BUT WAIT...

CLOUDKIT AND CORE DATA

- WWDC 2019 announcement

Overview Transcript

Using Core Data With CloudKit

CloudKit offers powerful, cloud-syncing technology while Core Data provides extensive data modeling and persistence APIs. Learn about combining these complementary technologies to easily build cloud-backed applications. See how new Core Data APIs make it easy to manage the flow of data through your application, as well as in and out of CloudKit. Join us to learn more about combining these frameworks to provide a great experience across all your customers' devices.

CLOUDKIT AND CORE DATA

- There are actually 2 ways to use CloudKit
  - Manually
  - Fully managed with Core Data Sync

Setting Up Core Data with CloudKit

Set up the classes and capabilities that sync your store to CloudKit.

Overview

To sync your Core Data store to CloudKit, you enable the CloudKit capability for your app. You also set up the Core Data stack with a persistent container that is capable of managing one or more local persistent stores that are backed by a CloudKit private database.

Configure a New Xcode Project

When you create a new project, you specify whether you want to add support for Core Data with CloudKit directly from the project setup interface. The resulting project instantiates an `NSPersistentCloudKitContainer` in your app's delegate. Once you enable CloudKit in your project, you use this container to manage one or more local stores that are backed with a CloudKit database.

1. Choose File > New > Project to create a new project.
2. Select a project template to use as the starting point for your project, and click Next.
3. Select the Use Core Data and Use CloudKit checkboxes.
4. Enter any other project details and click Next.
5. Specify a location for your project and click Create.

THIS METHOD IS MORE CORE DATA THAN

Frame
Core D
On Th
Overvi
See Al

IOS NEWS

CLASS NEWS

- Get on Apple's good side

Latest App Store redesign for iOS shows 800% increase in downloads for Featured apps

Peter Cao - Apr. 20th 2018 11:49 am PT @iPeterCao

IOS 11 APP STOR

CLASS NEWS

- Sign-up for presentations send me a date and idea in Slack

```
journal.json
{

  Entries = [
          {
            uuid = XXXXXXXX
            sync = true;
            cloudKitRecordID = ;

          },
          {
            uuid = XXXXXXXX
            sync = false;
            cloudKitRecordID = nil;

          },
        ...]
}
```

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

EMBEDDED FRAMEWORKS

EMBEDDED FRAMEWORKS

- Frameworks
  - Code
  - Resources (images, sounds)
  - You can share them between projects
  - Working with pre/post build scripts, you can automate including them in various projects

- You might want to package your code the same way

EMBEDDED FRAMEWORKS

- Pre iOS8
  - Include static libraries, but the process was not straightforward
  - Frameworks were a matter of semantics
- iOS8 added ability to create true custom frameworks
  - Common code used between projects

EMBEDDED FRAMEWORKS

SUBTITLE

- Required if you want to "share" code within your own application
  - Extensions
  - Watch app

EMBEDDED FRAMEWORKS

- Create a new framework
  - Editor > Add Target

EMBEDDED FRAMEWORKS

EMBEDDED FRAMEWORKS

New target in file Navigator

EMBEDDED FRAMEWORKS

Linked to main project

EMBEDDED FRAMEWORKS

2017-Session3-EmbeddedFrameworks 〉 GreetingKit 〉 GreetingKit.h 〉 No Selection

- 2017-Session3-E...ddedFrameworks
  - 2017-Session3-E...ddedFrameworks
    - AppDelegate.swift
    - ViewController.swift
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
  - GreetingKit
    - GreetingKit.h
    - Greetings.swift
    - Info.plist
  - Products

```
//
//  GreetingKit.h
//  GreetingKit
//
//  Created by T. Andrew Binkowski on 4/17/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

#import <UIKit/UIKit.h>

//! Project version number for GreetingKit.
FOUNDATION_EXPORT double GreetingKitVersionNumber;

//! Project version string for GreetingKit.
FOUNDATION_EXPORT const unsigned char GreetingKitVersionString[];

// In this header, you should import all the public headers of your framework using statements like #import <GreetingKit/PublicHeader.h>
```

If you are using objective C or if you have any objective C files in your framework, then you will include all public headers of your framework here.

If you don't have objective C code in our framework, leave this file as is.

EMBEDDED FRAMEWORKS

- Not all Cocoa Touch APIs are available for use in extensions
- For instance extensions cannot do the following:
  - Access the camera or microphone on an iOS device
  - Receive data using AirDrop (It can however send data using AirDrop)
  - Perform long-running background tasks
  - Use any API marked in header files with the NS_EXTENSION_UNAVAILABLE macro, or similar unavailability macro, or any API in an unavailable framework for example EventKit and HealthKit are unavailable to app extensions.
  - Access a `sharedApplication` object, and so cannot use any of the methods on that object

There are different applications

EMBEDDED FRAMEWORKS

You can have the compiler let you know when you use disallowed APIs.

EMBEDDED FRAMEWORKS

EMBEDDED FRAMEWORKS

EMBEDDED FRAMEWORKS

ACCESS CONTROL

- If your class is marked **private**
  - All variables, inits, and functions will default to private
- If your class is marked **internal (which is default), public, or open**
  - All variables, inits, and functions will default to internal
- A subclass's superclass must be at least as accessible
- Classes and class members declared public in Objective-C are imported into Swift as open
  - No such distinction in Objective-C

```swift
import Foundation

public class Greetings {

  public init() {
    // This initializer intentionally left empty
  }

  public func happy() {
    print("😃")
  }

  public func crazy() {
    print("🤪")
  }

}
```

The default init is picking up the default internal because the last thing Swift wants to do is expose your init as public API unless it is explicitly instructed to do so.

2017-Session3-EmbeddedFrame

Filter

EMBEDDED FRAMEWORKS

ACCESS CONTROL

- Open vs. Public

Open is more public than public

is a new access le he implementation of

SE-0117 Allow distin s and public overridability

vailable with the Swift 3 snapshot fr m August 7, 2016, and with Xcode 8 beta 6.

ort:

An `open` class is *accessible* and *su classable* outside of the defining module. An member is *accessible* and *overridabl* e outside of the defining module.

A `public` class is *accessible* but *no subclassable* outside of the defining module. class member is *accessible* but *not overridable* outside of the defining module.

`pen` is what `public` used to be in previous Swift releases and the access of `pub` restricted. Or, as Chris Lattner puts it in SE-0177: Allow distinguishing between pu ublic overridability:

en" is now simply "more public than public", providing a very simple and clean mod

ur example, `open var hashValue` is a property which is accessible and can be ov ject subclasses

EMBEDDED FRAMEWORKS

EMBEDDED FRAMEWORKS

- Playground
  - Embedded Framework

APP GROUP

WHAT IS AN APP GROUP?

- Allow applications and targets to share data via a special directory (outside of the sandbox)
  - Creates a folder that can be read/written to by different targets
  - Coordinated with your developer account

APP GROUP

- This should become the default place you think about writing data for all of you applications

HOW DO I CREATE AN APP GROUP?

APP GROUPS

SUBTITLE

- App Groups are allowed to share access to a folder
- When sharing data you must be aware of who may be accessing the data at different times (not obvious)
  - Deal with concurrent writes through NSFileCoordination
  - CoreData and NSUserDefaults handle this out of the box

APP GROUPS

CREATE A NEW GROUP (DEVELOPER BASED)

App Groups

T. Andr...iPhone 6
Finished running DaysDemo on iPhone X
2
DaysDemo
DaysDemo
DaysDemo.entitlements
No Selection
DaysDemo
DaysDemo
DataManager.swift
Model
Entry.swift
Journal.swift
DaysDemo.entitlements
AppDelegate.swift
ViewController.swift
JournalTabl...ntroller.swift
Main.storyboard
Info.plist
Greeting.swift
Key
Type
Value
Entitlements File
Dictionary
(4 items)
APS Environment
String
development
iCloud Container Identifiers
Array
(1 item)
Item 0
String
iCloud.$(CFBundleIdentifier)
iCloud Services
Array
(1 item)
Item 0
String
CloudKit
App Groups
Array
(1 item)
Item 0
String
group.daysdemo
File created when using groups
Same info as in GUI

APP GROUPS

```
// Print out the app group (for fun)
let sharedAppGroup: String = "group.2017-extend-this-app"
let directory: NSURL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)! as NSURL
print("App Group URL: \(directory)")
```

Access the app group container using file manager

- Can read and write to this directory from a container application and application extension points
- No other difference (its just a directory)

APP GROUPS

```
sharedDefaults = UserDefaults(suiteName: sharedAppGroup)
print(sharedDefaults?.dictionaryRepresentation() as Any)
```

- User defaults has a special API for shared defaults in app groups

APP GROUPS

- Playground
  - DataKitManager accessible by different targets

Todays...vorites 〉 iPhone X | ShareExtension | Build **Succeeded** | Today at 1:37 AM | ⚠ 2

ShareExtension 〉 ExtensionKit 〉 DataKitManager.swift 〉 No Selection

```
36
37 /// Store NSUserDefaults in local defaults in app group suite;  Use
38 /// a singleton so that we can easily access the data across different
39 /// extension points
40 public class LocalDefaultsManager: DataKitManager {
41
42   public static let sharedInstance = LocalDefaultsManager()
43   let sharedDefaults: UserDefaults?
44   var favorites: NSMutableArray?
45
46   init() {
47     sharedDefaults = UserDefaults(suiteName: sharedAppGroup)
48     //print(sharedDefaults?.dictionaryRepresentation() as Any)
49   }
50
51   public func add(object anObject: NSObject) {
52     let current: NSMutableArray = currentList()
53     current.add(anObject)
54     sharedDefaults?.set(current, forKey: favoritesKey)
55     sharedDefaults?.synchronize()
56     //print(currentList())
57   }
58
59   ///
60   public func currentList() -> NSMutableArray {
61     var current: NSMutableArray = []
62     if let tempNames: NSArray = sharedDefaults?.array(forKey: favoritesKey) as NSArray? {
63       current = tempNames.mutableCopy() as! NSMutableArray
64     }
65     return current
66   }
67
68   ///
69   public func reset() {
70     sharedDefaults?.set(NSMutableArray(), forKey: favoritesKey)
71     sharedDefaults?.synchronize()
```

ion
nsion
xtension.entitlements
egate.swift
ewController.swift
oryboard
.xcassets
Screen.xib
ting Files
plist
tension
eExtension.entitlements
cessor.js
iewController.swift
erface.storyboard
ting Files
plist
reExtension
iewController.swift
erface.storyboard
st
Kit
Manager.swift
onKit.h
ting Files
vorites
Favorites.entitlements
iewController.swift
erface.storyboard
ting Files
meworks

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

EXTENSIONS

EXTENSIONS

- Way to "extend" the system to provide a way to interface with your application
- More “Surface Area” for you app

EXTENSIONS

SUBTITLE

- Share extension extend the functionality of app
- Allows your app to (sort of) share data with other applications

EXTENSIONS

- Share extension on macOS

EXTENSIONS

SUBTITLE

- Today extension in Notification Center

EXTENSIONS

- Today extension in Notification Center on OSX

EXTENSIONS

SUBTITLE

- Custom action extensions
- Siri Suggestions

EXTENSIONS

- Action Extensions on OSX

EXTENSIONS

SUBTITLE

- Photo extensions

EXTENSIONS

- Document providers

EXTENSIONS

- Finder extensions

EXTENSIONS

SUBTITLE

- Third party keyboards

EXTENSIONS

- Content blockers

EXTENSIONS

SUBTITLE

- List of all extensions
- Some are exclusive macOS, tvOS, iOS

Extension point	Typical app extension functionality
Action (iOS and macOS; UI and non-UI variants)	Manipulate or view content originating in a host app.
Audio Unit (iOS and macOS; UI and non-UI variants)	Generates an audio stream to send to a host app, or modifies an audio stream from a host app and sends it back to the host app.
Broadcast UI (iOS and tvOS)	
Broadcast Upload (iOS and tvOS)	
Call Directory (iOS)	Identify and block incoming callers by their phone number. To learn more, see *CallKit Framework Reference*.
Content Blocker (iOS and macOS)	Indicate to WebKit that your content-blocking app has updated its rules. (This app extension has no user interface.)
Custom Keyboard (iOS)	Replace the iOS system keyboard with a custom keyboard for use in all apps.
Document Provider (iOS; UI and non-UI variants)	Provide access to and manage a repository of files.
	Present information about file sync (This app extension has no user interface.)

HTTPS://DEVELOPER.APPLE.COM/LIBRARY/ARCHIVE/DOCUMENTATION/GENERAL/CONCEPTUAL/EXTENSIBILITYPG/INDEX.HTML

EXTENSIONS

- Most common for iOS applications

Extension point	Typical app extension functionality
Action (iOS and OS X; UI and non-UI variants)	Manipulate or view content originating in a host app
Audio Unit (iOS and OS X; UI and non-UI variants)	Generates an audio stream to send to a host app, or modifies an audio stream from a host app and sends it back to the host app
Content Blocker (iOS and OS X)	Indicate to WebKit that your ad blocking app has updated its rules (This app extension has no user interface)
Custom Keyboard (iOS)	Replace the iOS system keyboard with a custom keyboard for use in all apps
Document Provider (iOS; UI and non-UI variants)	Provide access to and manage a repository of files.
Finder Sync (OS X)	Present information about file sync state directly in Finder (This app extension has no user interface)
Photo Editing (iOS)	Edit a photo or video within the Photos app
Share (iOS and OS X)	Post to a sharing website or share content with others
	Get a quick update or perform a quick task in the

DELIVERING EXTENSIONS

DELIVERING EXTENSIONS

- Extension container is the main application
  - Multiple extensions per container allowed

DELIVERING EXTENSIONS

- Extension points are what your extensions binds to in the system

DELIVERING EXTENSIONS

SUBTITLE

- Purpose built binaries
- Not independent applications
- Accessed via Apple frameworks code
- Not app to app interprocess communication
  - Not the same as UIApplication(openURL:)

DELIVERING EXTENSIONS

- Life cycle of an extension

DELIVERING EXTENSIONS

- Extensions communicate with the host (app that launched it)

DELIVERING EXTENSIONS

- Extension communicates with containing app via `openURL:` share resources

DELIVERING EXTENSIONS

SUBTITLE

- Presentation depends on the type of extension
  - Presented (you see something on the screen)
  - Non-presented (you don't see something)
    - Translator example

DELIVERING EXTENSIONS

- Extensions are bounced through frameworks not launched directly

This can make debugging a bit painful

SETTING UP EXTENSIONS

SETTING UP EXTENSIONS

SETTING UP EXTENSIONS

- Each share sheet has customized templates that provide basic setup of each extension
- Some have default interfaces that you can override

SETTING UP EXTENSIONS

SETTING UP EXTENSIONS

- Add app groups to share data with container extension
  - Possible that you may **<u>not</u>** need to share data between your extension and application

SHARE EXTENSIONS

SHARE EXTENSIONS

SHARE EXTENSIONS

SUBTITLE

- Display name
- Activation rules
- Implement a ViewController
  - Do something with the data you have access to
- Signal completion/ cancellation

Default UI piggybacked from Social Framework

SHARE EXTENSIONS

- Customize the name for your share extension
  - Default is product name (may need to change for visibility)

Doesn't always work...

SHARE EXTENSIONS

- Activation Rules
  - When system show my share extension?
  - What content is available

URLs, text, Images…

extensionContext

image 1	image 2	video

???

Social Network Extension

Photo Sharing Extension

Microblog Extension

Video Sharing Extension

SHARE EXTENSIONS

SHARE EXTENSIONS

▼ NSExtension	Dictionary	(4 items)
▼ NSExtensionAttributes	Dictionary	(1 item)
NSExtensionActivationRule	String	SUBQUERY(extensionItems, $extensionItem, SUBQUERY($exter

```
SUBQUERY(extensionItems, $extensionItem,
SUBQUERY($extensionItem.attachments, $attachment, ANY
$attachment.registeredTypeIdentifiers UTI-CONFORMS-TO "public.image").@count
== 1).@count == 1 OR SUBQUERY(extensionItems, $extensionItem,
SUBQUERY($extensionItem.attachments, $attachment, ANY
$attachment.registeredTypeIdentifiers UTI-CONFORMS-TO "com.adobe.pdf").@count
== 1).@count == 1
```

- Set activation rules in Info.plist of the extension

SHARE EXTENSIONS

Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
NSExtension	Dictionary	(3 items)
NSExtensionAttributes	Dictionary	(2 items)
NSExtensionActivationRule	Dictionary	(6 items)
NSExtensionActivationSupportsFileWithMaxCount	Number	0
NSExtensionActivationSupportsImageWithMaxCount	Number	1
NSExtensionActivationSupportsText	Boolean	YES
NSExtensionActivationSupportsVideoWithMaxCount	Number	0
NSExtensionActivationSupportsWebPageWithMaxCount	Number	1
NSExtensionActivationSupportsWebURLWithMaxCount	Number	1
NSExtensionJavaScriptPreprocessingFile	String	Processor
NSExtensionMainStoryboard	String	MainInterface
NSExtensionPointIdentifier	String	com.apple.share-services

- Condensed rules (instead of SQL query); compile to a predicate

SHARE EXTENSIONS

- Default UI is `SLComposeServiceViewController` subclass that has same interface as system sharing

SHARE EXTENSIONS

- You can additional information about the sharedItem from “extensionContext” property

SHARE EXTENSION FROM PHOTOS

SHARE EXTENSIONS

SUBTITLE

- Share extension with Photos App
  - Get the image
  - Get the text

```
import UIKit
import Social
import ExtensionKit
import MobileCoreServices

/// View Controller that is presented via the share extension.  This uses the
/// default share sheet from the `Social.framework`.
class ShareViewController: SLComposeServiceViewController {

  override func didSelectPost() {
    printLog("📷")

    // This is called after the user selects Post. Do the upload of
    // contentText and/or NSExtensionContext attachments.

    // While it is possible to save an UIImage to a dictionary as NSData as
    // we are doing here, it is generally not a good idea.  You could store
    // image on server or in App Group folder and keep the URL of the location
    // to store

    // Also, note that you get the app store URL, so you could use it to get
    // the image that way.

    if let item = self.extensionContext?.inputItems[0] as? NSExtensionItem {
      for ele in item.attachments!{
        let itemProvider = ele as! NSItemProvider

        if itemProvider.hasItemConformingToTypeIdentifier("public.jpeg"){
          NSLog("itemprovider: %@", itemProvider)
          itemProvider.loadItem(forTypeIdentifier: "public.jpeg", options: nil,
                                completionHandler: { (item, error) in

            var imgData: Data!
            if let url = item as? URL {
              imgData = try! Data(contentsOf: url)
            }

            if let img = item as? UIImage {
              imgData = UIImagePNGRepresentation(img)
            }

            let sharedItem: NSDictionary = [ "date": Date(),
                                             "contentText": self.contentText,
                                             "imageData": imgData]

            LocalDefaultsManager.sharedInstance.add(object: sharedItem)


          })
```

SHARE EXTENSIONS

```
override func didSelectPost() {

  if let item = self.extensionContext?.inputItems[0] as? NSExtensionItem {
    for ele in item.attachments!{
      let itemProvider = ele as! NSItemProvider

      if itemProvider.hasItemConformingToTypeIdentifier("public.jpeg"){
        NSLog("itemprovider: %@", itemProvider)
        itemProvider.loadItem(forTypeIdentifier: "public.jpeg", options: nil,
                              completionHandler: { (item, error) in

          var imgData: Data!
          if let url = item as? URL {
            imgData = try! Data(contentsOf: url)
          }

          if let img = item as? UIImage {
            imgData = UIImagePNGRepresentation(img)
          }

          let sharedItem: NSDictionary = [ "date": Date(),
                                           "contentText": self.contentText,
                                           "imageData": imgData]

          LocalDefaultsManager.sharedInstance.add(object: sharedItem)


        })
      }
    }
  }
```

Get image

Get text

Save somewhere that you app can read it

SHARE EXTENSIONS

```
  override func isContentValid() -> Bool {
    // Do validation of contentText and/or NSExtensionConte
    return true
  }

  override func configurationItems() -> [Any]! {
    // To add configuration options via table cells at the
    // return an array of SLComposeSheetConfigurationItem h
    return []
  }
}
```

NOTIFICATION CENTER WIDGETS

Notification Center Widgets

SUBTITLE

- Widgets are view controllers
  - Use the same methods
- Notification Center handles layout
  - Express preferred height with constraints or `preferredContentSize`
- Your responsibility
  - Animate content along with resize animation
  - Handle update requests

NOTIFICATION CENTER WIDGETS

- Entitlements
- View controller
- Storyboard (optional)

NOTIFICATION CENTER WIDGETS

NOTIFICATION CENTER WIDGETS

- `NCWidgetProviding` protocol allows you to customize appearance of a widget
  - Size
  - Updating
  - Insets

NotificationCenter › NCWidgetProviding

Protocol

NCWidgetProviding

The `NCWidgetProviding` protocol lets you customize some of the appearance and behavior of a widget (a widget is an app extension).

