iOS

Additional elements
View Controllers

• Custom subclasses of UIViewController to present app’s content
  – Content view controllers
    • UIViewController, UITableViewController, UICollectionViewController
  – Container view controllers
    • UINavigationController, UITabBarController, UISplitViewController
Content View Controllers

- Present content on the screen using a view or a group of views organized into a view hierarchy
  - Each controller is responsible for managing all the views in a single view hierarchy
  - A single controller should never manage multiple screens
UILTableViewController

- A built-in controller designed for managing tabular data
  - Manages a table view and adds support for many standard table-related behaviors
    - A table view presents data in a single-column list of multiple rows and is a means for displaying and editing hierarchical lists of information
  - Has a pointer to the root view of the interface, but it also has a separate pointer to the table view
Table view styles

- It is just an attribute we can set on the table view in the storyboard
UICollectionViewController

- Represents a view controller whose content consists of a collection view
  - Displays an ordered collection of data
- Similar to a table view displays data using a combination of cell, layout, and supplementary views
  - can display items in a grid or in a custom layout that you design
  - Each cell must be an instance of UICollectionViewCell
**UINavigationController**

- Presents data organized hierarchically
- Provides methods for managing a stack-based collection of content view controllers
UITabBarController

- Used to divide your app into distinct modes of operation
- The tab bar has multiple tabs, each represented by a child view controller
UITabBarController

- What if we had more than 4 View Controllers?
  - A More button appears
- Everything happens automatically
UIPageViewController

- Implements a page layout
  - One can flip between discrete pages of content as if it were a book
  - Each content page is provided by a content view controller
UISplitViewController

- It presents a master-detail interface
  - Changes in the primary view controller (the master) drive changes in a secondary view controller (the detail)
Address Book UI framework

The ContactsUI Framework

Classes  Protocols

Classes

- UIViewController
  - The UIViewController class provides the infrastructure for managing the views of your iOS apps.
- CNContactPickerController
  - The CNContactPickerController class creates a controller object that manages the contacts picker view.
- CNContactViewController
  - The CNContactViewController class implements the view to display a contact.

Protocols

- ABPeoplePickerNavigationController
- ABPersonViewController
- ABNewPersonViewController
- ABUnknownPersonViewController

This chapter includes some short code listings you can use as a starting point. For a fully worked example, see QuickTips.
More scenes
## Segue types

<table>
<thead>
<tr>
<th>Name</th>
<th>Interface Builder Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>🗺️</td>
<td>Present the content in the detail or master area depending on the content of the screen. If the app is displaying a master and detail view, the content is pushed onto the detail area. If the app is only displaying the master or the detail, the content is pushed on top of the current view controller stack.</td>
</tr>
<tr>
<td>Show Detail</td>
<td>🗺️</td>
<td>Present the content in the detail area. If the app is displaying a master and detail view, the new content replaces the current detail. If the app is only displaying the master or the detail, the content replaces the top of the current view controller stack.</td>
</tr>
<tr>
<td>Present Modally</td>
<td>🗺️</td>
<td>Present the content modally. There are options to choose a presentation style (UIModalPresentationStyle) and a transition style (UIModalTransitionStyle).</td>
</tr>
<tr>
<td>Present as Popover</td>
<td>🗺️</td>
<td>Present the content as a popover anchored to an existing view. There is an option to specify the possible directions of the arrow shown on one edge of the popover view (UIPopoverArrowDirection). There is also an option to specify the anchor view.</td>
</tr>
<tr>
<td>Custom</td>
<td>🗺️</td>
<td>A custom segue enabling you to write your own behaviors.</td>
</tr>
<tr>
<td>Push (Deprecated)</td>
<td>🗺️</td>
<td>Present the content by pushing it onto the current stack of view controllers.</td>
</tr>
<tr>
<td>Modal (Deprecated)</td>
<td>🗺️</td>
<td>Present the content modally on top of the existing screen. The options are the same as Present Modally.</td>
</tr>
<tr>
<td>Popover (Deprecated)</td>
<td>🗺️</td>
<td>Present the content as a popover. The options are the same as Present as Popover.</td>
</tr>
<tr>
<td>Replace (Deprecated)</td>
<td>🗺️</td>
<td>Replace the top view controller on the screen with the new content.</td>
</tr>
</tbody>
</table>
**Show** - Pushes the destination view controller onto the navigation stack, moving the source view controller out of the way (destination slides overtrop from right to left), providing a back button to navigate back to the source - on all devices

