplication.sharedApplication()
```

  - It manages all global behaviour
  - 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 openURL(NSURL)
func canOpenURL(NSURL) -> Boolfunc openURL(NSURL)
```

- **Registering or Scheduling Notifications (Push or Local)**

```
func (un)registerForRemoteNotifications()
func scheduleLocalNotification(UILocalNotification)
func registerUserNotificationSettings(UIUserNotificationSettings) // permit for badges, etc.
```

UIApplication

- **Setting the fetch interval for background fetching**
  - You must set this if you want background fetching to work …

```
func setMinimumBackgroundFetchInterval(NSTimeInterval)
```

  - Usually you will set this to UIApplicationBackgroundFetchIntervalMinimum
- **Asking for more time when backgrounded**

```
func backgroundTaskWithExpirationHandler(handler: () -> 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 networkActivityIndicatorVisible: Bool // unfortunately just a Bool, be careful
```

- **Finding out about things**

```
var backgroundTimeRemaining: NSTimeInterval { get } // until you are suspended
var preferredContentSizeCategory: String { 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
  - 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
- **Many require full Developer Program membership**

Camera

UIImagePickerController

- **Modal view to get media from camera or photo library i.e.**
  - you put it up with presentViewController(animated:completion:)
- **Usage**
  - 1. Create it & set its delegate (it can't do anything without its delegate)
  - 2. Configure it (source, kind of media, user edibility)
  - 3. Present it
  - 4. Respond to delegate methods when user is done/cancels picking the media
- **What the user can do depends on the platform**
  - Almost all devices have cameras, but some can record video, some can not
  - You can only offer camera or photo library on iPad (not both together at the same time)
  - As with all device-dependent API, we want to start by check what's available …

```
class func isSourceTypeAvailable(sourceType: UIImagePickerControllerSourceType) -> Bool
```

  - Source type is .PhotoLibrary or .Camera or .SavedPhotosAlbum (camera roll)

UIImagePickerController

- **But don't forget that not every source type can give video**
  - So, you then want to check …

```
class func availableMediaTypesForSourceType(UIImagePickerControllerSourceType) -> NSArray
```

  - Depending on device, will return one or more of these …

```
kUTTypeImage // pretty much all sources provide this, hardly worth checking with new devices
kUTTypeMovie // audio and video together, only some sources provide this
```

  - These are declared in the MobileCoreServices framework.
- **You can get even more specific about cameras**
  - (Though usually this is not necessary)

```
class func isCameraDeviceAvailable(UIImagePickerControllerCameraDevice) -> Bool
```

  - UIImagePickerControllerCameraDevice.Rear or .Front
  - There are other camera-specific interrogations too, for example …

```
class func isFlashAvailableForCameraDevice(UIImagePickerControllerCameraDevice) -> Bool
```

UIImagePickerController

- **Set the source and media type you want in the picker**
  - Example setup of a picker for capturing video (kUTTypeMovie) …
  - (From here out, UIImagePickerController will be abbreviated UIIPC for space reasons.)

```
let picker = UIImagePickerController()
picker.delegate = self // self has to say it implements UINavigationControllerDelegate too
if UIIPC.isSourceTypeAvailable(.Camera) {
    picker.sourceType = .Camera
    if let availableTypes = UIIPC.availableMediaTypesForSourceType(.Camera) {
        if (availableTypes as NSArray).containsObject(kUTTypeMovie) {
            picker.mediaTypes = [kUTTypeMovie]
            // proceed to put the picker
        }
    }
}
```

UIImagePickerController

- **Editability**

```
var allowsEditing: Bool
```

  - If true, then the user will have opportunity to edit the image/video inside the picker.
  - When your delegate is notified that the user is done, you'll get both raw and edited versions.

- **Limiting Video Capture**

```
var videoQuality: UIIPCQualityType
.TypeMedium // default
.TypeHigh
.Type640x480
.TypeLow
.TypeIFrame1280x720 // native on some devices
.TypeIFrame960x540 // native on some devices
var videoMaximumDuration: NSTimeInterval
```

UIImagePickerController

- **Present the picker**

```
presentViewController(picker, animated: true, completion: nil)
```

  - On iPad, if you are not offering Camera (just photo library), you must present with popover.
  - If you are offering the Camera on iPad, then full-screen is preferred.
  - Remember: on iPad, it's Camera OR Photo Library (not both at the same time).

- **Delegate will be notified when user is done**

```
func imagePickerController(UIIPC, didFinishPickingMediaWithInfo info: NSDictionary) {
    // extract image/movie data/metadata here from info
    presentingViewController.dismissViewControllerAnimated(true, completion: nil)
}
```

- **Also dismiss it when cancel happens**

```
func imagePickerControllerDidCancel(UIIPC) {
    presentingViewController.dismissViewControllerAnimated(true, completion: nil)
}
```

UIImagePickerController

- **What is in that info dictionary?**