Symbols

Customizing the Display

```
func widgetMarginInsets(forProposedMarginInsets: U
```

Called to let a widget accept the default margin inset values o
instead.

```
func widgetActiveDisplayModeDidChange(NCWidgetDisp
CGSize)
```

NOTIFICATION CENTER WIDGETS

- Data source needs to be accessible from widget
  - App Groups

```
import UIKit
import NotificationCenter
import ExtensionKit

class TodayViewController: UIViewController {

  /// Array to hold all the shared items loaded from NSUserDefaults
  var sharedItems: NSMutableArray = []

  @IBOutlet weak var tableView: UITableView!


  //
  // MARK: - Lifecycle
  //

  override func viewDidLoad() {
    super.viewDidLoad()

    // Size the widget for initial view
    self.extensionContext?.widgetLargestAvailableDisplayMode = NCWidgetDisplayMode.expanded

    // Retrieve the list of apps that is stored in AppGroups UserDefaults
    sharedItems = LocalDefaultsManager.sharedInstance.currentList()
  }

  //
  // MARK: - Table view data source
  //
  func numberOfSectionsInTableView(_ tableView: UITableView) -> Int {
    return 1
  }

  func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return sharedItems.count
  }

  func tableView(_ tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
    let sharedItemDictionary: NSDictionary = sharedItems[indexPath.row] as! NSDictionary

    cell.textLabel!.text = sharedItemDictionary["contentText"] as? String
    cell.detailTextLabel!.text = (sharedItemDictionary["date"] as? Date)!.description

    // The images are stored here as NSData inside of NSUserDefaults, we
    // need to convert to an UIImage before we add it to the table
    if let imageData = sharedItemDictionary["imageData"] as? Data {
      cell.imageView!.image = UIImage(data: imageData)
    }

    return cell
  }
```

NOTIFICATION CENTER WIDGETS

```
// MARK: - Table view data delegate methods
//


func tableView(_ tableView: UITableView, didSelectRowAtIndexPath indexPath: IndexPath) {
  tableView.deselectRow(at: indexPath, animated: true)
  let row = indexPath.row

  // We are using a custom URL to pass the index of the tapped cell
  // You can use this to customize the launch of the container application
  let launchString = "mobiuchicagoshare://indexPathRow=\(row)"
  self.extensionContext?.open(URL(string: launchString)!, completionHandler:nil)
}
```

- Launch the application with a custom URL to pass the cell tapped

NOTIFICATION CENTER WIDGETS

OPEN FROM TERMINATED

```
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    // Print out the app group path for debugging
    let sharedAppGroup: String = "group.extensiontest2"
    let directory: URL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)!
    print("App Group URL: \(directory)")

    // If the app was not in the background, it will launch with the options
    // key `UIAPplicationLaunchOptionsURLKey`
    if let options = launchOptions {
      let alertView = UIAlertView(title: "Launch Options", message: options.description,
                                  delegate: nil,
                                  cancelButtonTitle: "Cancel",
                                  otherButtonTitles: "OK")
      alertView.alertViewStyle = .default
      alertView.show()
    }
    return true
  }
```

OPEN FROM SUSPENDED

```
  func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any) -> Bool {
    print("The application was launched from the url: \(url)")
    return true
  }
```

NOTIFICATION CENTER WIDGETS

```
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    // Print out the app group path for debugging
    let sharedAppGroup: String = "group.extensiontest2"
    let directory: URL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)!
    print("App Group URL: \(directory)")

    // If the app was not in the background, it will launch with the options
    // key `UIAPplicationLaunchOptionsURLKey`
    if let options = launchOptions {
      let alertView = UIAlertView(title:
                                  delegat
                                  cancelB
                                  otherBu
      alertView.alertViewStyle = .defaul
      alertView.show()
    }
    return true
  }

  func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any) -> Bool {
    print("The application was launched from the url: \(url)")
    return true
  }
```

SINCE WE ARE USING THE SAME DATA SOURCE WE CAN CUSTOMIZE THE LAUNCH

NOTIFICATION CENTER WIDGETS

- Dynamic sizing
  - Widget is on the left or right column on iPad
  - Widget is shown in Notification Center or the Today view
  - Accessed from the Home or Lock screen
  - Portrait or landscape orientation
  - Users text size setting

https://pspdfkit.com/blog/2017/today-widget/#

Small Text Size

Large Text Size

NOTIFICATION CENTER WIDGETS

NOTIFICATION CENTER WIDGETS

```
// MARK: - NCWidgetProviding Methods
//


extension TodayViewController : NCWidgetProviding {

  func widgetActiveDisplayModeDidChange(_ activeDisplayMode: NCWidgetDisplayMode, withMaximumSize maxSize: CGSize){
    if (activeDisplayMode == NCWidgetDisplayMode.compact) {
      self.preferredContentSize = maxSize;
    } else {
      self.preferredContentSize = CGSize(width: 0, height: 400);
    }
  }


  func widgetPerformUpdate(completionHandler: (@escaping (NCUpdateResult) -> Void))
    // Perform any setup necessary in order to update the view.
    // If an error is encountered, use NCUpdateResult.Failed
    // If there's no update required, use NCUpdateResult.NoData
    // If there's an update, use NCUpdateResult.NewData

    // Refresh table
    self.tableView.reloadData()
    completionHandler(NCUpdateResult.newData)
  }

```

SIZING IS NOT GUARANTEED, USE STACK VIEWS FOR MORE FLEXIBILITY

NOTIFICATION CENTER WIDGETS

- Quick actions also show up on home screen force touch
- Different appearance based on quick actions

NOTIFICATION CENTER WIDGETS

- Debugging notes
  - Run the container application first
  - Run the extension from its scheme to get debugging information
  - User Interface
    - Optimize for size and visibility
    - Think about how users use today widget

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

SHARE EXTENSION FROM A WEBPAGE

SHARE EXTENSIONS

- Extension parses a web page with javascript

```
var Preprocessor = function() {};

Preprocessor.prototype = {
    run: function(arguments) {
        arguments.completionFunction({"URL": document.URL,
                                      "pageSource": document.documentElement.outerHTML,
                                      "title": document.title,
                                      "selection": window.getSelection().toString()});
    }
};

var ExtensionPreprocessingJS = new Preprocessor;
```

SHARE EXTENSIONS

- SLComposeServiceViewController subclass
- Custom UI will implement typical view controller methods

```
import UIKit
import Social
import ExtensionKit
import MobileCoreServices

/// View Controller that is presented via the share extension.  This uses the
/// default share sheet from the `Social.framework`.
class ShareViewController: SLComposeServiceViewController {

  /// Hold the app icon
  var imageToShare: UIImage?
  /// Hold the app store URL
  var urlToShare: String?
  /// Hold the application title
  var textToShare: String?


  //
  // MARK: - Share extension methods
  //

  override func presentationAnimationDidFinish() {•••}

  override func isContentValid() -> Bool {•••}

  override func didSelectPost() {•••}

  override func configurationItems() -> [Any]! {•••}
}
```

SHARE EXTENSIONS

- Iterate through and get all the attachments

```
override func presentationAnimationDidFinish() {
  let items = extensionContext?.inputItems
  var itemProvider: NSItemProvider?

  // Exit early if there are no items
  if items == nil && items!.isEmpty != false { return }

  // Get item
  let item = items![0] as! NSExtensionItem
  print("Attachment items: \(String(describing: items))")

  // Loop through all the attachment items and store their values in properties
  //
  // Note: The App Store makes three attachements available: image, plain-text, and url
  //

  if let attachments = item.attachments {

    for attachment in attachments {
      itemProvider = attachment as? NSItemProvider

      // Get text
      let textType = kUTTypePlainText as String
      if itemProvider?.hasItemConformingToTypeIdentifier(textType) == true {
        itemProvider?.loadItem(forTypeIdentifier: textType, options: nil) {•••}
      }

      // Get URL
      let urlType = kUTTypeURL as String
      if itemProvider?.hasItemConformingToTypeIdentifier(urlType) == true {•••}

      // Get image
      let imageType = kUTTypeImage as String
      if itemProvider?.hasItemConformingToTypeIdentifier(imageType) == true {
        itemProvider?.loadItem(forTypeIdentifier: imageType, options: nil) {•••}
      }

    }
  }
```

SHARE EXTENSIONS

- Process the callback from javascript to extract the information
- Get info and save to property in view controller to access later

```
attachments = item.attachments {

ttachment in attachments {
mProvider = attachment as? NSItemProvider

Get text
 textType = kUTTypePlainText as String
itemProvider?.hasItemConformingToTypeIdentifier(textType) == true {
temProvider?.loadItem(forTypeIdentifier: textType, options: nil) { (item, error) -> Void in
 if error == nil {
   if let text = item as? String {
     self.textToShare = text
   }
 }


Get URL
 urlType = kUTTypeURL as String
itemProvider?.hasItemConformingToTypeIdentifier(urlType) == true {
temProvider?.loadItem(forTypeIdentifier: urlType, options: nil) { (item, error) -> Void in
 if error == nil {
   if let url = item as? URL {
     self.urlToShare = url.absoluteString
   }
 }


Get image
 imageType = kUTTypeImage as String
itemProvider?.hasItemConformingToTypeIdentifier(imageType) == true {
temProvider?.loadItem(forTypeIdentifier: imageType, options: nil) { (item, error) -> Void in
 if error == nil {
   print("Item: \(String(describing: item))")
   if let image = item as? UIImage {
     self.imageToShare = image
   }
 }
```

SHARE EXTENSIONS

```
override func didSelectPost() {
  // This is called after the user selects Post. Do the upload of
  // contentText and/or NSExtensionContext attachments.

  // While it is possible to save an UIImage to a dictionary as NSData as
  // we are doing here, it is generally not a good idea.  You could store
  // image on server or in App Group folder and keep the URL of the location
  // to store

  // Also, note that you get the app store URL, so you could use it to get
  // the image that way.
  let imageData: Data? = UIImagePNGRepresentation(imageToShare!)
  let sharedItem: NSDictionary = [ "date": Date(),
                                   "contentText": textToShare!,
                                   "imageData": imageData!,
                                   "url": urlToShare!
  ]

  LocalDefaultsManager.sharedInstance.add(object: sharedItem)


  // Inform the host that we're done, so it un-blocks its UI.
  // Note: Alternatively you could call super's -didSelectPost, which will similarly complete the extension context.
  self.extensionContext!.completeRequest(returningItems: [], completionHandler: nil)
}
```

USER TAPS POST; ADD TO USER DEFAULTS IN APP GROUP

SHARE EXTENSIONS

```
override func isContentValid() -> Bool {
  // Do validation of contentText and/or NSExtensionContext attachments here
  return true
}


override func configurationItems() -> [Any]! {
  // To add configuration options via table cells at the bottom of the sheet,
  // return an array of SLComposeSheetConfigurationItem here.
  return []
}
```

- Do custom validation or add more configuration items

PHOTO SHARING EXTENSION

- Customize the parsing of the NSItemProviders

```
/// View Controller that is presented via the share extension.  This uses the
/// default share sheet from the `Social.framework`.
class ShareViewController: SLComposeServiceViewController {

  override func didSelectPost() {
    // This is called after the user selects Post. Do the upload of
    // contentText and/or NSExtensionContext attachments.

    // While it is possible to save an UIImage to a dictionary as NSData as
    // we are doing here, it is generally not a good idea.  You could store
    // image on server or in App Group folder and keep the URL of the location
    // to store

    // Also, note that you get the app store URL, so you could use it to get
    // the image that way.

    if let item = self.extensionContext?.inputItems[0] as? NSExtensionItem {
      for ele in item.attachments!{
        let itemProvider = ele as! NSItemProvider

        if itemProvider.hasItemConformingToTypeIdentifier("public.jpeg"){
          NSLog("itemprovider: %@", itemProvider)
          itemProvider.loadItem(forTypeIdentifier: "public.jpeg", options: nil, completionHandler: { (item, error) in

            var imgData: Data!
            if let url = item as? URL {
              imgData = try! Data(contentsOf: url)
            }

            if let img = item as? UIImage {
              imgData = UIImagePNGRepresentation(img)
            }

            let sharedItem: NSDictionary = [ "date": Date(),
                                             "contentText": self.contentText,
                                             "imageData": imgData]

            LocalDefaultsManager.sharedInstance.add(object: sharedItem)
```

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

VISION WITH COREML

COREML

- Framework (iOS11) to integrate machine learning models into your app

COREML

SUBTITLE

- Core ML is the foundation for domain-specific frameworks and functionality
  - Vision for image analysis
  - Foundation for natural language processing
  - GameplayKit for evaluating learned decision trees

COREML

SUBTITLE

- Apple uses it
  - Siri
  - Camera
  - QuickType
  - Photos

COREML

SUBTITLE

- Core ML itself builds on top of low-level primitives
  - Accelerate
  - BNNS (Basic Neural Network Subroutines)
  - Metal Performance Shaders
- Performed on device

COREML

SUBTITLE

- Machine learning is a class of algorithms that make predictions base on training data
- Predictions can be refined without human intervention at run time

COREML

SUBTITLE

- Many different kinds of machine learning algorithms
  - Try to estimate probability of being in a state
  - Move through additional layers to refine
- Prediction and probability (confidence)

COREML

- Nueral network for image detection using Tensor Flow

COREML

SUBTITLE

- Apple publishes some of their machine learning research at https://machinelearning.apple.com

Apple Machine Learning Journal

Personalized Hey Siri

Vol. 1, Issue 9 • April 2018
by Siri Team

Apple introduced the “Hey Siri” feature with the iPhone 6 (iOS 8). This feature allows users to invoke Siri without having to press the home button. When a user says, “Hey Siri, how is the weather today?” the phone wakes up upon hearing “Hey Siri” and processes the rest of the utterance as a Siri request.

The feature’s ability to listen continuously for the “Hey Siri” trigger phrase lets users access Siri in situations where their hands might be otherwise occupied, such as while driving or cooking, as well as in situations when their respective devices are not within arm’s reach. Imagine a scenario where a user is asking his or her iPhone 6 on the kitchen counter to set a timer while putting a turkey into the oven.

Read the article ›

COREML

- ML workflow
- CoreML comes in after you have trained your model

COREML

SUBTITLE

- Apple provides popular modes that are drag-and-drop ready for using with CoreML
  - Object detection
  - Handwriting recognition
- Tools for converting your own models to work with Apple Frameworks

MLMODEL

Working with Models

Build your apps with the ready-to-use Core ML models below, or use Core ML Tools to easily convert models into the Core ML format.

Models

MobileNet

MobileNets are based on a streamlined architecture that have depth-wise separable convolutions to build lightweight, deep neural networks.

Detects the dominant objects present in an image from a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

View original model details

Download Core ML Model (17.1 MB)

Model Converters

Core ML Tools

Core ML Tools is a python package that can be used to convert models from machine learning toolboxes into the Core ML format.

Get Core ML Tools ↗

Apache MXNet

MXNet helps you train machine learning models and convert them into the Core ML format.

Get MXNet model converter ↗

COREML

SUBTITLE

- Some models are already

MLMODEL

Working with Models

Build your apps with the ready-to-use Core ML models below, or use Core ML Tools to easily convert models into the Core ML format.

Model Converters

Core ML Tools

Core ML Tools is a python package that can be used to convert models from machine learning toolboxes into the Core ML format.

Get Core ML Tools ↗

Apache MXNet

MXNet helps you train machine learning models and convert them into the Core ML format.

Get MXNet model converter ↗

USING CORMEL FOR OBJECT DETECTION

OBJECT DETECTION

- Choose a model
- Factors
  - Size
  - Training data
  - Methodology

Models

Places205-GoogLeNet

Detects the scene of an image from 205 categories such as an airport terminal, bedroom, forest, coast, and more.

View original model details ›

Download Core ML Model

File size: 24.8 MB

ResNet50

Detects the dominant objects present in an image from a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

View original model details ›

Download Core ML Model

File size: 102.6 MB

Inception v3

Detects the dominant objects present in an image from a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

View original model details ›

Download Core ML Model

File size: 94.7 MB

VGG16

Detects the dominant objects present in an image from a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

View original model details ›

Download Core ML Model

File size: 553.5 MB

OBJECT DETECTION

- 2018-session4-vision
  - 2018-session4-vision
    - AppDelegate.swift
    - ViewController.swift
    - Main.storyboard
    - Assets.xcassets
    - LaunchScreen.storyboard
    - MobileNet.mlmodel
    - Info.plist
  - Products

Recognized by Xcode

Machine Learning Model

Name	MobileNet
Type	Neural Network Classifier
Size	17.1 MB
Author	Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam
Description	MobileNets are based on a streamlined architecture that have depth-wise separable convolutions to build light weight deep neural networks. Trained on ImageNet with categories such as trees, animals, food, vehicles, person etc. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications https://github.com/shicai/MobileNet-Caffe
License	Apache License. Version 2.0 http://www.apache.org/licenses/LICENSE-2.0

Model Class

MobileNet

Automatically generated Swift model class

Model Evaluation Parameters

Name	Type	Description
Inputs		
image	Image (Color 224 x 224)	Input image to be classified
Outputs		
classLabelProbs	Dictionary (String → Double)	Probability of each category

OBJECT DETECTION

Model Class

MobileNet

Automatically generated Swift model class

Creates a class

Model Evaluation Parameters

Name	Type	Description
Inputs		
image	Image (Color 224 x 224)	Input image to be classified
Outputs		
classLabelProbs	Dictionary (String → Double)	Probability of each category
classLabel	String	Most likely image category

OBJECT DETECTION

- Choose an image from and prepare it for CoreML

```
override func viewDidLoad() {
  super.viewDidLoad()

  // The images in asset catalog
  let images = ["dog","swim","cat","pro","apple"]

  // Grab a random one
  let randomIndex = Int(arc4random_uniform(UInt32(images.count)))
  let image = UIImage(named: images[randomIndex] )
  imageView.image = image

  // Convert to CIImage (CoreImage)
  guard let ciImage = CIImage(image: image!) else {
    fatalError("Not able to convert UIImage to CIImage")
  }

  detectObjects(image: ciImage)
}
```

OBJECT DETECTION

- Set up vision request

```
func detectObjects(image: CIImage) {

  // Load the ML model through its generated class
  guard let model = try? VNCoreMLModel(for: MobileNet().model) else {
    fatalError("Can't load the ML model")
  }

  // Create a Vision request with completion handler
  let request = VNCoreMLRequest(model: model) { request, error in

    guard let results = request.results as? [VNClassificationObservation] else {
        fatalError("Unexpected result type from VNCoreMLRequest")
    }

    // Put the top 5 into a string to show on the screen
    var tags = ""
    for result in results[0...5] {
      tags += "🤔 \(result.identifier) (\(Int(result.confidence * 100))%)\n"
    }

    // Update UI on main queue
    DispatchQueue.main.async {
       self.textView.text = tags
    }
  }

  // Run the Core ML model classifier on global dispatch queue
  let handler = VNImageRequestHandler(ciImage: image)
  DispatchQueue.global(qos: .userInteractive).async {
    do {
      try handler.perform([request])
    } catch {
      print(error)
    }
  }
}
```

OBJECT DETECTION

```
// Load the ML model through its generated class
guard let model = try? VNCoreMLModel(for: MobileNet().model) else {
  fatalError("Can't load the ML model")
}

// Create a Vision request with completion handler
let request = VNCoreMLRequest(model: model) { request, error in

  guard let results = request.results as? [VNClassificationObservation] else {
      fatalError("Unexpected result type from VNCoreMLRequest")
  }

  // Put the top 5 into a string to show on the screen
  var tags = ""
  for result in results[0...5] {
    tags += "🤔 \(result.identifier) (\(Int(result.confidence * 100))%)\n"
  }

  // Update UI on main queue
  DispatchQueue.main.async {
```

Load model by name

OBJECT DETECTION

```
// Create a Vision request with completion handler
let request = VNCoreMLRequest(model: model) { request, error in

  guard let results = request.results as? [VNClassificationObservation] else {
      fatalError("Unexpected result type from VNCoreMLRequest")
  }

  // Put the top 5 into a string to show on the screen
  var tags = ""
  for result in results[0...5] {
    tags += "🤔 \(result.identifier) (\(Int(result.confidence * 100))%)\n"
  }

  // Update UI on main queue
  DispatchQueue.main.async {
     self.textView.text = tags
  }
}
```

Create the request with completion handler

OBJECT DETECTION

```
    // Update UI on main queue
    DispatchQueue.main.async {
        self.textView.text = tags
    }
  }

  // Run the Core ML model classifier on global dispatch queue
  let handler = VNImageRequestHandler(ciImage: image)
  DispatchQueue.global(qos: .userInteractive).async {
    do {
      try handler.perform([request])
    } catch {
      print(error)
    }
  }
}
```

Start it off

NUMBER RECOGNITION

NUMBER DETECTION

- MNIST database of handwritten digits
- 60,000 training images
- 10,0000 testing images

NUMBER DETECTION

- Recognize a hand drawn image

NUMBER DETECTION

NUMBER DETECTION

Create Model

```
MLModel *ml_model = [[[keras_mnist_cnn alloc] init] model];
VNCoreMLModel *vnc_core_ml_model = [VNCoreMLModel modelForMLModel: ml_model error:nil];

VNCoreMLRequest *request = [[VNCoreMLRequest alloc] initWithModel: vnc_core_ml_model completionHandler:
    (VNRequestCompletionHandler) ^(VNRequest *request, NSError *error){
    NSArray *results = [request.results copy];

    VNClassificationObservation *res = ((VNClassificationObservation *)(results[0]));

    self.resultLabel.text = [NSString stringWithFormat: @"Digit may be: %@", res.identifier];
}];

NSDictionary *options_dict = [[NSDictionary alloc] init];
NSArray *request_array = @[request];

VNImageRequestHandler *handler = [[VNImageRequestHandler alloc] initWithCIImage:self.imageToDetect options:options_dict];
dispatch_queue_t myCustomQueue;
myCustomQueue = dispatch_queue_create("com.lukau.VNImageRequestHandlerQueue", NULL);

self.resultLabel.text = @"Predicting...";
dispatch_sync(myCustomQueue, ^{
    [handler performRequests:request_array error:nil];

});
```

NUMBER DETECTION

```
MLModel *ml_model = [[[keras_mnist_cnn alloc] init] model];
VNCoreMLModel *vnc_core_ml_model = [VNCoreMLModel modelForMLModel: ml_model error:nil];

VNCoreMLRequest *request = [[VNCoreMLRequest alloc] initWithModel: vnc_core_ml_model completionHandler:
    (VNRequestCompletionHandler) ^(VNRequest *request, NSError *error){
    NSArray *results = [request.results copy];

    VNClassificationObservation *res = ((VNClassificationObservation *)(results[0]));

    self.resultLabel.text = [NSString stringWithFormat: @"Digit may be: %@", res.identifier];
}];

NSDictionary *options_dict = [[NSDictionary alloc] init];
NSArray *request_array = @[request];

VNImageRequestHandler *handler = [[VNImageRequestHandler alloc] initWithCIImage:self.imageToDetect options:options_dict];
dispatch_queue_t myCustomQueue;
myCustomQueue = dispatch_queue_create("com.lukau.VNImageRequestHandlerQueue", NULL);

self.resultLabel.text = @"Predicting...";
dispatch_sync(myCustomQueue, ^{
    [handler performRequests:request_array error:nil];

});
```

Setup Vision Request

NUMBER DETECTION

```
MLModel *ml_model = [[[keras_mnist_cnn alloc] init] model];
VNCoreMLModel *vnc_core_ml_model = [VNCoreMLModel modelForMLModel: ml_model error:nil];

VNCoreMLRequest *request = [[VNCoreMLRequest alloc] initWithModel: vnc_core_ml_model completionHandler:
    (VNRequestCompletionHandler) ^(VNRequest *request, NSError *error){
    NSArray *results = [request.results copy];

    VNClassificationObservation *res = ((VNClassificationObservation *)(results[0]));

    self.resultLabel.text = [NSString stringWithFormat: @"Digit may be: %@", res.identifier];
}];

NSDictionary *options_dict = [[NSDictionary alloc] init];
NSArray *request_array = @[request];

VNImageRequestHandler *handler = [[VNImageRequestHandler alloc] initWithCIImage:self.imageToDetect options:options_dict];
dispatch_queue_t myCustomQueue;
myCustomQueue = dispatch_queue_create("com.lukau.VNImageRequestHandlerQueue", NULL);

self.resultLabel.text = @"Predicting...";
dispatch_sync(myCustomQueue, ^{
    [handler performRequests:request_array error:nil];

});
```

Start the Request

NUMBER DETECTION

- Demo

NUMBER DETECTION

Model trained on 28x28 pixels

```
// Original image
self.imageToDetect = [[CIImage alloc]initWithImage:self.drawingCanvas.image];

// You may get better performance by rescaling to match the database image size
UIImage *scaledCanvasImage = [self imageWithImage:self.drawingCanvas.image scaledToSize:CGSizeMake(28, 28)];
self.imageToDetect = [[CIImage alloc]initWithImage:scaledCanvasImage];
```

- You may get better performance by rescaling input size to training size

NUMBER DETECTION

Handwritten digit recognition with MNIST on iOS with Keras

Or: How to use a Keras deep learning neural network in iOS?