*Example*: Navigating inboxes/folders in Mail

**Show Detail** - Replaces the detail/secondary view controller when in a [UISplitViewController](https://developer.apple.com/library/content/documentation/UIKit/Reference/UISplitViewController_class/index.html) with no ability to navigate back to the previous view controller

*Example*: In Mail on iPad in landscape, tapping an email in the sidebar replaces the view controller on the right to show the new email

**Present Modally** - Presents a view controller in various different ways as defined by the Presentation option, covering up the previous view controller - most commonly used to present a view controller that animates up from the bottom and covers the entire screen on iPhone, but on iPad it's common to present it in a centered box format overtrop that darkens the underlying view controller

*Example*: Tapping the + button in Calendar on iPhone

**Popover Presentation** - When run on iPad, the destination appears in a small popover, and tapping anywhere outside of this popover will dismiss it. On iPhone, popovers are supported as well but by default if it performs a Popover Presentation segue, it will present the destination view controller modally over the full screen.

*Example*: Tapping the + button in Calendar on iPad (or iPhone, realizing it is converted to a full screen presentation as opposed to an actual popover)
Unwind Segue

- Can be used to “unwind” the navigation stack and specify a destination to go back to.
- Unwind segues always segue from the source or current view controller to an existing view controller, a view controller that is already present in the navigation hierarchy.
Unwind Segue (concrete process)

• Choose the view controller that should appear onscreen at the end of an unwind segue
• Define an unwind action method on the view controller you chose
  — @IBAction func myUnwindAction(unwindSegue: UIStoryboardSegue)
• Navigate to the view controller that initiates the unwind action
• Control-click the button (or other object) that should initiate the unwind segue
  — This element should be in the view controller you want to dismiss
• Drag to the Exit object at the top of the view controller scene
• Select your unwind action method from the relationship panel
We can get notified of the raw touch events or we can react to certain, predefined “gestures”

Gestures are recognized by class UIGestureRecognizer (abstract)
  – TapGestureRecognizer, UIPinchGestureRecognizer, UIRotationGestureRecognizer, UISwipeGestureRecognizer, UIPanGestureRecognizer, UIScreenEdgePanGestureRecognizer, UILongPressGestureRecognizer

There are two sides to using a gesture recognizer
  – Adding a gesture recognizer to a UIView to ask it to recognize that gesture
  – Providing the implementation of a method to “handle” that gesture when it happens
Internationalization

• The ability of code to adapt to local conventions in different parts of the world
  – This includes things like language, time and date formatting, and currency symbols and separators

• We must create a localization for each locale we want to support
  – A localization is a collection of strings, currency formats, graphics, sounds, and other resources that are specific to one locale
  – We declare supported localizations at the project level
**Deployment Target**

iOS Deployment Target: 8.1

**Configurations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Based on Configuration File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>No Configurations Set</td>
</tr>
<tr>
<td>Release</td>
<td>No Configurations Set</td>
</tr>
</tbody>
</table>

Use Release for command-line builds

**Localizations**

<table>
<thead>
<tr>
<th>Language</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>Development Language 2 Files Localized</td>
</tr>
</tbody>
</table>

Languages:
- French (fr)
- German (de)
- Chinese (Simplified) (zh-Hans)
- Chinese (Traditional) (zh-Hant)
- Japanese (ja)
- Spanish (es)
- Spanish (Mexico) (es-MX)
- Italian (it)
- Dutch (nl)
- Korean (ko)
- Portuguese (pt)
@IBAction func increment(_ sender: UIButton) {
    let text = NSLocalizedString("Value: ", comment: "english")
    counter += 1
    counterLabel.text = text + "(counter)"
}
Localizable.strings