```
UIImagePickerControllerMediaType // kUTTypeImage or kUTTypeMovie
UIImagePickerControllerOriginalImage // UIImage
UIImagePickerControllerEditedImage // UIImage
UIImagePickerControllerCropRect // CGRect (in an NSValue)
UIImagePickerControllerMediaMetadata  // Dictionary info about image
UIImagePickerControllerMediaURL // NSURL edited video
UIImagePickerControllerReferenceURL // NSURL original (unedited)
```

- **Saving taken images or video into the device's photo library**
  - Check out ALAssetsLibrary.
  - Or you can use the file system (though much less likely, we'll demo this just for demo purposes).

- **In general, much more sophisticated media capture is available**
  - This UIImagePickerController API is pretty simple, but more powerful API exists.
  - Check out AVCaptureDevice.

UIImagePickerController

- **Overlay View**

```
var cameraOverlayView: UIView
```

  - Be sure to set this view's frame properly.
  - Camera is always full screen (on iPhone/iPod Touch anyway): UIScreen's bounds property.
  - But if you use the built-in controls at the bottom, you might want your view to be smaller.

- **Hiding the normal camera controls (at the bottom)**

```
var showsCameraControls: Bool
```

  - Will leave a blank area at the bottom of the screen (camera's aspect 4:3, not same as screen's).
  - With no controls, you'll need an overlay view with a "take picture" (at least) button.
  - That button should send takePicture() to the picker.
  - Don't forget to dismissModalViewController when you are done taking pictures.

- **You can zoom or translate the image while capturing**

```
var cameraViewTransform: CGAffineTransform
```

  - For example, you might want to scale the image up to full screen (some of it will get clipped).

Persistence

Persistence

- **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
- **Core Data**
  - An object-oriented database
  - Primary way to store data in a sophisticated application
  - Hooks up rather easily to iCloud

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)
```

- **Not very common on iOS**

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 not a big deal on a phone).
  - Is used by Core Data (object-oriented database)

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 the class 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)
```

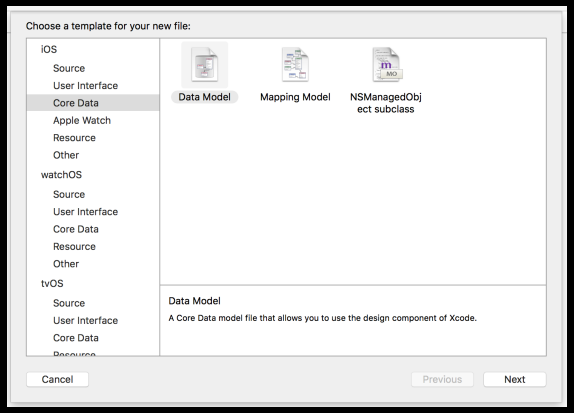
  - There will only be one NSURL in the returned Array in iOS (different on Mac).
  - Examples of NSSearchPathDirectory values NSDocumentsDirectory, NSCachesDirectory, NSDocumentationDirectory, etc. (See documentation for more)

Core Data

- **Where to store data?**
  - Sometimes you need to store large amounts of data or query it in a sophisticated manner.
  - But we still want it to be object-oriented!
- **Enter Core Data**
  - Object-oriented database.
  - Very, very powerful framework in iOS.
- **It's a way of creating an object graph backed by a database**
  - Usually backed by SQL (but also can do XML or just in memory).
- **How does it work?**
  - Create a visual mapping (using Xcode tool) between database and objects.
  - Create and query for objects using object-oriented API.
  - Access the "columns in the database table" using @NSManaged vars on those objects.

Core Data

- **Visual Map**
  - This is your database schema.
  - It is created with New File ...



Core Data

- **Visual Map**
  - This is your database schema.
  - It is created with New File ...
- **Create Entities and Attributes**
  - Which are similar to classes and properties

Core Data

- **Where to store data?**
  - Sometimes you need to store large amounts of data or query it in a sophisticated manner.
  - But we still want it to be object-oriented!
- **Enter Core Data**
  - Object-oriented database.
  - Very, very powerful framework in iOS.
- **It's a way of creating an object graph backed by a database**
  - Usually backed by SQL (but also can do XML or just in memory).
- **How does it work?**
  - Create a visual mapping (using Xcode tool) between database and objects.
  - Create and query for objects using object-oriented API.
  - Access the "columns in the database table" using @NSManaged vars on those objects.

NSURL

- **Building on top of these system paths**
  - NSURL methods:

```
func URLByAppendingPathComponent(String) -> NSURL
func URLByAppendingPathExtension(String) -> NSURL
```

- **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"
```

- **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 ...
    error: NSErrorPointer) -> Bool
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.