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

3D TOUCH

3D TOUCH

- iOS 9+, new iPhone models add a third dimension to the user interface
  - A user can now press your Home screen icon to immediately access functionality provided by your app
  - Within your app, a user can now press views to see previews of additional content and gain accelerated access to features

3D TOUCH

APIS

- The Home screen quick action API is for adding shortcuts to your app icon that anticipate and accelerate a user's interaction with your app
  - Sometimes at the expense of opening your application

3D TOUCH

APIS

- The Web view peek and pop API lets you enable system-mediated previews of HTML link destinations

3D TOUCH

APIS

- Peek and Pop
  - UIKit
  - Web view peek and pop API lets you enable system-mediated previews of HTML link destinations

3D TOUCH

APIS

- UITouch force properties let you add customized force-based user interaction to your app

3D TOUCH

APIS

- No matter which of these APIs you adopt, your app must check the availability of 3D Touch at runtime
  - iOS6+ and higher
  - iPad Pro

HOME SCREEN

HOME SCREEN

- Support two types of quick actions (static and dynamic)
- Static quick actions
  - Available to the user immediately upon app installation
  - Define Home screen static quick actions in your app's Info.plist file in the UIApplicationShortcutItems array

HOME SCREEN

- Info.plist
- Assets.xcassets
- Base.lproj
- Products

Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.2
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
UIApplicationShortcutItems	Array	(2 items)
Item 0	Dictionary	(5 items)
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeSearch
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1
UIApplicationShortcutItemTitle	String	shortcutTitle1
UIApplicationShortcutItemType	String	$(PRODUCT_BUNDLE_IDENTIFIER).First
UIApplicationShortcutItemUserI...	Dictionary	(1 item)
Item 1	Dictionary	(5 items)
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2
UIApplicationShortcutItemTitle	String	shortcutTitle2
UIApplicationShortcutItemType	String	$(PRODUCT_BUNDLE_IDENTIFIER).Second
UIApplicationShortcutItemUserI...	Dictionary	(1 item)
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Status bar tinting parameters	Dictionary	(1 item)
Supported interface orientations	Array	(3 items)

HOME SCREEN

▼ Item 0	Dictionary	(5 items)
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIcon
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle1
UIApplicationShortcutItemTitle	String	shortcutTitle1
UIApplicationShortcutItemType	String	$(PRODUCT_BUNDLE_IDENTIFIER).First
▼ UIApplicationShortcutItemUserI...	Dictionary	(1 item)
firstShorcutKey1	String	firstShortcutKeyValu
▼ Item 1	Dictionary	(5 items)
UIApplicationShortcutItemIconT...	String	UIApplicationShortcutIconTypeShare
UIApplicationShortcutItemSubtit...	String	shortcutSubtitle2
UIApplicationShortcutItemTitle	String	shortcutTitle2
UIApplicationShortcutItemType	String	$(PRODUCT_BUNDLE_IDENTIFIER).Second
▼ UIApplicationShortcutItemUserI...	Dictionary	(1 item)

CAN ALSO USE CUSTOM IMAGE

DATA DICT TO PASS

- Appearance properties and identifiers

HOME SCREEN

```
/*
    Called when the user activates your application by selecting a shortcut on the home screen, except when
    application(_:,willFinishLaunchingWithOptions:) or application(_:didFinishLaunchingWithOptions) returns
        `false`.
    You should handle the shortcut in those callbacks and return `false` if possible. In that case, this
    callback is used if your application is already launched in the background.
*/

func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem,
    completionHandler: @escaping (Bool) -> Void) {
    let handledShortCutItem = handleShortCutItem(shortcutItem)

    completionHandler(handledShortCutItem)
}
```

- Handle the shortcut item by parsing the `shortcutItem.type` to identify

UIApplicationShortcutItemType	String	$(PRODUCT_BUNDLE_IDENTIFIER).Second

HOME SCREEN

- Custom logic to handle the shortcut
- You have to take it from here

```
leShortCutItem(_ shortcutItem: UIApplicationShortcutItem) -> Bool {
andled = false

rify that the provided `shortcutItem`'s `type` is one handled by the application.
 ShortcutIdentifier(fullType: shortcutItem.type) != nil else { return false }

 let shortCutType = shortcutItem.type as String? else { return false }

n (shortCutType) {
ase ShortcutIdentifier.First.type:
   // Handle shortcut 1 (static).
   handled = true
   break
ase ShortcutIdentifier.Second.type:
   // Handle shortcut 2 (static).
   handled = true
   break
ase ShortcutIdentifier.Third.type:
   // Handle shortcut 3 (dynamic).
   handled = true
   break
ase ShortcutIdentifier.Fourth.type:
   // Handle shortcut 4 (dynamic).
   handled = true
   break
efault:
   break


nstruct an alert using the details of the shortcut used to open the application.
lertController = UIAlertController(title: "Shortcut Handled", message: "\"\(shortcutItem.loca
'", preferredStyle: .alert)
kAction = UIAlertAction(title: "OK", style: .default, handler: nil)
Controller.addAction(okAction)

splay an alert indicating the shortcut selected from the home screen.
```

HOME SCREEN

SUBTITLE

- Dynamic quick actions
  - Available to the user after first launch
  - Define Home screen dynamic quick actions with the UIApplicationShortcutItem, UIMutableApplicationShortcutItem, and UIApplicationShortcutIcon classes
  - Add dynamic quick actions to your app's shared UIApplication object using the shortcutItems property

Class

UIApplicationShortcutItem

An application shortcut item, also called a Home screen dynamic quick action, specifies a user-initiated action for your app.

Overview

On a device that supports 3D Touch, a user invokes the quick action by pressing your app's icon on the Home screen and then selecting the quick action's title. Your app delegate receives and handles the quick action.

You must specify the characteristics of your `UIApplicationShortcutItem` instances during initialization, before you register them with your app object. The quick actions you've registered with your app object are immutable.

Registering an Array of Dynamic Quick Actions With Your App

To register an array of Home screen dynamic quick actions, set the value of your shared app object's `shortcutItems` property with an `NSArray` instance containing your defined dynamic Home screen quick actions.

Changing Your App's Dynamic Quick Actions

To change your app's Home screen dynamic quick actions, replace your app object's `shortcutItems` array by setting a new value for the property. As a convenience for working with registered quick actions, this class has a mutable subclass, `UIMutableApplicationShortcutItem`. The following code snippet illustrates one way to use the `mutableCopy()` method, along with mutable quick actions, to change the title of a dynamic Home screen quick action:

HOME SCREEN

```
let shortcut = UIApplicationShortcutItem(type: "mobi.uchicago.shortcuts.dynamic",
                                         localizedTitle: "😀Shortcut Title",
                                         localizedSubtitle: "Dynamic",
                                         icon: UIApplicationShortcutIcon(type: .add),
                                         userInfo: nil)
UIApplication.shared.shortcutItems = [shortcut]
```

- Create and update the shortcut items

HOME SCREEN

```
// Handle the application shortcuts from the home screen
func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem,
    completionHandler: @escaping (Bool) -> Void) {
  print(shortcutItem)
  completionHandler(true)
}
```

- Static and dynamic shortcuts are handled the same way
- The `userInfo` dictionary/name should define the logic

HOME SCREEN

APPLICATION SHORTCUT PROJECT

ApplicationShortcuts 〉 iPhone 7 | Finished running AppShortcuts on iPhone 7 | 2

ApplicationShortcuts 〉 ApplicationShortcuts 〉 AppDelegate.swift 〉 application(_:didFinishLaunchingWithOptions:)

- ApplicationShortcuts
  - README.md
  - ApplicationShortcuts
    - AppDelegate.swift
    - ShortcutsTableViewController.swift
    - ShortcutDetailViewController.swift
    - Info.plist
    - Assets.xcassets
    - Base.lproj
  - Products

```
                handled = true
                break
            case ShortcutIdentifier.Fourth.type:
                // Handle shortcut 4 (dynamic).
                handled = true
                break
            default:
                break
        }

        // Construct an alert using the details of the shortcut used to open the application.
        let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\(shortcutItem.localizedTitle)
            \"", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
        alertController.addAction(okAction)

        // Display an alert indicating the shortcut selected from the home screen.
        window!.rootViewController?.present(alertController, animated: true, completion: nil)

        return handled
    }

    // MARK: - Application Life Cycle

    func applicationDidBecomeActive(_ application: UIApplication) {...}

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        // Override point for customization after application launch.
        var shouldPerformAdditionalDelegateHandling = true

        // If a shortcut was launched, display its information and take the appropriate action
        if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem
            {

            launchedShortcutItem = shortcutItem

            // This will block "performActionForShortcutItem:completionHandler" from being called.
            shouldPerformAdditionalDelegateHandling = false
        }
```

Identity and Type

Name: AppDelegate.swift

Type: Default - Swift Source

Location: Relative to Project

ApplicationShortcuts/AppDelegate.swift

Full Path: /Users/tabinkowski/Downloads/ApplicationShortcutsUsingUIApplicationShortcutItem/ApplicationShortcuts/AppDelegate.swift

On Demand Resource Tags

Only resources are taggable

Target Membership

☑ ApplicationShortcuts

Text Settings

Text Encoding: Unicode (UTF-8)

Line Endings: Default - macOS / Unix (LF)

Indent Using: Spaces

Widths: 2 (Tab) 2 (Indent)

☑ Wrap lines

HOME SCREEN

DEMO: EXTEND THIS APP SHOW SIMPLE

- 2017-ExtendThisApp.entitlements A
- AppDelegate.swift ?
- ViewController.swift ?
- Main.storyboard
- ForceViewController.swift A
- Assets.xcassets ?
- LaunchScreen.storyboard
- Info.plist ?
- DataKit
  - DataKitManager.swift ?
  - Constants.swift A
  - CoreDataManager.swift A
  - DataKit.h ?
  - Info.plist ?
  - Model.xcdatamodeld M
- Products
  - 2017-ExtendThisApp.app
  - DataKit.framework

```swift
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) ->
      Bool {
    print(NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).last as Any);

    // Print out the app group (for fun)
    let sharedAppGroup: String = "group.2017-extend-this-app"
    let directory: NSURL = FileManager.default.containerURL(forSecurityApplicationGroupIdentifier: sharedAppGroup)! as NSURL
    print("App Group URL: \(directory)")


    return true
  }

  func applicationWillResignActive(_ application: UIApplication) {
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary
        interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition
        to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to
        pause the game.
  }

  func applicationDidEnterBackground(_ application: UIApplication) {
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to
        restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
  }

  func applicationWillEnterForeground(_ application: UIApplication) {
    // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering
        the background.
  }

  func applicationDidBecomeActive(_ application: UIApplication) {
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the
        background, optionally refresh the user interface.
  }

  func applicationWillTerminate(_ application: UIApplication) {
    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
  }



  // Handle the application shortcuts from the home screen
  func application(_ application: UIApplication, performActionFor shortcutItem: UIApplicationShortcutItem, completionHandler: @escaping
```

No Quick Help

Search Documentation

search bar connected to a search dis...

Fixed Space Bar Button Item - Represents a fixed space item on a UIToolbar object.

Flexible Space Bar Button Item - Represents a flexible space item on a UIToolbar object.

View - Represents a rectangular region in which it draws and receives events

HOME SCREEN

DEMO: APPLICATION SHORTCUT PROJECT

ApplicationShortcuts 〉 iPhone 7 | Finished running AppShortcuts on iPhone 7 | 2

ApplicationShortcuts 〉 ApplicationShortcuts 〉 AppDelegate.swift 〉 application(_:didFinishLaunchingWithOptions:)

- ApplicationShortcuts
  - README.md
  - ApplicationShortcuts
    - AppDelegate.swift
    - ShortcutsTableViewController.swift
    - ShortcutDetailViewController.swift
    - Info.plist
    - Assets.xcassets
    - Base.lproj
  - Products

```
                handled = true
                break
            case ShortcutIdentifier.Fourth.type:
                // Handle shortcut 4 (dynamic).
                handled = true
                break
            default:
                break
        }

        // Construct an alert using the details of the shortcut used to open the application.
        let alertController = UIAlertController(title: "Shortcut Handled", message: "\"\(shortcutItem.localizedTitle)
            \"", preferredStyle: .alert)
        let okAction = UIAlertAction(title: "OK", style: .default, handler: nil)
        alertController.addAction(okAction)

        // Display an alert indicating the shortcut selected from the home screen.
        window!.rootViewController?.present(alertController, animated: true, completion: nil)

        return handled
    }

    // MARK: - Application Life Cycle

    func applicationDidBecomeActive(_ application: UIApplication) {...}

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        // Override point for customization after application launch.
        var shouldPerformAdditionalDelegateHandling = true

        // If a shortcut was launched, display its information and take the appropriate action
        if let shortcutItem = launchOptions?[UIApplicationLaunchOptionsKey.shortcutItem] as? UIApplicationShortcutItem
            {

            launchedShortcutItem = shortcutItem

            // This will block "performActionForShortcutItem:completionHandler" from being called.
            shouldPerformAdditionalDelegateHandling = false
        }
```

Identity and Type

Name: AppDelegate.swift

Type: Default - Swift Source

Location: Relative to Project

ApplicationShortcuts/AppDelegate.swift

Full Path: /Users/tabinkowski/Downloads/ApplicationShortcutsUsingUIApplicationShortcutItem/ApplicationShortcuts/AppDelegate.swift

On Demand Resource Tags

Only resources are taggable

Target Membership

ApplicationShortcuts

Text Settings

Text Encoding: Unicode (UTF-8)

Line Endings: Default - macOS / Unix (LF)

Indent Using: Spaces

Widths: 2 Tab, 2 Indent

Wrap lines

PEEK AND POP IN STORYBOARD

PEEK AND POP

- Peek
  - Preview of the content
  - "preview"
- Pop
  - Go to the content
  - "commit"

PEEK AND POP

- Indication of peek availability
  - With a light press, surrounding content blurs to tell the user a preview of additional content —the peek—is available

PEEK AND POP

- Peek
  - Press a bit more deeply and the view transitions to show the peek
  - If the user ends the touch at this point, the peek disappears and the app returns to its state before the interaction started

PEEK AND POP

- Pop
  - The user can press deeper still on the peek itself to navigate, using the system-provided pop transition, to the view being previewed as a peek
  - The pop view then fills your app's root view and displays a button to navigate back to where the interaction began

PEEK AND POP

- Peek quick actions
  - If the user swipes the peek upward, the system shows the peek quick actions
  - Each peek quick action is a deep link into your app
  - 🤷‍♀️

PEEK AND POP

SUBTITLE

- Implement Peek and Pop programmatically or in Storyboard

PEEK AND POP

DEMO: EXTEND THIS APP

FORCE PROPERTIES

FORCE PROPERTIES

FORCE PROPERTIES

SUBTITLE

- UITouch class has two new properties to support custom implementation of 3D Touch in your app
  - force
  - maximumPossibleForce
- Detect and respond to touch pressure in the UIEvent objects your app receives

FORCE PROPERTIES

- 😈?

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

ASSIGNMENT 3

ASSIGNMENT 3

- Picture Journal App
  - Users select a picture and add a caption
  - Meta data generated from picture
  - Share and Today Extensions
  - Sync using CloudKit subscriptions

ASSIGNMENT 3

Day One

ASSIGNMENT 3

- Day One
  - Apple Design Award winner
  - Success as a paid app (for a while)
  - Moved to subscription model
  - Move from using Dropbox flat files to iCloud Drive to custom sync solution

ASSIGNMENT 3

OVERVIEW

CLOUDKIT

ASSIGNMENT 3

HOME SCREEN

- Home screen is list
  - TableView (or CollectionView) should be segmented by month
  - Cell should display thumbnail, text blurb, and date (can include more if like)
  - Search bar should search by text and tags

ASSIGNMENT 3

HOME SCREEN

- Map screen will show locations from entries
  - Tapping on entry will show all entries in that location via location based query
  - Show back on main home screen

ASSIGNMENT 3

ADD/EDIT SCREEN

- User is in app and selects add a new entry
  - Entry has a photo and caption text, tags (user generated)
  - Meta-data is automatically generated
    - GPS, Date: From Photos framework
    - Weather: https://darksky.net/dev
    - Tags: Vision, NLP framework
  - Stored as a custom struct and serialized to disk as JSON
  - Everything can be edited (including date)

ASSIGNMENT 3

SETTINGS SCREEN

- Settings screen
  - Show iCloud user name
  - Allow user Sync (reset local cache); provide good feedback
  - Set a passcode
  - Bonus Point (on this day notification)

CLOUDKIT
DONE
JOHNNY APPLESEED
SYNC
PRIVACY
ON THIS DAY
SEARCH
today...
MARCH 18, 2018
This happened today...
MARCH 31, 2018
This happened today...
February 2018
FEBRUARY 28, 2018
This happened today...
EDIT
MARCH 31, 2018
Today the kids had an epic light saber battle. We discussed how hitting people with their back is turned is a bad thing.
KIDS, STAR WARS, PRINCESS

ASSIGNMENT 3

FUNCTIONALITY

ASSIGNMENT 3

- Sync is a hard problem
  - Local caching
  - Mirroring
  - Platform services (Firebase, Realm, not CloudKit)
- We are going to use a simple and free strategy using CloudKit public database

ASSIGNMENT 3

ASSIGNMENT 3

On app launch

- Fetch changes
- Subscribe to future changes

On push from CloudKit

- Fetch changes

ASSIGNMENT 3

OFFLINE-FIRST APPROACH

- Local change
  - Save to local data
  - Mark `synced = false`
  - Update UI (for user's sake)
  - Push to CloudKit
  - On success `synced = true`
  - Save the cloudKit `recordId` to local object

If not successful try again or put in queue to try later

ASSIGNMENT 3

OFFLINE-FIRST APPROACH

- Remote change received via subscription
  - Find local object that matches (recordId)
  - Mark `synced = false`
  - Update local object to match changes
  - Update UI (for user's sake)
  - On success `synced = true`

If not successful try again or put in queue to try later

ASSIGNMENT 3

SYNC FLOW

- Assumptions that make our simple sync approach plausible
  - Journal is single user (not many potential conflicts)
  - Sync data is just text (don't request images until we need them)
  - We can restore the database in a reasonable amount of time
  - Probably not that much data (~1 entry a day)

ASSIGNMENT 3

IMAGES

- Cache images
  - NSCache or Cache directory
  - They will be on iCloud if needed
- Ideas for performance
  - Resize images
  - Create thumbnail cache for table

Only request assets when you need them and then always cache

HTTPS://BLOG.FROZENFIRESTUDIOS.COM/WORKING-WITH-IMAGES-IN-CLOUDKIT-1E3579C67558

RECOMMENDED WORK FOR THIS WEEK

- Entry input, persist, and edit
  - Select photo(s); metadata extraction
  - Input text
  - Input tags
- Sync with CloudKit
  - Subscriptions
  - Dirty flag
- Notifications
- TableView screen
  - Searchbar with query
  - Section headers
  - Map point to search

GOING OVER NEXT WEEK

- Extensions
  - Today
  - Share
- Embedded Frameworks
- App Groups

GOING OVER NEXT WEEK

- 3D Touch
- Vision Framework

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 4

© 2020 T.A. Binkowski

CUSTOM NOTIFICATIONS FROM SUBSCRIPTIONS

CUSTOM NOTIFICATIONS

- Generic notifications from subscriptions
- This may not be the great user experience we want

CUSTOM NOTIFICATIONS

- Generic notifications from subscriptions
- This may not be the great user experience we want

CUSTOM NOTIFICATIONS

CUSTOM NOTIFICATIONS

Record Types

System Types

Users

Custom Types

alert

joke

joke_of_the_day

New Type

Delete Type

Field Name	Field Type	Indexes
System Fields		
recordName	Reference	None
createdBy	Reference	None
createdAt	Date/Time	None
modifiedBy	Reference	None
modifiedAt	Date/Time	None
changeTag	String	None
Custom Fields		
message	String	None

Add Field

Edit Indexes

CUSTOM NOTIFICATIONS

```
func registerSilentAlertSubscription() {
  let uuid: UUID = UUID()
  let identifier = "\(uuid)-alert"


  // Create the notification that will be delivered
  let notificationInfo = CKSubscription.NotificationInfo()
  notificationInfo.shouldSendContentAvailable = true
  notificationInfo.desiredKeys = ["message"]

  // Create the subscription
  let subscription = CKQuerySubscription(recordType: "alert",
                                         predicate: NSPredicate(value: true),
                                         subscriptionID: identifier,
                                         options: [CKQuerySubscription.Options.firesOnRecordCreation])

  subscription.notificationInfo = notificationInfo
  CKContainer.default().publicCloudDatabase.save(subscription,
                                                 completionHandler: ({returnRecord, error in
                                                  if let err = error {
                                                    print("ALERT: subscription failed \(err.localizedDescription)")
                                                  } else {
                                                    print("ALERT: subscription set up")
```

MESSAGE IS PASSED ALL THE WAY THROUGH TO THE IOS APP

CUSTOM NOTIFICATIONS

CLOUDKIT HELPER

CUSTOM NOTIFICATIONS

- alert.py
- Prompt user for a message and then adds a new record to the public database
  - Triggers the alert

```python
import sys
import json

import cloudkit_helper as ck

def add_alert(message):
    """Create a new alert with a custom message"""

    data = {
        'operations': [{
            'operationType': 'create',
            'record': {
                'recordType': 'alert',
                'fields': {
                    'message': {
                        'value': message,
                    }
                }
            }
        }]
    }

    print('Posting operation to create quote...')
    result = ck.cloudkit_request(
        '/development/public/records/modify', json.dumps(data))
    print(result['content'])


def main():
    message = input("What would you like to say to your users?\n")
    add_alert(message)


if __name__ == '__main__':
    main()
```

CUSTOM NOTIFICATIONS

```python
def add_alert(message):
    """Create a new alert with a custom message"""

    data = {
        'operations': [{
            'operationType': 'create',
            'record': {
                'recordType': 'alert',
                'fields': {
                    'message': {
                        'value': message,
                    }
                }
            }
        }]
    }

    print('Posting operation to create quote...')
    result = ck.cloudkit_request(
        '/development/public/records/modify', json.dumps(data))
    print(result['content'])
```

CUSTOM NOTIFICATIONS

```
12                'record': {
13                    'recordType': 'alert',
14                    'fields': {
15                        'message': {
16                            'value': message,
17                        }
18                    }
19                }
20            }]
21        }
22
23        print('Posting operation to create quote...')
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
3-2
What would you like to say to your users?
Buy more in app purchases...or else
```

CUSTOM NOTIFICATIONS

```
What would you like to say to your users?
🤩🤩🤩🥰🎱🥦You can even send emojis!!!!!!!
Posting operation to create quote...
URL:  https://api.apple-cloudkit.com/database/1/iCloud.mobi.uchicago.twenty-ni
Data:  {"operations": [{"operationType": "create", "record": {"recordType": "a
83d\ude04\ud83d\udc3d\ud83d\udc27\ud83d\udc25\ud83e\udd4e\ud83c\udfb1\ud83e\ud
Headers:  {'X-Apple-CloudKit-Request-KeyID': '99e04a281f141f4bce822833bd8e8782
1Date': '2019-11-12T19:29:27Z', 'X-Apple-CloudKit-Request-SignatureV1': 'MEYCI
BlmlJE6fNLf2UmpUZQsVC9s+'}
Params:  None
{
  "records" : [ {
    "recordName" : "0B361BB6-5E9D-4D10-8531-F607A96C480B",
    "recordType" : "alert",
    "fields" : {
      "message" : {
        "value" : "\uD83D\uDE00\uD83D\uDE03\uD83D\uDE04\uD83D\uDC3D\uD83D\uDC27\uD83D\uDC25\uD83E\uDD4E\uD83C\uDFB1\uD83E\uDE80 You can even send
  mojis!!!!!!!",
        "type" : "STRING"
      }
    },
    "pluginFields" : { },
    "recordChangeTag" : "k2w91wdk",
    "created" : {
      "timestamp" : 1573586969406,
      "userRecordName" : "_8a76cb86f6390ecf95f548796a270cc8",
```

EMOJIS SURVIVE

CUSTOM NOTIFICATIONS

- Create an amazing form that takes some text for the custom message

```
def get(self):
    self.response.headers['Content-Type'] = 'text/html'
    html = """
    <form action="/alerts/" method="post">
    <b>Alert Notifcation Message</b><br>
    <input type="text" name="message" value=""><br>
    <input type="submit" value="Submit">
    </form>
    """
    self.response.write(html)
```

IOS APP

CUSTOM NOTIFICATIONS

- Handle the remote notification
  - Read the APNS data
  - Extract the message
  - Create local notification

```
/// Called for push notifications
/// In this case, we are getting the changed record from cloudkit and then creating a new notification
func application(_ application: UIApplication,
                 didReceiveRemoteNotification userInfo: [AnyHashable : Any],
                 fetchCompletionHandler completionHandler: @escaping (UIBackgr

  // Get the APS notification data
  print(userInfo)
  let aps = userInfo["aps"] as! [String: AnyObject]

  // Do Something with the content available data
  let contentAvailable = aps["content-available"] as! Int
  if contentAvailable == 1 {
    print("APS: \(aps)")
    let cloudKitInfo = userInfo["ck"] as! [String: AnyObject]
    let recordId = cloudKitInfo["qry"]?["rid"] as! String
    let field = cloudKitInfo["qry"]?["af"] as! [String: AnyObject]

    // A message indicates that we have a received a silent notification
    if let message = field["message"] as? String {
      local_notification_from_silent(message: message, recordId: recordId)
    }
    completionHandler(.newData)

  } else {
    completionHandler(.noData)
  }
}
```

CUSTOM NOTIFICATIONS

SILENT NOTIFICATION CONTENT

```
[AnyHashable("aps"): {
    "content-available" = 1;
}, AnyHashable("ck"): {
    ce = 2;
    cid = "iCloud.cloud.uchicago.Cloudy[illegible]AChanceOfErros";
    nid = "2b661f39-5617-4a1c-a88[illegible]09cd9eae7a";
    qry =     {
        af =         {
            message = "Where are my emojis?";
        };
        dbs = 2;
        fo = 1;
        rid = "A0915CF5-E4E7-4C76-856E-48CE01ADCACE";
        sid = "14725FE6-9C8F-4BDC-96FF-3815E5FE1E3B-alert";
        zid = "_defaultZone";
        zoid = "_defaultOwner";
    };
}]
```

FIELD I REQUESTED

RECORD THAT MATCHED THE QUERY

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {
  print("Creating local notification")

  // Create notification content
  let content = UNMutableNotificationContent()
  content.title = "😂 Joke of the Day"
  content.subtitle = "Urgent Message"
  content.body = message
  content.sound = UNNotificationSound.default

  // Set up trigger
  let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1,repeats: false)

  // Create the notification request
  let center = UNUserNotificationCenter.current()
  let identifier = recordId
  let request = UNNotificationRequest(identifier: identifier,
                                      content: content, trigger: trigger)
  center.add(request, withCompletionHandler: { (error) in
    if let error = error {
      print("Something went wrong: \(error)")
    }
  })
```

CREATE A NOTIFICATION

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {
  print("Creating local notification")

  // Create notification content
  let content = UNMutableNotificationContent()
  content.title = "😂 Joke of the Day"
  content.subtitle = "Urgent Message"
  content.body = message
  content.sound = UNNotificationSound.default

  // Set up trigger
  let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1,repeats: false)

  // Create the notification request
  let center = UNUserNotificationCenter.current()
  let identifier = recordId
  let request = UNNotificationRequest(identifier: identifier,
                                      content: content, trigger: trigger)
  center.add(request, withCompletionHandler: { (error) in
    if let error = error {
      print("Something went wrong: \(error)")
    }
  })
```

CREATE A TRIGGER

TIME, 1 SEC

CUSTOM NOTIFICATIONS

```
func local_notification_from_silent(message: String, recordId: String) {
  print("Creating local notification")

  // Create notification content
  let content = UNMutableNotificationContent()
  content.title = "😂 Joke of the Day"
  content.subtitle = "Urgent Message"
  content.body = message
  content.sound = UNNotificationSound.default

  // Set up trigger
  let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 1,repeats: false)

  // Create the notification request
  let center = UNUserNotificationCenter.current()
  let identifier = recordId
  let request = UNNotificationRequest(identifier: identifier,
                                      content: content, trigger: trigger)
  center.add(request, withCompletionHandler: { (error) in
    if let error = error {
      print("Something went wrong: \(error)")
    }
  })
```

ADD TO NOTIFICATION CENTER

REVIEW

NOTIFICATIONS

NOTIFICATIONS

NOTIFICATIONS

No more interruptions
Notifications appear at the top of your screen and disappear quickly.

One-swipe access
Swipe down to reveal Notification Center.

"User Notification" not broadcast notifications

See more at a glance
View stocks and weather, too.

NOTIFICATIONS

- Only one application can run in the foreground
- Notifications give you flexibility in how to interact with users when they're not running your app
  - Messages
  - Game turn
  - Updates available

NOTIFICATIONS

- Re-engage your users
  - Notifications can launch your app

NOTIFICATIONS

- Notification styles
  - Modal
  - Alert
  - Sounds (default, custom)
  - Badges
  - Silent
    - Happens in background
    - “Trigger event”

NOTIFICATIONS

- Local Notifications
  - Scheduled by an application
  - Delivered by the iOS on the device
  - “Schedule a notification”

NOTIFICATIONS

- Push Notifications
  - Remote notifications
  - Sent by application's remote server to Apple Push Notification Service (APNS)
  - "Register a notification"

NOTIFICATIONS

NOTIFICATIONS

Local

Push

Appears the same on devices

NOTIFICATIONS

- Handling notifications in app delegate
  - application:didFinishLaunchingWithOptions:
  - application:didReceiveRemoteNotification:
  - application:didReceiveLocalNotification:

Options: local or push
UIApplicationLaunchOptionsLocalNotificationKey

NOTIFICATIONS

- iOS8 introduced interactive notifications
  - Engage with application without opening it

NOTIFICATIONS

- iOS8 introduced interactive notifications
  - Engage with application without opening it
- iOS9 allows text input (eg. messages) from interactive notifications
- iOS10 unified the API for working with UserNotification Framework

Permissions

PERMISSIONS

- Remember that users are always in control
  - Keep them informed
  - Do not bother them
    - Of course you are trying to get their permission to bombard them with notifications)
  - Make sure you app still functions without notifications
    - Inform the user of consequences
    - “To be informed on the latest info, enable notifications in Settings” don’t forget to show a link to settings

PERMISSIONS

```
let options: UNAuthorizationOptions = [.alert, .sound];
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: options) {
    (granted, error) in
    if !granted {
        print("Something went wrong")
    }
}
```

- Registering for notifications

PERMISSIONS

Standard Dialogue

PERMISSIONS

- Consideration on notifications
  - Users may not accept to receive notifications
  - Users can change preferences in settings
- Be prepared to handle all potential use cases

```
center.getNotificationSettings { (settings) in
    if settings.authorizationStatus != .authorized {
        // Notifications not allowed
    }
}
```

PERMISSIONS

- The old way
  - “Go to Settings > Privacy > Location > OUR_APP”

PERMISSIONS

PERMISSIONS

```
// Check if we have authorization to send notifications
guard let settings = UIApplication.sharedApplication().currentUserNotificationSettings() else { return }

// If we don't, let the use know and give them the option of going directly
// to the Settings screen for the application
if settings.types == .None {

  // Create an alert view controller
  let alertController = UIAlertController(title: "😒",
                                          message: "The notification permission was not authorized. Please enable it in Sett
                                          preferredStyle: .Alert)
  // Create to go to settings
  let settingsAction = UIAlertAction(title: "Settings", style: .Default) { (alertAction) in
    if let appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
      UIApplication.sharedApplication().openURL(appSettings)
    }
  }
  alertController.addAction(settingsAction)

  // Create cancel action that does nothing
  let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)
  alertController.addAction(cancelAction)

  // Show the alert and exit early
  presentViewController(alertController, animated: true, completion: nil)
  return
}
```

New Way

PERMISSIONS

```
// Check if we have authorization to send notification
guard let settings = UIApplication.sharedApplication()
if settings.types == .None {
  let alertController = UIAlertController(title: "😒",
                                          preferredSty

  let settingsAction = UIAlertAction(title: "Settings"

    if let appSettings = NSURL(string: UIApplicationOp
      UIApplication.sharedApplication().openURL(appSet
    }
  }
  alertController.addAction(settingsAction)

  let cancelAction = UIAlertAction(title: "Cancel", st
  alertController.addAction(cancelAction)

  presentViewController(alertController, animated: tru
  return
}
```

```
}

uthorized. Please enable it in Settings to continue.",
```

PERMISSIONS

Local Notifications

LOCAL NOTIFICATIONS

- Local notifications are delivered by the iOS device

LOCAL NOTIFICATIONS

- Messages are handled differently depending on the state of the application
  - Running
  - Background
  - Terminated
- Developer responsibility to handle each case

LOCAL NOTIFICATIONS

Launch app from notification

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {

  if let notification:UILocalNotification = launchOptions?[UIApplicationLaunchOptionsLocalNotificationKey] as? UILocalNotification {
    print("Launch from local notification: \(notification)")
  }

  return true
}
```

- Test for type of notification when launching from terminated state
- Different behavior
  - Go to a view controller
  - Set some date

LOCAL NOTIFICATIONS

Handle notification from suspended

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

LOCAL NOTIFICATIONS

Have to set delegate early

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

LOCAL NOTIFICATIONS

- Remember to handle all the states (if necessary)
  - App in foreground
  - App suspended
  - App terminated

Scheduling Local Notifications

SCHEDULING NOTIFICATIONS

- Notification request contain content and a trigger

- `title` : String containing the primary reason for the alert.
- `subtitle` : String containing an alert subtitle (if required)
- `body` : String containing the alert message text
- `badge` : Number to show on the app's icon.
- `sound` : A sound to play when the alert is delivered. Use `UNNotificationSound.default()` or create a custom sound from a file.
- `launchImageName` : name of a launch image to use if your app is launched in response to a notification.
- `userInfo` : A dictionary of custom info to pass in the notification
- `attachments` : An array of `UNNotificationAttachment` objects. Use to include audio, image or video content.

Content sent with notification

SCHEDULING NOTIFICATIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()
```

- Notification request contain content and a trigger

SCHEDULING NOTIFICATIONS

- Triggers
  - Time
  - Calendar
  - Location

SCHEDULING NOTIFICATIONS

- Schedule a notification for a number of seconds later

Time trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

_ = UNTimeIntervalNotificationTrigger(timeInterval: 300,
                                      repeats: false)
```

SCHEDULING NOTIFICATIONS

- Trigger at a specific date and time
- The trigger is created using a date components object which makes it easier for certain repeating intervals

Calendar trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

let date = Date(timeIntervalSinceNow: 3600)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,],
                                                  from: date)

let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate,
                                            repeats: false)
```

SCHEDULING NOTIFICATIONS

```
let triggerDaily = Calendar.current.dateComponents([.hour,.minute,.second,], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDaily, repeats: true)

let triggerWeekly = Calendar.current.dateComponents([.weekday,.hour,.minute,.second,], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerWeekly, repeats: true)
```

- Repeating notifications

SCHEDULING NOTIFICATIONS

Location trigger

```
let trigger = UNLocationNotificationTrigger(triggerWithRegion:region, repeats:false)
```

- Trigger when a user enters or leaves a geographic region
- The region is specified through a CoreLocation `CLRegion`
- You need to get permissions just like `CoreLocation`

SCHEDULING NOTIFICATIONS

```
UNUserNotificationCenter.current().
    getPendingNotificationRequests {
    requests in
    print(requests)
}
```

- List all the current notifications

SCHEDULING NOTIFICATIONS

```
let center = UNUserNotificationCenter.current()
center.removeAllPendingNotificationRequests()
```

- Delete notifications

SCHEDULING NOTIFICATIONS

```
@IBAction func tapDelete(_ sender: Any) {
  listNotification()

  let center = UNUserNotificationCenter.current()
  center.removeAllPendingNotificationRequests()

  listNotification()
}
```

- Delete notifications

Notification Actions

NOTIFICATION ACTIONS

- Define custom actions for your notifications

NOTIFICATION ACTIONS

```
// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:
    [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
                                      actions: [snoozeAction,deleteAction],
                                      intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

NOTIFICATION ACTIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk CALENDAR"
content.sound = UNNotificationSound.default()
content.categoryIdentifier = "CalendarCategory"

let date = Date(timeIntervalSinceNow: 10)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,], from:
    date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:
    [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
                                      actions: [snoozeAction,deleteAction],
                                      intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

NOTIFICATION ACTIONS

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {

  print("AppDelegate: Did Receive: \(response)")

  // Determine the user action
   switch response.actionIdentifier {
  case UNNotificationDismissActionIdentifier:
    print("Dismiss Action")
  case UNNotificationDefaultActionIdentifier:
    print("Default")
  case "Snooze":
    print("Snooze")
  case "DeleteAction":
    print("Delete")
  default:
    print("Unknown action")
  }
  completionHandler()
}
```

Switch on the actionIdentifier

Push Notifications

PUSH NOTIFICATIONS

- Sent from Apple's servers to device
- Apple Push Notification Service (APNS) is the gateway for push notification
  - SSL certificate
  - Provisioning
  - Networking code
  - Device tokes

image via Big Nerd Ranch iOS Programming

PUSH NOTIFICATIONS

1. An app enables push notification (the user has to confirm that he wishes to receive these notifications)
2. The app receives a “device token”
3. The app sends the device token to your server
4. The server sends a push notification to the Apple Push Notification Service
5. APNS sends the push notification to the user’s device

PUSH NOTIFICATIONS

PUSH NOTIFICATION

- Requirements for push notifications
  - iOS device
  - Paid iOS Developer membership
  - Server
    - Ability to run background processes
    - Install SSL certificates
    - Make outing TLS connections on non-standard ports

PUSH NOTIFICATION

- Many third party solutions for push notifications
  - Parse
  - Firebase
  - App Engine
  - ...

PUSH NOTIFICATIONS

```
{"aps": {
    "badge": 100,
    "alert": "Push Notification Test",
    "sound": "SoundFile.aif"},
    "type":"websiteUrl",
    "data":"http://mySite.com"
}
```

Manage the badges in your code (not incremental)

Include sound files in your bundle

- Push notification payload
  - JSON dictionary
  - <256 bytes

Pass any key-values in the notification

PUSH NOTIFICATION

- Receiving notification require app delegate methods to be implemented
- Code path is the same for local notification

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

nomad / houston

Watch 88 | Star 2,620 | Fork 224

Code | Issues 27 | Pull requests 5 | Projects 0 | Wiki | Insights

Apple Push Notifications; No Dirigible Required http://nomad-cli.com

houston notifications ruby cli nomad apns

157 commits | 1 branch | 19 releases | 32 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

mpvosseller authored and dankimio committed on Jan 24 Set default passphrase to nil instead of the empty string. (#158) Latest commit 592bbeb on Jan 24

bin	Set default passphrase to nil instead of the empty string. (#158)	a month ago
lib	Set default passphrase to nil instead of the empty string. (#158)	a month ago
spec	Mutable content: update README, add spec (#137)	a year ago
.gitignore	Update .gitignore	a year ago
Gemfile	Update code style (#134)	a year ago
LICENSE	Updating copyright	3 years ago
README.md	README: use unregistered_devices instead of devices (#156)	5 months ago
Rakefile	Rakefile: add Bundler tasks, add default task	a year ago
houston.gemspec	Update dependencies	a year ago

README.md

HOUSTON

NOTIFICATIONS WRAP-UP

- Notifications keep users engage by opening your app
- Notifications are not guaranteed to be delivered by Apple
- Tips and tricks:
  - Push was important before multi-tasking in iOS4, but many uses can be implemented as local notifications
  - Third-party option for remote notifications
  - Use CloudKit for remote notifications via subscriptions
- Don't abuse notifications…this is one thing that Apple seems to regularly enforce in the App Store Guidelines
  - "Push notifications can not be used for advertising"

- Journal
  - Extract metadata from images
- CloudKit for data storage/sync
- Local storage using JSON and NSCache
  - Offline first
- Extensions
- Embedded Frameworks
  - Extensions
    - Share and Today
    - App Groups
  - 3D Touch
  - Image Cache (with Core Data)
  - Custom View Controller Transition
- Autotagging imagesiboo
- Mobile first architecture

THE UNIVERSITY OF CHICAGO

• SESSION 4

S

© 2016 T.A. Binkowski

SCREENSHOT IMAGE CACHE

SCREENSHOT IMAGE CACHE

- Screen shot images can be big and they will be scrolled

SCREENSHOT IMAGE CACHE

SUBTITLE

- Options for caching
  - Memory
  - Documents directory
  - ApplicationSupport/Cache
- Options for storage
  - Write file
  - .sqlite
  - Core Data
- Options for Core Data
  - In-memory only
  - Persist with timestamp

SCREENSHOT IMAGE CACHE

SCREENSHOT IMAGE CACHE

```
lazy var applicationDocumentsDirectory: NSURL = {...}()

lazy var managedObjectModel: NSManagedObjectModel = {...}()

lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
  // The persistent store coordinator for the application. This implementation creates and returns a coordinator, having added the store for the application to it. This property is
optional since there are legitimate error conditions that could cause the creation of the store to fail.
  // Create the coordinator and store
  let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)

  // Create store
  let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
  var failureReason = "There was an error creating or loading the application's saved data."
  do {
    try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url, options: nil)
  } catch {
    // Report any error we got.
    var dict = [String: AnyObject]()
    dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
    dict[NSLocalizedFailureReasonErrorKey] = failureReason

    dict[NSUnderlyingErrorKey] = error as NSError
    let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
    // Replace this with code to handle the error appropriately.
    // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
    NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
    abort()
  }

  // Create the image cache persistant store defined as a separate "Configuration"
  let imageCacheURL = self.applicationDocumentsDirectory.URLByAppendingPathComponent("ImageCache.sqlite")
  print("ImageCache URL: \(url)")
  do {
    try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: "ImageCache", URL: imageCacheURL, options: nil)
  } catch {
    print("Image Cache store problem")
    abort()
  }

  return coordinator
```

APPLICATION DATA

IMAGE CACHE

SCREENSHOT IMAGE CACHE

SCREENSHOT IMAGE CACHE

SCREENSHOT IMAGE CACHE

```
e("ImageCache",
go")!)
```

```
inManagedObjectC
```

- Create image entity with metadata to determine when to purge it
- Core Data handles the management of reading/writing the file to/from memory/disk

SCREENSHOT IMAGE CACHE

SCREENSHOT IMAGE CACHE

- Transformable vs Binary Data
  - Big SQLite File
  - Transformable is binary data with some metadata
    - http://stackoverflow.com/questions/22387924/changing-existing-core-data-image-store-from-transformable-to-binary-data-allows

SCREENSHOT IMAGE CACHE

- Additional configurations could be used for multiple store
  - Application (preloaded)
  - Favorites (user defined)
  - ImageCache

BACKGROUND TASKS

BACKGROUND TASKS

SUBTITLE

- State changes in iOS
- Background is a quick stop on way to suspended state
- Applications can be allotted additional time to run tasks in the background

BACKGROUND TASKS

SUBTITLE

- Legitimate background tasks
  - Track location for fitness app
  - Play music over lock screen
  - Download/update data so that it is ready for launch

BACKGROUND TASKS

- Three techniques on iOS
  - Finite-length tasks
    - Start a short task in the foreground and ask for time to finish that task when moved to background
  - URLSession background tasks
    - Initiate downloads in the foreground; hand off management of those downloads to the system
  - Background Execution modes
    - Support specific tasks can declare their support for one or more background execution modes

BACKGROUND TASKS

- Apple is strict about utility for background modes
- Some techniques have been abused, but don't count on it working for long

BACKGROUND DOWNLOAD

BACKGROUND DOWNLOAD

- `URLSession` provides built-in support for background downloads
- System manages them through suspension or termination
  - Will restart app when download complete

USER FORCED TERMINATION KILLS DOWNLOADS UNTIL RELAUNCH

BACKGROUND DOWNLOAD

- Setting up a background download
  - Create the configuration object using the `backgroundSessionConfigurationWithIdentifier`: method of `NSURLSessionConfiguration`
  - Set the value of the configuration object's `sessionSendsLaunchEvents` property to true
  - If your app starts transfers while it is in the foreground, it is recommend that you also set the discretionary property of the configuration object to YES
  - Configure any other properties of the configuration object as appropriate.
  - Use the configuration object to create your NSURLSession object

BACKGROUND DOWNLOAD

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?
  var backgroundSessionCompletionHandler: (() -> Void)?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {...}


  func application(_ application: UIApplication,
                   handleEventsForBackgroundURLSession identifier: String,
                   completionHandler: @escaping () -> Void) {
    print("handleEventsForBackgroundURLSession: \(identifier)")
    self.backgroundSessionCompletionHandler = completionHandler
  }


}
```

ASSIGN THE COMPLETION HANDLER TO A PROPERTY TO KEEP IT ALIVE

BACKGROUND DOWNLOAD

```
//
//  BackgroundDownloadViewController.swift
//  WatchYourBack
//
//  Created by T. Andrew Binkowski on 5/7/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit

class BackgroundDownloadViewController: UIViewController {

  /// Start a download that will persist through moving to background
  @IBAction func tapDownload(_ sender: Any) {…}

  /// ImageView to show our downloaded image
  @IBOutlet weak var imageView: UIImageView!
}


///
///
///
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {


  func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,
                  didWriteData bytesWritten: Int64,
                  totalBytesWritten: Int64,
                  totalBytesExpectedToWrite: Int64) {…}

  func urlSession(_ session: URLSession,
                  task: URLSessionTask,
                  didCompleteWithError error: Error?) {…}

  func urlSession(_ session: URLSession,
                  downloadTask: URLSessionDownloadTask,
                  didFinishDownloadingTo location: URL){…}
```

BACKGROUND DOWNLOAD

```
//  WatchYourBack
//
//  Created by T. Andrew Binkowski on 5/7/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit

class BackgroundDownloadViewController: UIViewController {

  /// Start a download that will persist through moving to background
  @IBAction func tapDownload(_ sender: Any) {
    print("Tap download")
    let config = URLSessionConfiguration.background(withIdentifier: "mobi.uchicago.download")
    config.sessionSendsLaunchEvents = true
    let session = URLSession(configuration: config,
                             delegate: self,
                             delegateQueue: OperationQueue())
    let url = URL(string: "https://static.pexels.com/photos/104827/cat-pet-animal-domestic-104827.jpeg")
    let downloadTask = session.downloadTask(with: url!)
    downloadTask.resume()
  }

  /// ImageView to show our downloaded image
  @IBOutlet weak var imageView: UIImageView!
}


///
///
///
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {
```

BACKGROUND DOWNLOAD

```
import UIKit

class BackgroundDownloadViewController: UIViewController {

  /// Start a download that will persist through moving to background
  @IBAction func tapDownload(_ sender: Any) {…}

  /// ImageView to show our downloaded image
  @IBOutlet weak var imageView: UIImageView!
}


///
///
///
extension BackgroundDownloadViewController: URLSessionDownloadDelegate {


  func urlSession(_ session: URLSession, downloadTask: URLSessionDownloadTask,
                  didWriteData bytesWritten: Int64,
                  totalBytesWritten: Int64,
                  totalBytesExpectedToWrite: Int64) {
    if totalBytesExpectedToWrite > 0 {
      let progress = Float(totalBytesWritten) / Float(totalBytesExpectedToWrite)
      print("Progress \(downloadTask) \(progress)")
    }
  }

  func urlSession(_ session: URLSession,
                  task: URLSessionTask,
                  didCompleteWithError error: Error?) {…}
```

BACKGROUND DOWNLOAD

```
///
func urlSession(_ session: URLSession,
                task: URLSessionTask,
                didCompleteWithError error: Error?) {
  print("Task completed: \(task), error: \(String(describing: error))")
}

///
func urlSession(_ session: URLSession,
                downloadTask: URLSessionDownloadTask,
                didFinishDownloadingTo location: URL){

  print("Download finished: \(location)")

  let path = NSSearchPathForDirectoriesInDomains(FileManager.SearchPathDirectory.documentDirectory,
                                                 FileManager.SearchPathDomainMask.userDomainMask,
                                                 true)[0]
  let fileManager = FileManager()
  let destinationURLForFile = URL(fileURLWithPath: path.appendingFormat("/downloaded.jpg"))
  try? fileManager.removeItem(at: destinationURLForFile)

  do {
    try fileManager.moveItem(at: location, to: destinationURLForFile)
    let image = UIImage(contentsOfFile: path.appendingFormat("/downloaded.jpg"))
    DispatchQueue.main.async {
      self.imageView.image = image
    }
  } catch {
    print("An error occurred while moving file to destination url")
  }
}
```

BACKGROUND DOWNLOAD

- Supported by URLSession on iOS and watchOS

BACKGROUND FETCH

BACKGROUND FETCH

- Special mode that will opportunistically fetch content
- Prevent "blinking data"
- ~ 30 seconds

BACKGROUND FETCH

- When a good opportunity arises, the system wakes or launches your app into the background and calls the app delegate's `application:performFetchWithCompletionHandler:` method
- Subject to OS scheduling (and punishment)
  - Following guidelines gives preferences to future runs

BACKGROUND FETCH

- Set background mode

BACKGROUND FETCH

```
ion, didFinishLaunchingWithOptions launchOptions: [UIAppl

undFetchInterval(UIApplicationBackgroundFetchIntervalMini

dFetchInterval(10)
```

- Set intervals

BACKGROUND FETCH

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?
  var backgroundSessionCompletionHandler: (() -> Void)?


  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {...}

  // Support for background fetch
  func application(_ application: UIApplication, performFetchWithCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void) {

    // A. Find the view controller we want in the hierachy
    if let tabBarController = window?.rootViewController as? UITabBarController {
      if let fetchViewController = tabBarController.viewControllers?[0] as? BackgroundFetchViewController    {
        // B. Execute function in fetch view controller
        fetchViewController.fetchData {
          // C. In completion block, update the UI
          print("In completion block")
          fetchViewController.updateInterface()
          // D. Tell OS we're done so that it can update the multitasking
          // snapshot
          completionHandler(.newData)
        }
      }
    }

  }
```

BACKGROUND FETCH

```
//
//  ViewController.swift
//  WatchYourBack
//
//  Created by T. Andrew Binkowski on 5/7/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit

class BackgroundFetchViewController: UIViewController {

  var dateFormatter: DateFormatter = ...

  @IBOutlet weak var timeLabel: UILabel!

  var currentTime: Date? {
    didSet {
      if let currentTime = self.currentTime {...}
    }
  }

  //
  // MARK: -
  //
  override func viewDidLoad() {...}

  override func didReceiveMemoryWarning() {...}


  func fetchData(_ completion: () -> Void) {...}

  func updateInterface() {...}
}
```

View controller that has functions called in background task

BACKGROUND FETCH

```
class BackgroundFetchViewController: UIViewController {

  var dateFormatter: DateFormatter = ...

  @IBOutlet weak var timeLabel: UILabel!

  var currentTime: Date? {
    didSet {
      if let currentTime = self.currentTime {
        self.timeLabel.text = dateFormatter.string(from: currentTime)
      }
    }
  }

  //
  // MARK: -
  //
  override func viewDidLoad() {
    super.viewDidLoad()
    self.updateInterface()
  }

  //
  // MARK: - Fetching and Updating
  //
  func fetchData(_ completion: () -> Void) {
    // Some long running task
    for i in 0...100 {
      print(i)
    }
    completion()
  }

  func updateInterface() {
    print("UpdateUI")
    self.currentTime = Date()
  }
}
```

Keep track of time and update the UI when changed

Long-running task

Update the label

BACKGROUND FETCH

- Testing on device is unpredictable
- For once, testing on simulator is easier

BACKGROUND FETCH

- Screenshot is updated in multitasking view

BACKGROUND FETCH

- Screenshot is updated in multitaskin

FINITE-LENGTH TASKS

FINITE-LENGTH TASKS

- Execute a task in the background

```
rt UIKit

s BackgroundTaskViewController: UIViewController {

erride func viewDidLoad() {
super.viewDidLoad()
longRunningTask()


nc longRunningTask() {
var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid
bti = UIApplication.shared.beginBackgroundTask(expirationHandler:
  // Clean up after run (or denial)

  // Call the completion handler
  UIApplication.shared.endBackgroundTask(bti)
})

// Do work
for i in 0...100000000 {
  print(i)
}

DispatchQueue.main.async {
  UIApplication.shared.endBackgroundTask(bti)
```

FINITE-LENGTH TASKS

```
  super.viewDidLoad()
  longRunningTask()
}

func longRunningTask() {
  var bti : UIBackgroundTaskIdentifier = UIBackgroundTaskInvalid
  bti = UIApplication.shared.beginBackgroundTask(expirationHandler: {
    // Clean up after run (or denial)

    // Call the completion handler
    UIApplication.shared.endBackgroundTask(bti)
  })

  // Do work
  for i in 0...100000000 {
    print(i)
  }
```

0

What happens when finished

Work

Tell OS you're done

BREAK TIME 😀

ASSIGNMENT 3

BUILD A BETTER APP STORE

SUBTITLE

- Search iTunes API
  - *Step 1. Query iTunes API*
    - Return JSON
  - *Step 2. Loop through JSON*
    - Create new instance 'Application' Entity
    - Copy the important data from JSON to your model
    - Run JSON['description'] through NSLinguistic tagger to extract the key terms you choose

Assignment

Discoverability in the App Store is one of the biggest problems facing developers (that and Core Data and iCloud syncing). In this assignment, we will create a subset of the App Store that is focused on games and implement a search engine using core iOS technologies. In addition, you will allow users to create a wishlist of games they are interested in. They will be able to add games to the wishlist from within the app and from a share extension from the App Store application. They will also be able to view their wishlist from a Today extension.

Application Overview

The application will come bundled with a Core Data store representing app data that is taken from the iTunes search API. You should select a subset of approximately 100 apps from the Game category of the AppStore. The user will launch the app and be presented with a default list of application.

A search bar will be used to query the local Core Data store of games. The results should be returned presented in a stylized table view.

> Tip: Look at the layout of the App Store for a guideline on what information to display for each application. See Figure 1.

BUILD A BETTER APP STORE

APP ARCHITECTURE

APP ARCHITECTURE

- This Week
  - iTunes API query and parsing
    - NSLinguisticTagger
  - Core Data model
    - Load Core Data into table
    - Style table cells (downloading and showing screenshots)
    - Live search
    - SKStoreProductViewController
    - Networking and Core Data Stack into Embedded Framework

VIEW CONTROLLER EVOLUTION

iPhone
Evolution

THE VIEW CONTROLLER

THE VIEW CONTROLLER

IN THE BEGINNING…

- UIViewControllers jobs
  - Manages "screenful of objects"
  - Manage view hierarchy
  - Applications flow between view controllers

THE VIEW CONTROLLER

SUBTITLE CUSTOM VIEW CONTROLLERS IMPLEMENTATION

- Subclass `UIViewController`
- Associate the view controller with a view hierarchy
- Override select APIs to suit your needs
- Add your application logic

Image via Apple

THE VIEW CONTROLLER

SUBTITLE

- Commonly overridden methods
  - Loading callbacks
    - `viewDidLoad()`
    - `viewDidUnload()`
  - Appearance callbacks
    - `viewWillAppear()`
    - `viewDidAppear()`
    - `viewWillDisappear()`
    - `viewDidDisappear()`

```
//
//  ViewController.swift
//  Session1-PageViewController
//
//  Created by T. Andrew Binkowski on 4/2/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
    // MARK: - Navigation

    // In a storyboard-based application, you will often want to
    override func prepare(for segue: UIStoryboardSegue, sender:
        // Get the new view controller using segue.destinationVi
        // Pass the selected object to the new view controller.
    }
    */

}
```

THE VIEW CONTROLLER

Other than segues

- How do view controllers get on/off the screen?
  - By container controllers
  - By presentation

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

  - By dismissal

```
presentingViewController?.dismissViewControllerAnimated(true, completion: nil)
```

  - By direct view manipulation

```
window.rootViewController = rootViewController;
```

THE VIEW CONTROLLER

SUBTITLE

- `UIViewController` advantages
  - Consistent with Apple apps
  - Easily reusable
  - Simplify many of the tasks for managing views and models

Image via Apple

VIEW CONTROLLERS

- For many interfaces it "makes sense"
- Screenful of content

VIEW CONTROLLERS

SUBTITLE

- "Screenful of content" getting harder to define

VIEW CONTROLLERS

WHO'S HUNGRY?

VIEW CONTROLLERS

- Special controllers manage “unit of content” not “pageful of content”
  - UISplitViewController
  - UITabBarController

AND NOW...

- The root view controller does manage a “screenful of content”
- window.rootView Controller = aViewController

VIEW CONTROLLERS

- Summary
  - View controllers are just controllers—the “C” in MVC
  - View controllers manage a view hierarchy—not a single view!
  - View controllers are typically self contained (reusable)
  - View controllers connect and support common iOS application flows
  - More complex aesthetics or application flows need more flexibility

CONTENT VS. CONTAINER VIEW CONTROLLERS

VIEW CONTROLLER CONTAINERS

- And now...
  - Content view controller
    - Your custom view
  - Container view controllers
    - Split view controller
    - Tab bar controllers

VIEW CONTROLLER CONTAINERS

- Container Controllers
  - Manage a hierarchy of child views
    - UITabBarController
    - UINavigationController
    - UISplitViewController
- Content Controllers
  - Manage individual screens
  - Can be more than one view controller per screen (as seen in a UISplitViewController)

VIEW CONTROLLER CONTAINERS

- UINavigationController
  - View controllers on a stack that you push/pop them off to view
  - Transition animation where you may see multiple view controllers
  - Contain additional content views
    - Navigation bar
    - Tool bar

VIEW CONTROLLER CONTAINERS

- UITabBarController
  - Array of view controllers
  - Only one shown at a time

VIEW CONTROLLER CONTAINERS

VIEW CONTROLLER CONTAINERS

custom view controller container

https://github.com/hollance/MHTabBarController

VIEW CONTROLLER CONTAINERS

VIEW CONTROLLER CONTAINMENT

VIEW CONTROLLER CONTAINERS

Blue
View color

Blue Arrow
Subview relationship

Gold
View controller color

Split View Controller

Gray Arrow
Parent view controller relationship

Image via Apple

VIEW CONTROLLER CONTAINERS

CONTENT CONTROLLER

Image via Apple

VIEW CONTROLLER CONTAINERS

Image via Apple

VIEW CONTROLLER CONTAINERS

Image via Apple

VIEW CONTROLLER CONTAINERS

- Container views allow a view to have child views which are in turn managed by their own view controllers

Image via Apple

VIEW CONTROLLER CONTAINERS

- Container Controllers are responsible for child/ parent relationships between view controllers
  - Pushing view controllers
  - Remove view controllers

Image via Apple

VIEW CONTROLLER CONTAINERS

Tasks

- Configuring a View Controller Using Nib Files
- Interacting with Storyboards and Segues
- Managing the View
- Presenting View Controllers
- Supporting Custom Transitions and Presentations
- Responding to View Events
- Configuring the View's Layout Behavior
- Testing for Specific Kinds of View Transitions
- Configuring the View Rotation Settings
- Adapting to Environment Changes
- **Managing Child View Controllers in a Custom Container**

Language: Objective-C **Swift** Both On This Page Options

Managing Child View Controllers in a Custom Container

```
childViewControllers
addChildViewController(_:)
removeFromParentViewController()
transitionFromViewController(_:toViewController:duration:options:animations:completion:)
shouldAutomaticallyForwardAppearanceMethods()
beginAppearanceTransition(_:animated:)
endAppearanceTransition()
setOverrideTraitCollection(_:forChildViewController:)
overrideTraitCollectionForChildViewController(_:)
```

Responding to Containment Events

```
willMoveToParentViewController(_:)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically
  - 2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.
    - Container sets the child's size and position before adding the view; order matters
  - 3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically

2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.

- Container sets the child's size and position before adding the view; order matters

  - 3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Add a view controllers view to containers view hierarchy
  - 1. Calls the container's addChildViewController: method to add the child .
    - This calls the child's `willMoveToParentViewController:` method automatically
  - 2. It accesses the child's view property to retrieve the view and adds it to its own view hierarchy.
    - Container sets the child's size and position before adding the view; order matters

3. Call the child's didMoveToParentViewController: method to signal that the operation is complete

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

```
// Create instance of view controller and color it.
let gvc = ColorViewController()
gvc.view.backgroundColor = UIColor.redColor()

// 1.
addChildViewController(gvc)

// 2.
gvc.view.frame = CGRectMake(50, 50, 200, 200)
view.addSubview(gvc.view)

// 3.
gvc.didMoveToParentViewController(self)
```

CONNECTING VIEW CONTROLLERS

REMOVING A VIEW CONTROLLER FROM A CONTAINER

```
// 1. Calls the child's willMoveToParentViewController: method with a
// parameter of nil to tell the child that it is being removed.
sender.willMoveToParentViewController(nil)

// 2. Clean up the view hierarchy
sender.view.removeFromSuperview()

// 3. Calls the child's removeFromParentViewController method to remove it
// from the container.
sender.removeFromParentViewController()
```

CONNECTING VIEW CONTROLLERS

SUBTITLE

- Remove a view controllers view to containers view hierarchy
  - 1. Calls the child's `willMoveToParentViewController:` method with a parameter of nil to tell the child that it is being removed.
  - 3. Calling the `removeFromParentViewController` method automatically calls the child's `didMoveToParentViewController:` method

```
// 1. Calls the child's willMoveToParentViewController: method
// parameter of nil to tell the child that it is being removed
sender.willMoveToParentViewController(nil)

// 2. Clean up the view hierarchy
sender.view.removeFromSuperview()

// 3. Calls the child's removeFromParentViewController method
// from the container.
sender.removeFromParentViewController()
```

CONNECTING VIEW CONTROLLERS

- But do they know about each other?
  - Not if you're practicing good MVC
- Possible ways to pass messages
  - NSNotifications
  - KVO
  - Delegation

CONNECTING VIEW CONTROLLERS

- Create a protocol to allow communication between parent and child

child view controller

CONNECTING VIEW CONTROLLERS

- Tapping on the button in the child will remove it from the parent

ViewControllers 〉 iPhone 6 | Finished running ViewControllers on iPhone 6

ColorViewController.xib | Debug

llers 〉 ViewControllers 〉 ColorViewController.swift 〉 ColorViewController

```
roller.swift
rs

 Andrew Binkowski on 3/27/16.
016 The University of Chicago, Department of Computer Science. All rights reserved.



 allows the remove child view controller cycle to be called
 view controller

vControllerDelegate: class {...}


er that can either dismiss itself or tell a parent view
 remove it

ntroller: UIViewController {

rence to t
te: Co

tions

tapCloseButton(sender: UIButton) {

Controller != nil {
emoveFromContainerViewController(self)

iewController?.dismissViewControllerAnimated(true, completion: nil)



iewDidLoad() {...}
```

CONNECTING VIEW CONTROLLERS

- More versatile implementation
- Determine if under containment or content view controller

ViewControllers 〉 iPhone 6 | Finished running ViewControllers on iPhone 6

ColorViewController.xib | Debug

llers 〉 ViewControllers 〉 ColorViewController.swift 〉 ColorViewController

```
roller.swift
rs

 Andrew Binkowski on 3/27/16.
016 The University of Chicago, Department of Computer Science. All rights reserved.



 allows the remove child view controller cycle to be called
 view controller

vControllerDelegate: class {•••}


er that can either dismiss itself or tell a parent view
 remove it

ntroller: UIViewController {

rence to the parent view controller
te: ColorViewControllerDelegate?


tions


tapCloseButton(sender: UIButton) {

Controller != nil {
emoveFromContainerViewController(self)

iewController?.dismissViewControllerAnimated(true, completion: nil)




iewDidLoad() {•••}
```

CHILD VIEW CONTROLLER VIEW TRANSITIONS

VIEW CONTROLLER CONTAINERS

- Transit n children s

VIEW CONTROLLER CONTAINERS

- Tr
  be dren
  vi ers

```
unc tapSwapViewControllers(_ sender: UIBarButtonItem) {
 a new view controller (yellow color)
 ColorViewController(nibName: "GreenViewController", bu
ate = self
backgroundColor = UIColor.yellow
frame = CGRect(x: 50, y: 50, width: 200, height: 200)
alpha = 0.0

ew controllers have to have the same parent...but
 add it to the view hierarchy
iewController(gvc)

tion between them; fade out/ fade in
sition(from: gvc2!, to: gvc, duration: 2.0,
      options: UIViewAnimationOptions.curveEaseInOut,
      animations: {
        self.gvc2!.view.alpha = 0.0
        gvc.view.alpha = 1.0

mated) in
'done")
```

VIEW CONTROLLER CONTAINERS

CONTAINER VIEW

- Make you own tab bar

-

BUTTONS

HOME. USERS. SETTINGS.

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

SUBTITLE

- Add Container Views to a View Controller in Interface Builder
- Automatically performs the parent-child dance for you

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

VIEW CONTROLLER CONTAINMENT USING INTERFACE BUILDER

SOME IMPORTANT THINGS TO REMEMBER ABOUT CONTAINMENT

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Container controllers are responsible for child/parent relationships
- There are consistent and inconsistent hierarchies
  - `UIViewControllerHierarchyInconsistencyException`
    - The view associated with a view controller was added directly to a view hierarchy
    - Pointing to view controller that are not in that view hierarchy
- Appearance callbacks are different than content view controllers

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Appearance callbacks
  - `addChildViewController` does not trigger appearance callbacks (e.g. viewDid/WillAppear)
  - This view controller is a child (nothing to do with appearance)
- Appearance callbacks called when they move in/out of the window hierarchy
  - Not necessarily when you “see” it
  - Add a view “behind” a view and you do not see
  - Think of it as “Made it into the window’s view hierarchy”

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

APPEARANCE CALLBACKS

- viewWillAppear:
  - ▪ Called before the view is added to the windows' view hierarchy
  - ▪ Called before `view.layoutSubviews()` (if necessary)
- viewDidAppear:
  - Called after the view is added to the view hierarchy
  - Called after view.layoutSubviews() (if necessary)
- viewWillDisappear:
  - Called before the view is removed from the windows' view hierarchy
- viewDidDisappear:
  - Called after the view is removed from the windows' view hierarchy

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- When should you create a custom view controller?
  - Never, use Apple’s built in containers
  - Design, you want custom aesthetic
  - Function, new application flow

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Apple is updating container controllers
  - iOS5 UISplitViewController always shows master view controller
  - iOS7 Custom ViewController transitions (modal vc on iPhone)
  - iOS8 Custom presentation controllers
  - ...
  - iOS11 UIHamburger??

WHAT YOU NEED TO KNOW ABOUT HIERARCHIES

- Practical suggestions for building a container view controller (from Apple)
  - Design as content view controller first
  - Never access any view other than the top-level view of the child view controller (i.e. don’t expose unnecessary details to children)
  - The container should use protocols to declare methods and properties

THE UNIVERSITY OF CHICAGO

• SESSION 3

S

© 2016 T.A. Binkowski

IOS NEWS

IOS NEWS

- More CloudKit integration

Rumor claims Apple TV to add multi-user login and Picture-in-Picture video with tvOS 11

9to5Mac · by Benjamin Mayo · Yesterday

Apple will almost certainly unveil tvOS 11 (along with iOS 11, macOS 10.13 and watchOS 4) at WWDC in June, but not much has been said about what new stuff these software

IOS NEWS

- It never gets old…

Instagram goes after Pinterest with saved post collections

The Unofficial Apple Weblog (TUAW) · by Jon Fingas · 10:38 am

IOS NEWS

- Apple Podcasts
- iTunes Connect down this week
- Big changes for iTunes???

IOS NEWS

- App review extortion?

CLASS NEWS

CLASS NEWS

- Office Hours this Thursday 10-11:30 in Young 302
- Email going out this week
  - About case study
  - Game preferences with SpriteKit
  - Final project ideas

CORE DATA

hnique	Pros	Cons
hiving	Allows ordered relationships (arrays, not sets). Easy to deal with versioning.	Reads all the objects in (no faulting). No incremental updates.
ice	Makes it easy to share data with other devices and applications.	Requires a server and a connection to internet.
e Data	Lazy fetches by default. Incremental updates.	Versioning is awkward (but can certa be done using an `NSModelMapping`). No real ordered relationships (at the t this is being written).

- Options for data storage and persistence

CORE DATA

SUBTITLE

- A framework that provides object-relational mapping
  - Converts objective-c objects into data stored on disk
- Creates a visual mapping between database and objects
- An object-oriented API that conforms to MVC paradigm
- API for constructing detailed queries
- Access the data using properties on the objects

Introduction to Core Data Programming Guide

The Core Data framework provides generalized and automated solutions to common tasks associate
and object graph management, including persistence.

Who Should Read This Document

You should read this document to gain an understanding of the Core Data framework. *You must be*
of Cocoa development, including the Objective-C language and memory management.

Important: Although this document provides a thorough treatment of the fundamentals of the Cor
simply reading from start to finish is not a good strategy for learning how to use the technology ef
should typically augment your understanding by following the related tutorials provided in the Refe
description of the recommended learning path, see *Core Data Starting Point*.

Organization of This Document

The following articles explain the problems the Core Data Framework addresses, the solutions it pro
functionality, and common tasks you might perform:

- Technology Overview describes what Core Data is and why you might choose to use it.
- Core Data Basics describes the fundamental architecture of the technology.
- Managed Object Models describes the features of a managed object model.
- Using a Managed Object Model describes how you use a managed object model in your applicatio
- Managed Objects describes the features of a managed object, the `NSManagedObject` class, and h

CORE DATA

SUBTITLE

- Core Data is an object graph management and persistence framework
- Core Data is not a database!

CORE DATA

- RDB table =~ class (entity)
  - Column =~ class instance variables (attributes)

CORE DATA

WHAT YOU SHOULD KNOW ABOUT CORE DATA

SUBTITLE

- Object graph store
  - Model layer represent state of your app
- Persistence
  - Read/write state of model objects to disk
- Querying
  - SQL-like parameter based queries
- Versioning
    - Lightweight migration

CORE DATA

WHAT YOU PROBABLY DO NOT KNOW

SUBTITLE

- Change tracking
- Optimize data work flows while in memory
- Faulting
  - Reduce application's memory usage
- Uniquing
  - Ensures that, in a given managed object context, you never have more than one managed object to represent a given record

CORE DATA

Faulting

Uniquing

Faulting is a mechanism Core Data employs to reduce your application's memory usage
Uniquing ensures that you never have more than one managed object to represent a given record

CORE DATA

NSManagedObjectContext
NSManaged Object
NSManaged Object
NSManaged Object
NSPersistent StoreCoordinator
NSPersistent Store
SQLite
File System
Typical Core Data Stack

CORE DATA

NSManagedObjectContext
NSManaged Object
NSManaged Object
NSManaged Object
NSManagedObjectContext
NSManaged Object
NSManaged Object
NSManaged Object
NSManagedObjectContext
NSManaged Object
NSManaged Object
NSManaged Object
NSPersistent StoreCoordinator
NSPersistent Store
NSPersistent Store
NSPersistent Store
SQLite
SQLite
SQLite
File System
File System
File System
Background
Advanced Core Data Stack

CORE DATA

SUBTITLE

- Multiple Managed Object Context (MOC) setups
  - MOC only know about their objects
  - Objects only know about their MOC
- Persistent store doesn't care about what its storing

CORE DATA

APPLICATION FLOW

SUBTITLE

- Setup Core Data Stack
  - Object model
  - Persistent store
  - MOC

```
lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it. This property is optional since there are legitimate
     error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, e
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
                in a shipping application, although it may be useful during

            /*
             Typical reasons for an error here include:
             * The parent directory does not exist, cannot be created, or di
             * The persistent store is not accessible, due to permissions or
             * The device is out of space.
             * The store could not be migrated to the current model version.
             Check the error message to determine what the actual problem wa
             */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
                in a shipping application, although it may be useful during
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

CORE DATA

APPLICATION FLOW

2017-CoreDataBasics 〉 iPhone 7 Plus

Running PodcastAPI

SeardchTableViewController.swift

_017_CoreDataBasics.xcdatamodel

2017-CoreDataBasics 〉 2017-CoreDataBasics 〉 _017_CoreDataBasics.xcdatamodeld 〉 _017_CoreDataBasics.xcdatamodel 〉 Person

- 2017-CoreDataBasics M
  - 2017-CoreDataBasics
    - AppDelegate.swift M
    - ViewController.swift M
    - Main.storyboard M
    - SeardchTable...Controller.swift A
    - TableViewController.swift A
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - _017_CoreData....xcdatamodeld M
  - PodcastAPI.playground M
  - Products

ENTITIES

- Person

FETCH REQUESTS

CONFIGURATIONS

- Default

Attributes

Attribute	Type
age	Integer 16
name	String
timestamp	Date

Relationships

Relationship	Destination	Inverse

Fetched Properties

Fetched Property	Predicate

CORE DATA

APPLICATION FLOW

SUBTITLE

- Create in instance of your entity and set the property values

```
var person = Person(context: context!)
person.name = "Homer"
person.age = 42
person.timestamp = Date() as NSDate

person = Person(context: context!)
person.name = "Marge2"
person.age = 41
person.timestamp =  Date() as NSDate
```

CORE DATA

APPLICATION FLOW

- Changes are kept in memory until you explicitly call ‘save’
- What happens?
  - MOC figures out what has changed
  - The managed object context then passes these changes on to the persistent store coordinator
  - The persistent store coordinator coordinates with the store (SQLite3 db) to write into the SQL database on disk

```
asChanges {

:Context.save()

mplementation with code to handle the e

; the application to generate a crash lo
 should not use this function in a ship
although it may be useful during develop
or as NSError
 error \(nserror), \(nserror.userInfo)"
```

CORE DATA

APPLICATION FLOW

- Make a fetch request
- Core Data brings back entity objects
- Fine grain control on the performance

```
n objects
uest: NSFetchRequest<Person> = Per


ies = try context?.fetch(personFet

hedEntities! {
r.name!) ##########")



in response to error condition
```

CORE DATA

Why Should You Use Core Data?

Already done for you

There are a number of reasons why it may be appropriate for you to use Core Data. One of the simplest metrics is that, with Core Data, the amount of code you write to support the model layer of your application is typically 50% to 70% smaller as measured by lines of code. This is primarily due to the features listed above—the features Core Data provides are features you don't have to implement yourself. Moreover they're features you don't have to test yourself, and in particular you don't have to optimize yourself.

It's a mature technology

Core Data has a mature code base whose quality is maintained through unit tests, and is used daily by millions of customers in a wide variety of applications. The framework has been highly optimized over several releases. It takes advantage of information provided in the model and runtime features not typically employed in application-level code. Moreover, in addition to providing excellent security and error-handling, it offers best memory scalability of any competing solution. Put another way: you could spend a long time carefully crafting your own solution optimized for a particular problem domain, and not gain any performance advantage over what Core Data offers for free for any application.

Xcode integration

In addition to the benefits of the framework itself, Core Data integrates well with the OS X tool chain. The model design tools allow you to create your schema graphically, quickly and easily. You can use templates in the Instruments application to measure Core Data's performance, and to debug various problems. On OS X desktop, Core Data also integrates with Interface Builder to allow you to create user interfaces from your model. These aspects help to further shorten application design, implementation, and debugging cycles.

CORE DATA

What Core Data Is Not

NOT A RDB

Having given an overview of what Core Data is and does, and why it may be useful, it is also useful to correct some common misperceptions and state what it is not.

- Core Data is not a relational database or a relational database management system (RDBMS).

  Core Data provides an infrastructure for change management and for saving objects to and retrieving them from storage. It can use SQLite as one of its persistent store types. It is not, though, in and of itself a database. (To emphasize this point: you could for example use just an in-memory store in your application. You could use Core Data for change tracking and management, but never actually save any data in a file.)

- Core Data is not a silver bullet.

  Core Data does not remove the need to write code. Although it is possible to create a sophisticated application solely using the Xcode data modeling tool and Interface Builder, for more real-world applications you will still have to write code.

- Core Data does not rely on Cocoa bindings.

  Core Data integrates well with Cocoa bindings and leverages the same technologies—and used together they can significantly reduce the amount of code you have to write—but it is possible to use Core Data without bindings. You can readily create a Core Data application without a user interface (see *Core Data Utility Tutorial*).

"real-world"?

CORE DATA

DOWNFALLS

```
NSManagedObjectModel *model = <#Get a model#>;
NSFetchRequest *requestTemplate = [[NSFetchRequest alloc] init];
NSEntityDescription *publicationEntity =
          [[model entitiesByName] objectForKey:@"Publication"];
[requestTemplate setEntity:publicationEntity];

NSPredicate *predicateTemplate = [NSPredicate predicateWithFormat:
                                  @"(mainAuthor.firstName like[cd] $FIRST_NAME) AND \
                                  (mainAuthor.lastName like[cd] $LAST_NAME) AND \
                                  (publicationDate > $DATE)"];
[requestTemplate setPredicate:predicateTemplate];
[model setFetchRequestTemplate:requestTemplate
                      forName:@"PublicationsForAuthorSinceDate"];
```

- Verbose (simple fetch)
- Versioning
- Historical misunderstanding

CORE DATA STACK IN IOS10

CORE DATA

APPLICATION FLOW

SUBTITLE

- This was tremendously simplified in iOS10
- Most criticism of CoreData it probably is outdated

```
lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it. This property is optional since there are legitimate
     error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, e
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
                in a shipping application, although it may be useful during

            /*
             Typical reasons for an error here include:
             * The parent directory does not exist, cannot be created, or di
             * The persistent store is not accessible, due to permissions or
             * The device is out of space.
             * The store could not be migrated to the current model version.
             Check the error message to determine what the actual problem wa
             */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            // Replace this implementation with code to handle the error app
            // fatalError() causes the application to generate a crash log a
                in a shipping application, although it may be useful during
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

CORE DATA

APPLICATION FLOW

- Setup Core Data Stack
  - Object model
  - Persistent store
  - MOC

Old way had to set up all parts of the stack manually

```
he Core Data store file. This code uses a directory named "mobi.uchicago._016_CoreDataStack" in the applicatio
rDirectory(.DocumentDirectory, inDomains: .UserDomainMask)


{
 This property is not optional. It is a fatal error for the application not to be able to find and load its mo
rce("_016_CoreDataStack", withExtension: "momd")!
URL)!


reCoordinator = {
ication. This implementation creates and returns a coordinator, having added the store for the application to

anagedObjectModel: self.managedObjectModel)
ByAppendingPathComponent("SingleViewCoreData.sqlite")
 or loading the application's saved data."

SSQLiteStoreType, configuration: nil, URL: url, options: nil)


 initialize the application's saved data"
lureReason

r
OR_DOMAIN", code: 9999, userInfo: dict)
r appropriately.
e a crash log and terminate. You should not use this function in a shipping application, although it may be us
rappedError.userInfo)")


t = {
plication (which is already bound to the persistent store coordinator for the application.) This property is o
```

```
_managedObjectModel = [[NSManagedObjectModel alloc] initWithContentsOfURL:[self modelURL]];
_persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
                                                initWithManagedObjectModel:self.managedObjectModel];


_managedObjectContext = [[NSManagedObjectContext alloc]
                                                initWithConcurrencyType:NSMainQueueConcurrencyType];


self.managedObjectContext.persistentStoreCoordinator = self.persistentStoreCoordinator;
[self.managedObjectContext.persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
                                                                      configuration:nil
                                                                                URL:[self storeURL]
                                                                            options:nil
                                                                              error:&error];
```

- Application Flow: Setup Core Data Stack

CORE DATA

```
tyDescription.entityForName("Standard", inManagedObjec
ntity!, insertIntoManagedObjectContext: context)
dentifier
a["category"] as? String
= data["stateStandard"] as? String
data["subCategory"] as? String
uid"] as? String
```

- Create/Edit entities

CORE DATA

APPLICATION FLOW

```
guard self._fetchedResultsController == nil else {
  return self._fetchedResultsController!
}
let fetchRequest = NSFetchRequest(entityName: "Lesson")
let sectionSortDescriptor = NSSortDescriptor(key: "currentIndexSection", ascending: true)
let rowSortDescriptor = NSSortDescriptor(key: "currentIndexRow", ascending: true)
fetchRequest.sortDescriptors = [sectionSortDescriptor,rowSortDescriptor]
if let cp = currentPredicate {
  fetchRequest.predicate = cp
} else {
  fetchRequest.predicate = defaultPredicate
}

let frc = NSFetchedResultsController(fetchRequest: fetchRequest, managedObjectContext: self.context, sectionNameKeyPath: "currentIndexSection", cacheName: nil)
frc.delegate = self
self._fetchedResultsController = frc
return self._fetchedResultsController!
```

- Fetch

CORE DATA STACK

CORE DATA STACK

SUBTITLE

- Core Data stack
  - One or more managed object contexts
  - A single persistent store coordinator
  - Persistent store

CORE DATA STACK

"Scratch Pad" for managed objects

Model objects

Aggregates all the stores

Maps between records in the store and objects in the application

CORE DATA STACK

SUBTITLE

- Persistent Store Coordinator defines stack
- Can be multiple contexts, stores, etc.

CORE DATA STACK

XCODE TEMPLATE

- Xcode will create codes stubs for working the persistent store
- AppDelegate with have a property called `persistentContainer`
- These are fairly complete and will not need to be dramatically altered

```
: NSPersistentContainer = {

r for the application. This implementation
ontainer, having loaded the store for the
 property is optional since there are legitimate
ould cause the creation of the store to fail.

tentContainer(name: "_017_CoreDataBasics")
Stores(completionHandler: { (storeDescription, error) in
 as NSError? {
implementation with code to handle the error appropriately.
causes the application to generate a crash log and terminate. You
g development.


 for an error here include:
rectory does not exist, cannot be created, or disallows writing.
t store is not accessible, due to permissions or data protection
 out of space.
ld not be migrated to the current model version.
 message to determine what the actual problem was.

solved error \(error), \(error.userInfo)")
```

CORE DATA STACK

XCODE TEMPLATE

- Provides a function to save the current store

```
a Saving support

 {
."
ersistentContainer.viewContext
hanges {

text.save()
```

```
ace this implementation with code to handle the e
lError() causes the application to generate a cra
ful during development.
rror = error as NSError
ror("Unresolved error \(nserror), \(nserror.userI
```

CORE DATA STACK

WORKING WITH DATA

- Keys are NSString (i.e. 'thumbnailData','date', 'name')
- The value is whatever is stored
  - Numbers and booleans are NSNumber type objects
  - “To-many” relationships are NSSet
  - Single relationships are NSManagedObjects
  - Binary data is NSData

CORE DATA STACK

WORKING WITH DATA

SUBTITLE

- Behind the scenes
  - SQL is generated
  - Results are fetched (lazily)
- Writes to database are only done in memory
  - Need to save them or you will loose data

```
func saveContext () {
    if managedObjectContext.hasChanges {
        do {
            try managedObjectContext.save()
        } catch {
            // Replace this implementation with code to handle the error
            // appropriately.
            // abort() causes the application to generate a crash log and
            // terminate. You should not use this function in a shipping
            // application, although it may be useful during development.
            let nserror = error as NSError
            NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
            abort()
        }
    }
}
```

OLD CORE DATA STACK

CORE DATA STACK

XCODE TEMPLATE

SUBTITLE

- Xcode will create codes stubs for working with `NSManagedObjectContext`
- AppDelegate with have a property called `managedObjectContext`
- These are fairly complete and will not need to be dramatically altered

iPhone 6s Plus | 2016-CoreDataStack: **Ready** | Today at 12:32 AM

AppDelegate.swift

2016-CoreDataStack 〉 2016-CoreDataStack 〉 AppDelegate.swift 〉 AppDelegate

```
}

// MARK: - Core Data stack

lazy var applicationDocumentsDirectory: NSURL = {
    // The directory the application uses to store the Core Data store file
    let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDir
    return urls[urls.count-1]
}()

lazy var managedObjectModel: NSManagedObjectModel = {
    // The managed object model for the application. This property is not c
    let modelURL = NSBundle.mainBundle().URLForResource("_016_CoreDataStack
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()

lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    // The persistent store coordinator for the application. This implement
    // Create the coordinator and store
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self
    let url = self.applicationDocumentsDirectory.URLByAppendingPathComponen
    var failureReason = "There was an error creating or loading the applica
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, conf
    } catch {
        // Report any error we got.
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the applica
        dict[NSLocalizedFailureReasonErrorKey] = failureReason

        dict[NSUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999,
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and term
        NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }

    return coordinator
}()
```

CORE DATA STACK

XCODE TEMPLATE

MODEL FILE

```
lazy var managedObjectModel: NSManagedObjectModel = {
    // The managed object model for the application. This property is not optional. It is a fatal error for the applicat
    let modelURL = NSBundle.mainBundle().URLForResource("_016_CoreDataStack", withExtension: "momd")!
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()
```

```
lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    // The persistent store coordinator for the application. This implementation creates and returns a coordinator, havi
    // Create the coordinator and store
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
    let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
    var failureReason = "There was an error creating or loading the application's saved data."
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, configuration: nil, URL: url, options: nil)
    } catch {
        // Report any error we got.
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved data"
        dict[NSLocalizedFailureReasonErrorKey] = failureReason

        dict[NSUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999, userInfo: dict)
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and terminate. You should not use this function in a s
        NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }

    return coordinator
}()
```

STORE FILE

CORE DATA STACK

XCODE TEMPLATE

SUBTITLE

- Best practices (according to Apple)
  - Pass NSManagedObject around via API
  - Do not treat as a global in AppDelegate

The way people typically did it

```
// MARK: - Core Data stack

lazy var applicationDocumentsDirectory: NSURL = {
    // The directory the application uses to store the Core Data store file
    let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDir
    return urls[urls.count-1]
}()

lazy var managedObjectModel: NSManagedObjectModel = {
    // The managed object model for the application. This property is not o
    let modelURL = NSBundle.mainBundle().URLForResource("_016_CoreDataStack
    return NSManagedObjectModel(contentsOfURL: modelURL)!
}()

lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
    // The persistent store coordinator for the application. This implement
    // Create the coordinator and store
    let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self
    let url = self.applicationDocumentsDirectory.URLByAppendingPathComponen
    var failureReason = "There was an error creating or loading the applica
    do {
        try coordinator.addPersistentStoreWithType(NSSQLiteStoreType, confi
    } catch {
        // Report any error we got.
        var dict = [String: AnyObject]()
        dict[NSLocalizedDescriptionKey] = "Failed to initialize the applica
        dict[NSLocalizedFailureReasonErrorKey] = failureReason

        dict[NSUnderlyingErrorKey] = error as NSError
        let wrappedError = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999,
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and termi
        NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
        abort()
    }

    return coordinator
}()
```

CORE DATA STACK

WORKING WITH DATA

- Create objects in the database
  - insertNewObjectForEntityName()
- Access your data
  - - (id)valueForKey:(NSString *)key;
- Set your data
  - - (void)setValue:(id)value forKey:(NSString *)key;

Without a custom subclass

CREATE A CORE DATA MODEL

CORE DATA MODEL

- Provide a simple data model (.xcdatamodel)
- Xcode will generate the boiler plate methods

CORE DATA

SUBTITLE

- .xcdatamodeld is a directory
  - Stores data model
  - Provides built-in versioning
- Changing your data model will break your app
  - Think about the changes you may make

CORE DATA

- Model
  is .xcdatamodel file
- Tasks
  - Add entities
  - Define attributes
- Attributes and relationships of an entity equivalent to @properties of a class

- Entity maps a database table to an object in our code

CORE DATA

- NSManagedObject
  - Subclass of NSObject
  - Access you Core Data model and data
  - The returned object of a Core Data query
  - Key-Value pairs represent the properties of an entity
  - Not thread safe (i.e. do work on main thread)

- Subclass NSManagedObject to access the properties
  - Can also get them using key-value coding with a subclass (this is uncommon)

CORE DATA

- Xcode will generate subclass files of NSManagedObject for you

CORE DATA

- Attributes map to a database column
- Will be able to access via @property in code

CORE DATA

SUBTITLE

- Attributes
  - Type
    - Maps to foundation object
  - Optional
  - Transient
    - Create a property that exist on object, but have no backing
  - Indexed
    - Create an index in the SQLite backend
    - Only what you will search on

CORE DATA

SUBTITLE

- Attribute types
  - Foundation types
    - NSData for everything else
  - Binary Data for holding images
  - Transformable for nonstandard object types

CORE DATA

SUBTITLE

- External Binary Data
  - New in iOS5
- When enabled, Core Data heuristically decides on a per-value basis
  - Save the data directly in the database
  - Store a URI to a separate file

CORE DATA

SUBTITLE

- Image storage guidelines
  - < 100 kb store in entity
  - < 1 mb store in a separate entity to avoid performance issues
  - > 1 mb store on disk and reference the path storing nonstandard object types within Core Data

CORE DATA

- Entities can have relationships
- Example: Users and Session
  - A user can have multiple sessions
  - A session has only one user

CORE DATA

Session
User
FETCH REQUESTS
CONFIGURATIONS
Default
Attribute
Type
id
String
Relationships
Relationship
Destination
Inverse
sessions
Session
user
Fetched Properties
Fetched Property
Predicate
Name
sessions
Properties
Transient
Optional
Destination
Session
Inverse
user
Delete Rule
Nullify
Type
To Many
Arrangement
Ordered
Count
Unbounded
Minimum
Unbounded
Maximum
Advanced
Index in Spotlight
Store in External Record File
User Info
C Block typedef - Define a block as a type.
C Inline Block as Variable - Save a block to a variable to allow reuse or

CORE DATA

SUBTITLE

- "To Many" relationship
  - Each user can have many different sessions
- Maps to an `NSSet` of `NSManagedObjects`
  - Remember that sets have no order guarantee

CORE DATA

- Inverse relationship from user to sessions
  - Each session can have only 1 user

CORE DATA

SUBTITLE

- Inverse goes only one way
- Maps to a single NSManagedObject

CORE DATA

Session
User
FETCH REQUESTS
CONFIGURATIONS
Default
Entity Attribute Type
Session active Boolean
Session endTime Date
User id String
Session startTime Date
Session uuid String
Relationships
Entity Relationship Destination Inverse
User sessions Session user
Session user User sessions
Fetched Properties
Entity Fetched Property Predicate
Name Multiple Values
Class NSManagedObject
Abstract Entity
Parent Entity No Parent Entity
Indexes
User Info
Key Value
C Block typedef - Define a block as a type.
C Inline Block as Variable - Save a block to a variable to allow reuse or

CREATE NSMANAGEDOBJECT SUBCLASS PRIOR TO IOS10

NSMANAGED OBJECT SUBCLASS

SUBTITLE

- Simplify access of object attributes
  - No type checking
- Create a subclass of NSManagedObject
  - @property for each attribute
- Xcode will auto-generate them for you

NSMANAGED OBJECT SUBCLASS

NSMANAGED OBJECT SUBCLASS

NSMANAGED OBJECT SUBCLASS

ENTITIES
- Session
- User

FETCH REQUESTS

CONFIGURATIONS
- Default

Select the entities you would like to manage

Select	Entity
✓	Session
✓	User

Cancel | Previous | Next

ultiple Values
SManagedObject
Abstract Entity
o Parent Entity

Value

typedef - Define a block as

Inline Block as Variable - Save a block to a variable to allow reuse or passing it as an argument.

NSMANAGED OBJECT SUBCLASS

NSMANAGED OBJECT SUBCLASS

- User+CoreDataProperties.swift
- User.swift
- 2016-CoreDataStack
  - AppDelegate.swift
  - ViewController.swift
  - Main.storyboard
  - Assets.xcassets
  - LaunchScreen.storyboard
  - Info.plist
  - _016_CoreDataS...k.xcdatamodeld
- Products

```swift
//  User.swift
//  2016-CoreDataStack
//
//  Created by T. Andrew Binkowski on 4/11/16.
//  Copyright © 2016 The University of Chicago, Department of Computer Science. All rights reserved.
//

import Foundation
import CoreData


class User: NSManagedObject {

// Insert code here to add functionality to your managed object subclass

}
```

Add methods for User class (don't have to be related to Core Data)

NSMANAGED OBJECT SUBCLASS

- User+CoreDataProperties.swift
- User.swift
- 2016-CoreDataStack
  - AppDelegate.swift
  - ViewController.swift
  - Main.storyboard
  - Assets.xcassets
  - LaunchScreen.storyboard
  - Info.plist
  - _016_CoreDataS...k.xcdatamodeld
- Products

```swift
//  User+CoreDataProperties.swift
//  2016-CoreDataStack
//
//  Created by T. Andrew Binkowski on 4/11/16.
//  Copyright © 2016 The University of Chicago, Department of Computer Science. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension User {

    @NSManaged var name: String?
    @NSManaged var sessions: NSSet?

}
```

NSSet for relationships

Extension on the User class

Optionals

NSMANAGED OBJECT SUBCLASS

- User+CoreDataProperties.swift
- User.swift
- 2016-CoreDataStack
  - AppDelegate.swift
  - ViewController.swift
  - Main.storyboard
  - Assets.xcassets
  - LaunchScreen.storyboard
  - Info.plist
  - _016_CoreDataS...k.xcdatamodeld
- Products

```swift
//  User+CoreDataProperties.swift
//  2016-CoreDataStack
//
//  Created by T. Andrew Binkowski on 4/11/16.
//  Copyright © 2016 The University of Chicago, De
//
//  Choose "Create NSManagedObject Subclass…" from
//  to delete and recreate this implementation fil
//

import Foundation
import CoreData

extension User {

    @NSManaged var name: String?
    @NSManaged var sessions: NSSet?

}
```

ONLY GENERATE THIS ONE TIME

Attribute Type String

NSMANAGED OBJECT SUBCLASS

```
17 13:04:52.939 MyFavoriteApps[2503:fb03] Unresolved error Error Domain=NSCocoaErrorDomain Code=134100 "The operation couldn't be completed. (Cocoa e
' UserInfo=0x6b63560 {metadata=<CFBasicHash 0x6b67b20 [0x173fb48]>{type = immutable dict, count = 7,
=>
<CFString 0x6b67ee0 [0x173fb48]>{contents = "NSStoreModelVersionIdentifiers"} = <CFArray 0x6b68340 [0x173fb48]>{type = immutable, count = 1, values =
<CFString 0x173acd8 [0x173fb48]>{contents = ""}

<CFString 0x6b67f10 [0x173fb48]>{contents = "NSPersistenceFrameworkVersion"} = <CFNumber 0x6b67dd0 [0x173fb48]>{value = +386, type = kCFNumberSInt64T
<CFString 0x6b682e0 [0x173fb48]>{contents = "NSStoreModelVersionHashes"} = <CFBasicHash 0x6b683c0 [0x173fb48]>{type = immutable dict, count = 1,
=>
<CFString 0x6b68360 [0x173fb48]>{contents = "Event"} = <CFData 0x6b68370 [0x173fb48]>{length = 32, capacity = 32, bytes =
46d30e7f32c2cc809958add1e7 ... 846e97d7af01cc79}


<CFString 0x10f8ad8 [0x173fb48]>{contents = "NSStoreUUID"} = <CFString 0x6b680c0 [0x173fb48]>{contents = "838007D0-B9E9-4CE6-9379-4394F44B051D"}
<CFString 0x10f8978 [0x173fb48]>{contents = "NSStoreType"} = <CFString 0x10f8988 [0x173fb48]>{contents = "SQLite"}
<CFString 0x6b67d90 [0x173fb48]>{contents = "_NSAutoVacuumLevel"} = <CFString 0x6b68420 [0x173fb48]>{contents = "2"}
 <CFString 0x6b68310 [0x173fb48]>{contents = "NSStoreModelVersionHashesVersion"} = <CFNumber 0x6b22940 [0x173fb48]>{value = +3, type = kCFNumberSInt3

=The model used to open the store is incompatible with the one used to create the store}, {
data =     {
NSPersistenceFrameworkVersion = 386;
NSStoreModelVersionHashes =         {
    Event = <5431c046 d30e7f32 c2cc8099 58add1e7 579ad104 a3aa8fc4 846e97d7 af01cc79>;
};
NSStoreModelVersionHashesVersion = 3;
NSStoreModelVersionIdentifiers =         (
    ""
);
NSStoreType = SQLite;
NSStoreUUID = "838007D0-B9E9-4CE6-9379-4394F44B051D";
'_NSAutoVacuumLevel" = 2;

on = "The model used to open the store is incompatible with the one used to create the store";
```

Need to regenerate managed objects after change

NSMANAGEDOBJECT SUBCLASS IN IOS10

NSMANAGEDOBJECT SUBCLASS IN IOS10

- Xcode 8 `codegen` will create subclasses for you at compile time
  - You don't ever see them

NSMANAGEDOBJECT SUBCLASS IN IOS10

- Thats it 😀

BREAK TIME 😀

A SIMPLE CORE DATA APPLICATION

A SIMPLE CORE DATA APPLICATION

- Session tracker application
  - Session and User model
  - Load and fetch data
  - Show in table with NSFetchedResultsController
  - Use UISearchBar to live filter results

A SIMPLE CORE DATA APPLICATION

A SIMPLE CORE DATA APPLICATION

DATA MODEL

DATA MODEL

2017-CoreDataBasics 〉 iPhone 7 Plus | 2017-CoreDataSessionTracker Clean **Succeeded**

SeardchTableViewController.swift | _017_CoreDataBasics.xcdatamodel

2017-CoreDataSessionTracker 〉 2017-CoreDataBasics 〉 _017_CoreDataBasics.xcdatamodeld 〉 _017_CoreDataBasics.xcdatamodel 〉 User

- 2017-CoreDa...essionTracker M
  - 2017-CoreDataBasics
    - AppDelegate.swift M
    - ViewController.swift M
    - Main.storyboard M
    - SeardchTa...troller.swift A
    - TableView...troller.swift A
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - _017_Core...datamodeld M
    - PodcastAPI.playground M
  - Products

ENTITIES
- Session
- User

FETCH REQUESTS

CONFIGURATIONS
- Default

Attributes

Attribute	Type
age	Integer 16
name	String
timestamp	Date

Relationships

Relationship	Destination	Inverse
sessions	Session	user

Fetched Properties

Fetched Property	Predicate

DATA MODEL

TableView...troller.swift A
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_017_Core...datamodeld M
PodcastAPI.playground M
Products
Default
User
Attributes
age
name
timestamp
Relationships
sessions
Session
Attributes
active
startTime
timestamp
uuid
Relationships
user
same model
two view

ADDING DATA

ADDING DATA

User+CoreDataProperties.swift
User.swift
2016-CoreDataStack
AppDelegate.swift
ViewController.swift
Main.storyboard
TableViewController.swift
SearchTableViewController.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
_016_CoreDataS...k.xcdatamodeld
Products
FETCH REQUESTS
CONFIGURATIONS
Default
User
Attributes
gender
name
Relationships
sessions
Session
Attributes
active
startTime
timestamp
uuid
Relationships
user

ADDING DATA

- 2017-CoreDa...essionTracker M
  - 2017-CoreDataBasics
    - AppDelegate.swift M
    - ViewController.swift M
    - Main.storyboard M
    - SeardchTa...troller.swift A
    - TableView...troller.swift A
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - _017_Core...datamodeld M
    - PodcastAPI.playground M
  - Products

```swift
func applicationWillTerminate(_ application: UIApplication) {…}

// MARK: - Core Data stack

lazy var persistentContainer: NSPersistentContainer = {
    /*
     The persistent container for the application. This implementation
     creates and returns a container, having loaded the store for the
     application to it. This property is optional since there are legitimate
     error conditions that could cause the creation of the store to fail.
    */
    let container = NSPersistentContainer(name: "_017_CoreDataBasics")
    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping
                application, although it may be useful during development.

            /*
             Typical reasons for an error here include:
             * The parent directory does not exist, cannot be created, or disallows writing.
             * The persistent store is not accessible, due to permissions or data protection when the device is locked.
             * The device is out of space.
             * The store could not be migrated to the current model version.
             Check the error message to determine what the actual problem was.
             */
            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveContext () {
    print("Saving...")
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
            dump()


        } catch {
            // Replace this implementation with code to handle the error appropriately.
            // fatalError() causes the application to generate a crash log and terminate. You should not use this function in a shipping
                application, although it may be useful during development.
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

app delegate

ADDING DATA

- AppDelegate.swift M
- ViewController.swift M
- Main.storyboard M
- SeardchTa...troller.swift A
- TableView...troller.swift A
- Assets.xcassets
- LaunchScreen.storyboard
- Info.plist
- _017_Core...datamodeld M
- PodcastAPI.playground M
- Products

```swift
//
//  Created by T. Andrew Binkowski on 4/16/17.
//  Copyright © 2017 T. Andrew Binkowski. All rights reserved.
//

import UIKit
import CoreData

class ViewController: UIViewController {

  var context: NSManagedObjectContext? {
    return (UIApplication.shared.delegate as? AppDelegate)?
      .persistentContainer.viewContext
  }

  override func viewDidLoad() {
    super.viewDidLoad()

    // Create some people
    var person = User(context: context!)
    person.name = "Homer"
    person.age = 42
    person.timestamp = Date() as NSDate

    person = User(context: context!)
    person.name = "Marge"
    person.age = 41
    person.timestamp =  Date() as NSDate

    // 4
    (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
  }


  override func viewWillAppear(_ animated: Bool) {•••}


  @IBAction func tapAddPeople(_ sender: UIButton) {•••}
}
```

Find our persistent container

Create objects

Save to disk

ADDING DATA

```
override func viewDidLoad() {
  super.viewDidLoad()

  // Create some people
  let user1 = User(context: context!)
  user1.name = "Homer"
  user1.age = 42
  user1.timestamp = Date() as NSDate

  // Create some people
  let user2 = User(context: context!)
  user2.name = "Marge"
  user2.age = 42
  user2.timestamp = Date() as NSDate

  // Let's create a bunch sessesions and add them to difference users
  for i in 0...10 {
    let session = Session(context: context!)
    // Just for fun, make the even sessions true
    session.active = (i % 2 == 0) ? true : false
    session.timestamp = Date() as NSDate
    session.user = user2
    session.user = user1
  }

  // Save
  (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

ADD SOME RANDOM Sessions

ADDING DATA

```
@IBAction func tapAddPeople(_ sender: UIButton) {

  let person = User(context: context!)
  person.name = "Random Person"
  person.age = 42
  person.timestamp = Date() as NSDate

  let session = Session(context: context!)
  session.active = true
  session.timestamp = Date() as NSDate
  session.user = person

  (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
}
```

ADD SOME RANDOM DATA

ADDING DATA

```
func saveContext () {
    print("Saving...")
    let context = persistentContainer.viewContext
    if context.hasChanges {
        do {
            try context.save()
            dump()


        } catch {
            // Replace this implementation with code to handle the error appropriat
            // fatalError() causes the application to generate a crash log and term
                application, although it may be useful during development.
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}
```

FETCHING DATA

FETCHING DATA

SUBTITLE

- Fetch managed objects by sending a request to managed object context
- Use the NSFetchedData method
- Optional
  - Set a predicate (NSPredicate)
  - Set a sort descriptor (NSSortDescriptor)

Language: Swift Objective-C Both On This Pag

NSFetchRequest

An instance of `NSFetchRequest` describes search criteria used to retrieve data from a persistent st

Inheritance

- NSObject
  - NSPersistentStoreRequest
    - NSFetchRequest

Conforms To

NSCoding
NSCopying
NSObject

Import Stateme

`@import CoreD`

Availability

Available in iOS

Managing the Fetch Request's Entity

`+ fetchRequestWithEntityName:`

`- initWithEntityName:`

`entityName` *Property*

`entity` *Property*

`includesSubentities` *Property*

Fetch Constraints

`predicate` *Property*

`fetchLimit` *Property*

FETCHING DATA

```
// Fetch our person objects
let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
do {
  let fetchedEntities = try context?.fetch(personFetchRequest)
  // Log it out
  for user in fetchedEntities! {
    print("> \(user.name!) ##########")
    for session in user.sessi
      print(session)
    }
  }

} catch {
  // Do something in response
}
```

```
> Homer #########
<Session: 0x610000094eb0> (entity: Session; id: 0xd000000000c00000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p48> ; data: {
    active = 0;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x610000097340> (entity: Session; id: 0xd000000000c40000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p49> ; data: {
    active = 1;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x6100000977a0> (entity: Session; id: 0xd000000000a40000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p41> ; data: {
    active = 1;
    startTime = nil;
    timestamp = "2017-04-17 17:23:01 +0000";
    user = "0xd000000000340002 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/User/p13>";
    uuid = nil;
})
<Session: 0x610000095f90> (entity: Session; id: 0xd000000000c80000 <x-coredata://C2A8F07B-ABC6-4C9D-A2BC-2AAECA21079F/Session/p50> ; data: {
    active = 0;
    startTime = nil;
```

FETCHING DATA

```
// Fetch our person objects
let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
do {
  let fetchedEntities = try context?.fetch(personFetchRequest)
  // Log it out
  for user in fetchedEntities! {
    print("> \(user.name!) ##########")
    for session in user.sessions! {
      print(session)
    }
  }

} catch {
  // Do something in response to error condition
}
```

setup fetch request

execute fetch

our in-memory data store

FETCHING DATA

```
SFetchRequest<User>(entityName: "Use
tchSize = 10
 NSSortDescriptor(key: "timestamp", a
criptors = [dateDescriptor]
```

- Set sort descriptors for returned results
- Takes array of sort descriptors as arguments

FETCHING DATA

```
// Print data store to the console
func dump() {

  let personFetchRequest: NSFetchRequest<User> = User.fetchRequest()
  personFetchRequest.fetchLimit = 20
  personFetchRequest.relationshipKeyPathsForPrefetching = ["session"]
  personFetchRequest.propertiesToFetch = ["timestamp"]

  do {
    let context = persistentContainer.viewContext

    let fetchedEntities = try context.fetch(personFetchRequest)
    print("Dump")
    for user in fetchedEntities {
      print("# \(user.name!) ##########")
      for session in user.sessions! {
        print(session)
      }
    }

  } catch {
    // Do something in response to error condition
  }

}
```

prepare for efficiency

limit for efficiency

FETCHING DATA

- NSPredicate defines logical conditions
  - Constrain a search
  - In-memory filtering
- Examples
  - grade == "7"
  - firstName like "Bob"
  - name CONTAINS[cd] “son”
  - (firstName begins with 'M') AND (lastName like 'Adderley'

Language:

NSFetchRequest

An instance of `NSFetchRequest` describes search criteria used to

Inheritance

- `NSObject`
  - `NSPersistentStoreRequest`
    - `NSFetchRequest`

Conforms To

`NSCoding`
`NSCopying`
`NSObject`

Managing the Fetch Request’s Entity

`+ fetchRequestWithEntityName:`

`- initWithEntityName:`

`entityName` *Property*

`entity` *Property*

`includesSubentities` *Property*

Fetch Constraints

`predicate` *Property*

`fetchLimit` *Property*

FETCHING DATA

- NSPredicate can be applied to NSArrays, NSDictionary and NSFetchRequest
- Predicate format string syntax
  - Regular expression
- NSCompoundPredicate
  - Chain multiple predicates together

Language:

NSFetchRequest

An instance of `NSFetchRequest` describes search criteria used to

Inheritance

- `NSObject`
  - `NSPersistentStoreRequest`
    - `NSFetchRequest`

Conforms To

`NSCoding`
`NSCopying`
`NSObject`

Managing the Fetch Request's Entity

`+ fetchRequestWithEntityName:`

`- initWithEntityName:`

`entityName` *Property*

`entity` *Property*

`includesSubentities` *Property*

Fetch Constraints

`predicate` *Property*

`fetchLimit` *Property*

FETCHING DATA

```
hFetchRequest: NSFetchRequest<User> = User.fetchRequest()
chRequest.fetchLimit = 20
chRequest.relationshipKeyPathsForPrefetching = ["session"]
chRequest.propertiesToFetch = ["timestamp"]

xact match only
chRequest.predicate = NSPredicate(format: "firstName == %@ && age == %@", argumentArray: ["Homer",42])

everything
chRequest.predicate = NSPredicate(format: "TRUEPREDICATE")

matching string
chRequest.predicate = NSPredicate(format: "name CONTAINS[cd] %@", text)
```

- Predicate examples

DELETING DATA

DELETING DATA

- Fetch the objects
- Iterate through them and delete them
- iOS8 new feature
  - Batch delete
- Save the managed object context
  - Deleting removes from memory
  - Commit the changes to save to the on-disk store

```
let personFetchRequest: NSFetch
personFetchRequest.fetchLimit =
personFetchRequest.relationship
personFetchRequest.propertiesTo

// Fetch everything
personFetchRequest.predicate = 

do {
  let context = persistentConta

  let fetchedEntities = try con
  print("Dump")
  for user in fetchedEntities {
    print("# \(user.name!) ####
    for session in user.session
      print(session)
    }
    // You could delete the use
    context.delete(user)

  }

} catch {
  // Do something in response t
}
```

DELETING DATA

- Fetch the objects
- Iterate through them and delete them
- iOS8 new feature
  - Batch delete
- Save the managed object context
  - Deleting removes from memory
  - Commit the changes to save to the on-disk store

```
chRequest: NSFetchRequest<User> = User.fetchRequest()
quest.fetchLimit = 20
quest.relationshipKeyPathsForPrefetching = ["session"]
quest.propertiesToFetch = ["timestamp"]

ything
quest.predicate = NSPredicate(format: "TRUEPREDICATE")


 = persistentContainer.viewContext

Entities = try context.fetch(personFetchRequest)
")
 fetchedEntities {
\(user.name!) ##########")
on in user.sessions! {
ession)

uld delete the user
elete(user)
```

DELETING DATA

```
let fetchedEntities = try context.fetch(personFetchRequest)
print("Dump")
for user in fetchedEntities {
  print("# \(user.name!) ##########")
  for session in user.sessions! {
    print(session)
  }
  // You could delete the user
  context.delete(user)

}
```

- Delete managed objects
  - Fetch the objects
  - Iterate through them and delete them
  - Save the managed object context 😃

DELETING DATA

Cascading deletes

Attributes

Attribute	Type
gender	String
name	String

Relationships

Relationship	Destination	Inverse
sessions	Session	user

Fetched Properties

Fetched Property	Predicate

No Action
✓ Nullify
Cascade
Deny

Relationship

Name: sessions
Properties: Transient, Optional
Destination: Session
Inverse: user
Delete Rule: Nullify
Type: To Many
Arrangement: Ordered
Count: Unbounded, Minimum
Unbounded, Maximum
Advanced: Index in Spotlight
Store in External Record File

User Info

Key	Value

Versioning

Hash Modifier: Version Hash Modifier
Renaming ID: Renaming Identifier

CORE DATA IN A TABLE

CORE DATA AND TABLES

- Core Data can be tightly integrated with tables views
  - Reduce memory overhead
  - Improve response times
- Loads data for cells being shown
- NSFectchedResultsController has direct methods for creating
  - Index path access
  - Section key path
  - Section groups
  - Index titles

NSFetchedResultsController Cla

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.f
Availability	Available in iOS 3.0 and later.
Companion guide	Core Data Programming Guide
Declared in	NSFetchedResultsController.h
Related sample code	CoreDataBooks DateSectionTitles iPhoneCoreDataRecipes TopSongs

Overview

You use a fetched results controller to efficiently manage the resul
`UITableView` object.

While table views can be used in several ways, this object is primar
expects its data source to provide cells as an array of sections mad
fetch request that specifies the entity, an array containing at least
`NSFetchedResultsController` efficiently analyzes the result of t
in the result set, and for the index.

In addition, `NSFetchedResultsController` provides the followin

- It optionally monitors changes to objects in its associated mana
  delegate (see "The Controller's Delegate").
- It optionally caches the results of its computation so that if the
  to be repeated (see "The Cache").

A controller thus effectively has three modes of operation, determi
is set.

CORE DATA AND TABLES

- NSFetchedResultsController managed results from Core Data fetch for UITableView object
  - Provides the “dataSource” for the UITableView
- Create an instance of NSFetchedResultsController
  - Entity
  - NSArray of sort ordering
  - Optional: NSPredicate
- Generates
  - Rows, sections and index

NSFetchedResultsController Cla

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreData.f
Availability	Available in iOS 3.0 and later.
Companion guide	Core Data Programming Guide
Declared in	NSFetchedResultsController.h
Related sample code	CoreDataBooks DateSectionTitles iPhoneCoreDataRecipes TopSongs

Overview

You use a fetched results controller to efficiently manage the resul
`UITableView` object.

While table views can be used in several ways, this object is primar
expects its data source to provide cells as an array of sections mad
fetch request that specifies the entity, an array containing at least
`NSFetchedResultsController` efficiently analyzes the result of t
in the result set, and for the index.

In addition, `NSFetchedResultsController` provides the followin

- It optionally monitors changes to objects in its associated mana
  delegate (see “The Controller’s Delegate”).
- It optionally caches the results of its computation so that if the
  to be repeated (see “The Cache”).

A controller thus effectively has three modes of operation, determi
is set.

CORE DATA AND TABLES

- Create an instance of NSFetchedResultsController
  - Entity
  - NSArray of sort ordering
  - Optional: NSPredicate
- Generates
  - Rows, sections and index

```
jectContext? {
shared.delegate as? AppDelegate)?.persistentContainer.


results controller in a block; we will only be using
his view controller
ontroller: NSFetchedResultsController<User> = {

etchRequest<User>(entityName: "User")
hSize = 10
SSortDescriptor(key: "timestamp", ascending: false)
iptors = [dateDescriptor]

troller = NSFetchedResultsController(fetchRequest: fet
                                     managedObjectCont
                                     sectionNameKeyPat
                                     cacheName: nil)

ler.delegate = self
Controller
```

CORE DATA AND TABLES

```
///
var context: NSManagedObjectContext? {
  return (UIApplication.shared.delegate as? AppDelegate)?.persistentContainer.viewContext
}

/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {

  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

CORE DATA AND TABLES

```
///
var context: NSManagedObjectContext? {
  return (UIApplication.shared.delegate as? AppDelegate)?.persistentContainer.viewContext
}

/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {

  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

CORE DATA AND TABLES

```
// MARK: - Table view data source

override func numberOfSections(in tableView: UITableView) -> Int {
  if let sections = fetchedResultsController.sections {
    return sections.count
  }
  return 0
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return currentSection.numberOfObjects
  }
  return 0

}

override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return String(currentSection.name.characters.prefix(1))
  }
  return nil
}
```

CORE DATA AND TABLES

Even handle sectioning a table for you

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
  let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)

  // Configure the cell...
  let user = fetchedResultsController.object(at: indexPath)
  cell.textLabel?.text = user.name
  cell.detailTextLabel?.text = user.timestamp?.description
  return cell
}
```

CORE DATA AND TABLES

```
override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
  if let sections = fetchedResultsController.sections {
    let currentSection = sections[section]
    return currentSection.name
  }
  return nil
}
```

```
let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                           managedObjectContext: self.context!,
                                                           sectionNameKeyPath: "name",
                                                           cacheName: nil)
```

Section headers are created automatically in NSSectionInfo

CORE DATA AND TABLES

- Many more options
  - Display
  - Performance
- NSFetchedResultsController delegate
  - Watch changes to the table
  - Update

SEARCH A STORE

SEARCH A STORE

- There are many options for searching
- Things to consider
  - Data source (internet, local)
  - Filtering algorithm (eg. character limit)
  - Filtering options
  - Where do you want the search bar
  - Data to fetch (all vs. some)

SEARCH A STORE

- Live filtering against a Core Data store
  - UISearch bar that takes text input
  - Initially the search predicate matches everything (all results)
  - As the user types, update the search predicate
  - Canceling or clearing the field returns the predicate to everything

SEARCH A STORE

SUBTITLE

- UISearchController introduced in iOS9

-

SEARCH A STORE

```
/// Search bar
let searchController = UISearchController(searchResultsController: nil)


// MARK: - Lifecycle
override func viewDidLoad() {

  super.viewDidLoad()
  // Load up the search controller.  There are many options on how it displays
  // that do not effect the behavior.
  searchController.searchResultsUpdater = self
  searchController.dimsBackgroundDuringPresentation = false
  searchController.searchBar.searchBarStyle = .prominent
  //searchController.hidesNavigationBarDuringPresentation = false

  // Show a scope bar for additional filtering parameters
  searchController.searchBar.scopeButtonTitles = ["Like Beer", "Likes Wine"]
  searchController.searchBar.delegate = self

  // Place the search bar in the header
  tableView.tableHeaderView = searchController.searchBar
  //navigationItem.titleView = searchController.searchBar
  definesPresentationContext = true

  fetch()

}
```

Add a UISearchBar and set preferences

SEARCH A STORE

```
// Load up the search controller.  There are many options on how it displays
// that do not effect the behavior.
searchController.searchResultsUpdater = self
searchController.dimsBackgroundDuringPresentation = false
searchController.searchBar.searchBarStyle = .Prominent
//searchController.hidesNavigationBarDuringPresentation = false

// Show a scope bar for additional filtering parameters
searchController.searchBar.scopeButtonTitles = ["Like Beer", "Likes Wine"]
searchController.searchBar.delegate = self

// Place the search bar in the header
tableView.tableHeaderView = searchController.searchBar
//navigationItem.titleView = searchController.searchBar
definesPresentationContext = true

// Offset the table to hide the search bar
//tableView.contentOffset = CGPoint(x:0, y:44)
```

SEARCH A STORE

```
/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {
  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  fetchRequest.predicate = self.currentPredicate
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()

/// Predicate variables; set default value to everything
var currentPredicate: NSPredicate = NSPredicate(format: "TRUEPREDICATE")

/// Search bar
let searchController = UISearchController(searchResultsController: nil)
```

SEARCH A STORE

```
///
/// UISearchResultsDelegate
///


extension SearchTableViewController: UISearchResultsUpdating {

  /// Update the search results based on the text in the search bar.  We could
  /// also use the scope bar, but we are not in this example
  /// - paremeter searchController: The search controller sending the message
  ///
  func updateSearchResults(for searchController: UISearchController) {
    // Use this is you want to use the scope to filter on as well
    //let searchBar = searchController.searchBar
    //let scope = searchBar.scopeButtonTitles![searchBar.selectedScopeButtonIndex]
    let scope = "Dummy Text to hide scope for now"

    filterResultsForSearchText(searchController.searchBar.text!,scope: scope)
  }
}

///
/// UISearchBarDelegate
///
extension SearchTableViewController: UISearchBarDelegate {

  func searchBar(_ searchBar: UISearchBar, selectedScopeButtonIndexDidChange selectedScope: Int) {
    filterResultsForSearchText(searchBar.text!, scope: searchBar.scopeButtonTitles![selectedScope])
  }
}
```

Get the text

SEARCH A STORE

```
///
/// UISearchResultsDelegate
///


extension SearchTableViewController: UISearchResultsUpdating {

  /// Update the search results based on the text in the search bar.  We could
  /// also use the scope bar, but we are not in this example
  /// - paremeter searchController: The search controller sending the message
  ///
  func updateSearchResults(for searchController: UISearchController) {
    // Use this is you want to use the scope to filter on as well
    //let searchBar = searchController.searchBar
    //let scope = searchBar.scopeButtonTitles![searchBar.selectedScopeButtonIndex]
    let scope = "Dummy Text to hide scope for now"

    filterResultsForSearchText(searchController.searchBar.text!,scope: scope)
  }
}

///
/// UISearchBarDelegate
///
extension SearchTableViewController: UISearchBarDelegate {

  func searchBar(_ searchBar: UISearchBar, selectedScopeButtonIndexDidChange selectedScope: Int) {
    filterResultsForSearchText(searchBar.text!, scope: searchBar.scopeButtonTitles![selectedScope])
  }
}
```

Get the text

SEARCH A STORE

```
/// different times: initialization, search, cleared search.
func fetch() {
  print("🐶 Fetch with predicate: \(self.fetchedResultsController.fetchRequest.predicate!)")
  do {
    fetchedResultsController.fetchRequest.predicate = currentPredicate
    try self.fetchedResultsController.performFetch()
  } catch {
    let nserror = error as NSError
    print("Unresolved error \(nserror), \(nserror.userInfo)")
  }
}

/// Filter the data store by the text query and scope in the search bar
/// - Note: Scope is not actually used here.
func filterResultsForSearchText(_ text: String, scope: String = "All") {
  print("\t🎹SearchBar: \(text) Scope: \(scope)")

  // Match everything unless there is text in the search bar
  if text.characters.count == 0 {
    self.currentPredicate = NSPredicate(format: "TRUEPREDICATE")
  } else {
    self.currentPredicate = NSPredicate(format: "name CONTAINS[cd] %@", text)
  }
  fetch()
```

Fetch with updated predicate

Update the predicate

SEARCH A STORE

```
/// Compute the fetched results controller in a block; we will only be using
/// one controller for this view controller
lazy var fetchedResultsController: NSFetchedResultsController<User> = {
  let fetchRequest = NSFetchRequest<User>(entityName: "User")
  fetchRequest.fetchBatchSize = 10
  fetchRequest.predicate = self.currentPredicate
  let dateDescriptor = NSSortDescriptor(key: "timestamp", ascending: false)
  fetchRequest.sortDescriptors = [dateDescriptor]

  let _fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest,
                                                             managedObjectContext: self.context!,
                                                             sectionNameKeyPath: "name",
                                                             cacheName: nil)
  _fetchedResultsController.delegate = self
  return _fetchedResultsController

}()
```

SOME THINGS THAT WILL MAKE YOU CRAZY

Add a new attribute
- Need to remove the old dataStore (delete app)

- 2016-CoreDataStack
  - Session+Core...roperties.swift
  - Session.swift
  - User+CoreDat...roperties.swift
  - User.swift
  - 2016-CoreDataStack
    - AppDelegate.swift
    - ViewController.swift
    - Main.storyboard
    - TableViewController.swift
    - Assets.xcassets
    - LaunchScreen.storyboard
    - Info.plist
    - _016_CoreDa...cdatamodeld
  - Products

```swift
//
//  User+CoreDataProperties.swift
//  2016-CoreDataStack
//
//  Created by T. Andrew Binkowski on 4/11/16.
//  Copyright © 2016 The University of Chicago, Department of Computer Science. All rights reserved.
//
//  Choose "Create NSManagedObject Subclas                                    nu
//  to delete and recreate this implementa                                    .
//

import Foundation
import CoreData

extension User {

    @NSManaged var name: String?
    @NSManaged var gender: String?
    @NSManaged var sessions: NSSet?

}
```

MANUALLY ADD THIS

```
Memory                    21 MB
Disk                      Zero KB/s
Network                   Zero KB/s
Thread 1 Queue: c...ead (serial)
    0 __pthread_kill
    2 abort
    3 AppDelegate.(persistent...
    4 AppDelegate.persistentS...
    5 AppDelegate.(managed...
    6 AppDelegate.managedO...
    7 ViewController.viewDidL...
    36 UIApplicationMain
    37 main
    38 start
    39 start
Thread 2 Queue: c...ger (serial)
Thread 3
Thread 4
Thread 5
```

```
61    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
62        // The persistent store coordinator for the application. This implementation creates and returns a coordinator, having a
63        // Create the coordinator and store
64        let coordinator = NSPersistentStoreCoordinator(managedObjectModel: self.managedObjectModel)
65        let url = self.applicationDocumentsDirectory.URLByAppendingPathComponent("SingleViewCoreData.sqlite")
66        var failureReason = "There was an error creating or loading the application's saved data."
```

2016-CoreDataStack 〉 Thread 1 〉 3 AppDelegate.(persistentStoreCoordinator.getter).(closure #1)

```
initialize the application's saved data" UserInfo={NSLocalizedDescription=Failed to initialize the application's saved data,
NSLocalizedFailureReason=There was an error creating or loading the application's saved data., NSUnderlyingError=0x7fde0d51c70
{Error Domain=NSCocoaErrorDomain Code=134100 "(null)" UserInfo={metadata={
    NSPersistenceFrameworkVersion = 641;
    NSStoreModelVersionHashes =     {
        Session = <b8c99403 71b4692f 46c2e939 8d386149 79b301f6 b11259b7 eddce342 d50b90eb>;
        User = <9640ffde 03148fb5 73041b6a 4b9ba1d8 b7d53a4c 143bc592 108b0c83 a9cdc902>;
    };
    NSStoreModelVersionHashesVersion = 3;
    NSStoreModelVersionIdentifiers =     (
        ""
    );
    NSStoreType = SQLite;
    NSStoreUUID = "3B4CC88F-A685-409F-A963-BD53AADCCEA9";
    "_NSAutoVacuumLevel" = 2;
}, reason=The model used to open the store is incompatible with the one used to create the store}}}, [NSLocalizedDescription:
Failed to initialize the application's saved data, NSLocalizedFailureReason: There was an error creating or loading the
application's saved data., NSUnderlyingError: Error Domain=NSCocoaErrorDomain Code=134100 "(null)" UserInfo={metadata={
    NSPersistenceFrameworkVersion = 641;
    NSStoreModelVersionHashes =     {
        Session = <b8c99403 71b4692f 46c2e939 8d386149 79b301f6 b11259b7 eddce342 d50b90eb>;
        User = <9640ffde 03148fb5 73041b6a 4b9ba1d8 b7d53a4c 143bc592 108b0c83 a9cdc902>;
    };
    NSStoreModelVersionHashesVersion = 3;
    NSStoreModelVersionIdentifiers =     (
        ""
```

In a production application there is quite a bit of housekeeping to do to make changes to you data model or data.

Typically, requires making complete copies of a store and making changes in background and updating.

CORE DATA FIRESIDE CHAT

SUBTITLE

- Core Data is powerful, but frustrating
  - API is getting better but is still verbose and unforgiving
  - Odd integration with InterfaceBuilder
  - Non-trivial schema migration can be difficult
  - Coordinating with backend can add complexity

- Best use is in concert with TableViewController using NSFetchedResultsController
  - Most of the core code is boilerplate from templates
  - Best table performance possible

CORE DATA FIRESIDE CHAT

SUBTITLE

- Shipping your app with preloaded database
- Option 1:
  - Include raw data
  - Create and populate a datastore on first run
- Option 2:
  - Create auxiliary program to populate a datastore
  - Include in your application bundle
  - Copy to documents directory on first run

CORE DATA FIRESIDE CHAT

SUBTITLE

- Finding the file on your Mac
  - -(NSURL*)applicationDocumentsDirectory
  - ~/Library/ApplicationSupport/Simulator
- Loading in sqlite3
  - `> sqlite3 file.db`
- Versioning you Core Data store
  - Easy to add new entity or entity attributes
  - Difficult to change relationships
- Mapping Model to perform large scale changes

ICLOUD

CORE DATA FIRESIDE CHAT

- As of iOS10 it is deprecated

ICLOUD

- 6028 Startown Road, Maiden, North Carolina

THE UNIVERSITY OF CHICAGO

• SESSION 3

S

© 2016 T.A. Binkowski

REVIEW

NOTIFICATIONS

NOTIFICATIONS

NOTIFICATIONS

No more interruptions
Notifications appear at the top of your screen and disappear quickly.

One-swipe access
Swipe down to reveal Notification Center.

"User Notification" not broadcast notifications

See more at a glance
View stocks and weather, too.

NOTIFICATIONS

- Only one application can run in the foreground
- Notifications give you flexibility in how to interact with users when they're not running your app
  - Messages
  - Game turn
  - Updates available

NOTIFICATIONS

- Re-engage your users
  - Notifications can launch your app

NOTIFICATIONS

- Notification styles
  - Modal
  - Alert
  - Sounds (default, custom)
  - Badges
  - Silent
    - Happens in background
    - “Trigger event”

NOTIFICATIONS

- Local Notifications
  - Scheduled by an application
  - Delivered by the iOS on the device
  - “Schedule a notification”

NOTIFICATIONS

- Push Notifications
  - Remote notifications
  - Sent by application's remote server to Apple Push Notification Service (APNS)
  - "Register a notification"

NOTIFICATIONS

NOTIFICATIONS

Local

Push

Appears the same on devices

NOTIFICATIONS

- Handling notifications in app delegate
  - application:didFinishLaunchingWithOptions:
  - application:didReceiveRemoteNotification:
  - application:didReceiveLocalNotification:

Options: local or push
UIApplicationLaunchOptionsLocalNotificationKey

NOTIFICATIONS

- iOS8 introduced interactive notifications
  - Engage with application without opening it

NOTIFICATIONS

- iOS8 introduced interactive notifications
  - Engage with application without opening it
- iOS9 allows text input (eg. messages) from interactive notifications
- iOS10 unified the API for working with UserNotification Framework

Permissions

PERMISSIONS

- Remember that users are always in control
  - Keep them informed
  - Do not bother them
    - Of course you are trying to get their permission to bombard them with notifications)
  - Make sure you app still functions without notifications
    - Inform the user of consequences
    - “To be informed on the latest info, enable notifications in Settings” don’t forget to show a link to settings

PERMISSIONS

```
let options: UNAuthorizationOptions = [.alert, .sound];
let center = UNUserNotificationCenter.current()
center.requestAuthorization(options: options) {
    (granted, error) in
    if !granted {
        print("Something went wrong")
    }
}
```

- Registering for notifications

PERMISSIONS

Standard Dialogue

PERMISSIONS

- Consideration on notifications
  - Users may not accept to receive notifications
  - Users can change preferences in settings
- Be prepared to handle all potential use cases

```
center.getNotificationSettings { (settings) in
    if settings.authorizationStatus != .authorized {
        // Notifications not allowed
    }
}
```

PERMISSIONS

- The old way
  - “Go to Settings > Privacy > Location > OUR_APP”

PERMISSIONS

PERMISSIONS

```
// Check if we have authorization to send notifications
guard let settings = UIApplication.sharedApplication().currentUserNotificationSettings() else { return }

// If we don't, let the use know and give them the option of going directly
// to the Settings screen for the application
if settings.types == .None {

  // Create an alert view controller
  let alertController = UIAlertController(title: "😒",
                                          message: "The notification permission was not authorized. Please enable it in Sett
                                          preferredStyle: .Alert)
  // Create to go to settings
  let settingsAction = UIAlertAction(title: "Settings", style: .Default) { (alertAction) in
    if let appSettings = NSURL(string: UIApplicationOpenSettingsURLString) {
      UIApplication.sharedApplication().openURL(appSettings)
    }
  }
  alertController.addAction(settingsAction)

  // Create cancel action that does nothing
  let cancelAction = UIAlertAction(title: "Cancel", style: .Cancel, handler: nil)
  alertController.addAction(cancelAction)

  // Show the alert and exit early
  presentViewController(alertController, animated: true, completion: nil)
  return
}
```

New Way

PERMISSIONS

```
// Check if we have authorization to send notification
guard let settings = UIApplication.sharedApplication()
if settings.types == .None {
  let alertController = UIAlertController(title: "😒",
                                          preferredSty

  let settingsAction = UIAlertAction(title: "Settings"

    if let appSettings = NSURL(string: UIApplicationOp
      UIApplication.sharedApplication().openURL(appSet
    }
  }
  alertController.addAction(settingsAction)

  let cancelAction = UIAlertAction(title: "Cancel", st
  alertController.addAction(cancelAction)

  presentViewController(alertController, animated: tru
  return
}
```

```
}

uthorized. Please enable it in Settings to continue.",
```

PERMISSIONS

Local Notifications

LOCAL NOTIFICATIONS

- Local notifications are delivered by the iOS device

LOCAL NOTIFICATIONS

- Messages are handled differently depending on the state of the application
  - Running
  - Background
  - Terminated
- Developer responsibility to handle each case

LOCAL NOTIFICATIONS

Launch app from notification

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {

  if let notification:UILocalNotification = launchOptions?[UIApplicationLaunchOptionsLocalNotificationKey] as? UILocalNotification {
    print("Launch from local notification: \(notification)")
  }

  return true
}
```

- Test for type of notification when launching from terminated state
- Different behavior
  - Go to a view controller
  - Set some date

LOCAL NOTIFICATIONS

Handle notification from suspended

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

LOCAL NOTIFICATIONS

Have to set delegate early

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

LOCAL NOTIFICATIONS

- Remember to handle all the states (if necessary)
  - App in foreground
  - App suspended
  - App terminated

Scheduling Local Notifications

SCHEDULING NOTIFICATIONS

- Notification request contain content and a trigger

- `title` : String containing the primary reason for the alert.
- `subtitle` : String containing an alert subtitle (if required)
- `body` : String containing the alert message text
- `badge` : Number to show on the app's icon.
- `sound` : A sound to play when the alert is delivered. Use `UNNotificationSound.default()` or create a custom sound from a file.
- `launchImageName` : name of a launch image to use if your app is launched in response to a notification.
- `userInfo` : A dictionary of custom info to pass in the notification
- `attachments` : An array of `UNNotificationAttachment` objects. Use to include audio, image or video content.

Content sent with notification

SCHEDULING NOTIFICATIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()
```

- Notification request contain content and a trigger

SCHEDULING NOTIFICATIONS

- Triggers
  - Time
  - Calendar
  - Location

SCHEDULING NOTIFICATIONS

- Schedule a notification for a number of seconds later

Time trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

_ = UNTimeIntervalNotificationTrigger(timeInterval: 300,
                                      repeats: false)
```

SCHEDULING NOTIFICATIONS

- Trigger at a specific date and time
- The trigger is created using a date components object which makes it easier for certain repeating intervals

Calendar trigger

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk"
content.sound = UNNotificationSound.default()

let date = Date(timeIntervalSinceNow: 3600)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,],
                                                  from: date)

let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate,
                                            repeats: false)
```

SCHEDULING NOTIFICATIONS

```
let triggerDaily = Calendar.current.dateComponents([.hour,.minute,.second,], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDaily, repeats: true)

let triggerWeekly = Calendar.current.dateComponents([.weekday,.hour,.minute,.second,], from: date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerWeekly, repeats: true)
```

- Repeating notifications

SCHEDULING NOTIFICATIONS

Location trigger

```
let trigger = UNLocationNotificationTrigger(triggerWithRegion:region, repeats:false)
```

- Trigger when a user enters or leaves a geographic region
- The region is specified through a CoreLocation `CLRegion`
- You need to get permissions just like `CoreLocation`

SCHEDULING NOTIFICATIONS

```
UNUserNotificationCenter.current().
    getPendingNotificationRequests {
    requests in
    print(requests)
}
```

- List all the current notifications

SCHEDULING NOTIFICATIONS

```
let center = UNUserNotificationCenter.current()
center.removeAllPendingNotificationRequests()
```

- Delete notifications

SCHEDULING NOTIFICATIONS

```
@IBAction func tapDelete(_ sender: Any) {
  listNotification()

  let center = UNUserNotificationCenter.current()
  center.removeAllPendingNotificationRequests()

  listNotification()
}
```

- Delete notifications

Notification Actions

NOTIFICATION ACTIONS

- Define custom actions for your notifications

NOTIFICATION ACTIONS

```
// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:
    [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
                                      actions: [snoozeAction,deleteAction],
                                      intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

NOTIFICATION ACTIONS

```
let content = UNMutableNotificationContent()
content.title = "Don't forget"
content.body = "Buy some milk CALENDAR"
content.sound = UNNotificationSound.default()
content.categoryIdentifier = "CalendarCategory"

let date = Date(timeIntervalSinceNow: 10)
let triggerDate = Calendar.current.dateComponents([.year,.month,.day,.hour,.minute,.second,], from:
    date)
let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

// Actions
let snoozeAction = UNNotificationAction(identifier: "Snooze", title: "Snooze", options: [])
let deleteAction = UNNotificationAction(identifier: "DeleteAction", title: "Delete", options:
    [.destructive])
let category = UNNotificationCategory(identifier: "CalendarCategory",
                                      actions: [snoozeAction,deleteAction],
                                      intentIdentifiers: [], options: [])
UNUserNotificationCenter.current().setNotificationCategories([category])
```

NOTIFICATION ACTIONS

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {

  print("AppDelegate: Did Receive: \(response)")

  // Determine the user action
   switch response.actionIdentifier {
  case UNNotificationDismissActionIdentifier:
    print("Dismiss Action")
  case UNNotificationDefaultActionIdentifier:
    print("Default")
  case "Snooze":
    print("Snooze")
  case "DeleteAction":
    print("Delete")
  default:
    print("Unknown action")
  }
  completionHandler()
}
```

Switch on the actionIdentifier

Push Notifications

PUSH NOTIFICATIONS

- Sent from Apple's servers to device
- Apple Push Notification Service (APNS) is the gateway for push notification
  - SSL certificate
  - Provisioning
  - Networking code
  - Device tokes

image via Big Nerd Ranch iOS Programming

PUSH NOTIFICATIONS

1. An app enables push notification (the user has to confirm that he wishes to receive these notifications)
2. The app receives a “device token”
3. The app sends the device token to your server
4. The server sends a push notification to the Apple Push Notification Service
5. APNS sends the push notification to the user’s device

PUSH NOTIFICATIONS

PUSH NOTIFICATION

- Requirements for push notifications
  - iOS device
  - Paid iOS Developer membership
  - Server
    - Ability to run background processes
    - Install SSL certificates
    - Make outing TLS connections on non-standard ports

PUSH NOTIFICATION

- Many third party solutions for push notifications
  - Parse
  - Firebase
  - App Engine
  - ...

PUSH NOTIFICATIONS

```
{"aps": {
    "badge": 100,
    "alert": "Push Notification Test",
    "sound": "SoundFile.aif"},
    "type":"websiteUrl",
    "data":"http://mySite.com"
}
```

Manage the badges in your code (not incremental)

Include sound files in your bundle

- Push notification payload
  - JSON dictionary
  - <256 bytes

Pass any key-values in the notification

PUSH NOTIFICATION

- Receiving notification require app delegate methods to be implemented
- Code path is the same for local notification

```
/// Called when the app is in the background
func userNotificationCenter(_ center: UNUserNotificationCenter,
                            didReceive response: UNNotificationResponse,
                            withCompletionHandler completionHandler: @escaping () -> Void) {
  print(response)
  if response.actionIdentifier == UNNotificationDismissActionIdentifier {
    // The user dismissed the notification without taking action
  }
  else if response.actionIdentifier == UNNotificationDefaultActionIdentifier {
    // The user launched the app
    print("AppDelegate: Did Receive: \(response)")
  }
  completionHandler()
}
```

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

PUSH NOTIFICATION

- There are many third party services for setting up push notifications

nomad / houston

Watch 88 | Star 2,620 | Fork 224

Code | Issues 27 | Pull requests 5 | Projects 0 | Wiki | Insights

Apple Push Notifications; No Dirigible Required http://nomad-cli.com

houston notifications ruby cli nomad apns

157 commits | 1 branch | 19 releases | 32 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

mpvosseller authored and dankimio committed on Jan 24 Set default passphrase to nil instead of the empty string. (#158) Latest commit 592bbeb on Jan 24

bin	Set default passphrase to nil instead of the empty string. (#158)	a month ago
lib	Set default passphrase to nil instead of the empty string. (#158)	a month ago
spec	Mutable content: update README, add spec (#137)	a year ago
.gitignore	Update .gitignore	a year ago
Gemfile	Update code style (#134)	a year ago
LICENSE	Updating copyright	3 years ago
README.md	README: use unregistered_devices instead of devices (#156)	5 months ago
Rakefile	Rakefile: add Bundler tasks, add default task	a year ago
houston.gemspec	Update dependencies	a year ago

README.md

HOUSTON

NOTIFICATIONS WRAP-UP

- Notifications keep users engage by opening your app
- Notifications are not guaranteed to be delivered by Apple
- Tips and tricks:
  - Push was important before multi-tasking in iOS4, but many uses can be implemented as local notifications
  - Third-party option for remote notifications
  - Use CloudKit for remote notifications via subscriptions
- Don't abuse notifications…this is one thing that Apple seems to regularly enforce in the App Store Guidelines
  - "Push notifications can not be used for advertising"