// Localizable.strings
TestApp

/*
Created by Luciano Baresi on 1/6/15.
Copyright (c) 2015 Luciano Baresi. All rights reserved.
*/

"Value: " = "Valeur: ";
Data Management
Three options

- File System
  - Based on the UNIX file system
- SQLite
  - Embedded DBMS (like in Android)
- Core Data
  - Object-oriented database
  - Powerful framework in iOS
File system

• Interactions with the file system are limited to the directories inside the app’s sandbox
  — Exception: when an app uses public system interfaces to access things such as the user’s contacts or music

• During installation of a new app, the installer creates a number of containers for the app
  — Each container has a specific role
<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppName.app</td>
<td>This is the app’s <strong>bundle</strong>. This directory contains the app and all of its resources. You cannot write to this directory. To prevent tampering, the bundle directory is signed at installation time. Writing to this directory changes the signature and prevents your app from launching. You can, however, gain read-only access to any resources stored in the apps bundle. For more information, see the Resource Programming Guide. The contents of this directory are not backed up by iTunes. However, iTunes does perform an initial sync of any apps purchased from the App Store.</td>
</tr>
<tr>
<td>Documents/</td>
<td>Use this directory to store user-generated content. The contents of this directory can be made available to the user through file sharing; therefore, this directory should only contain files that you may wish to expose to the user. The contents of this directory are backed up by iTunes.</td>
</tr>
<tr>
<td>Documents/Inbox</td>
<td>Use this directory to access files that your app was asked to open by outside entities. Specifically, the Mail program places email attachments associated with your app in this directory. Document interaction controllers may also place files in it. Your app can read and delete files in this directory but cannot create new files or write to existing files. If the user tries to edit a file in this directory, your app must silently move it out of the directory before making any changes. The contents of this directory are backed up by iTunes.</td>
</tr>
<tr>
<td>Library/</td>
<td>This is the top-level directory for any files that are not user data files. You typically put files in one of several standard subdirectories. iOS apps commonly use the <strong>Application Support</strong> and <strong>Caches</strong> subdirectories; however, you can create custom subdirectories. Use the <strong>Library</strong> subdirectories for any files you don’t want exposed to the user. Your app should not use these directories for user data files. The contents of the <strong>Library</strong> directory (with the exception of the <strong>Caches</strong> subdirectory) are backed up by iTunes. For additional information about the Library directory and its commonly used subdirectories, see The Library Directory Stores App-Specific Files.</td>
</tr>
<tr>
<td>tmp/</td>
<td>Use this directory to write temporary files that do not need to persist between launches of your app. Your app should remove files from this directory when they are no longer needed; however, the system may purge this directory when your app is not running. The contents of this directory are not backed up by iTunes.</td>
</tr>
</tbody>
</table>
Files and directories

• Directories
  — You must use the methods of FileManager
  — A process can create directories anywhere it has permission to do so

• Files
  — When specifying the location of files, you can use either NSURL or NSString objects
    • The use of the NSURL class is generally preferred
  — Two parts: creation of a record for the file in the file system and filling the file with content

• To copy items around the file system, you use class FileManager
iCloud Storage API

The diagram illustrates how iCloud Storage API works. It shows a cloud symbol with a document icon representing the cloud storage. Two Syncing Daemons connect to the cloud, each managing local storage (a desktop and a smartphone). Arrows indicate the synchronization process between the cloud and local storage devices.
How it works

• A document is not moved to iCloud immediately
  — First, it is moved from its current location in the file system to a local system-managed directory where it can be monitored by the iCloud service
  — After that transfer, the file is transferred to iCloud and to the user’s other devices as soon as possible
• Apps are expected to use file coordinator objects to perform all changes
  — File coordinators mediate changes between your app and the daemon that facilitates the transfer of the document to and from iCloud
  — The file coordinator acts like a locking mechanism for the document
• Class NSDocument helps manage documents in iCloud
iCloud Key-Value Data Store

• Helps store discrete values in iCloud for app preferences, app configuration, or app state
File Coordinators and File Presenters

• NSFileCoordinator coordinates the reads and writes performed by our app and the sync daemon on the same document
  — We use presenters in conjunction with an NSFileCoordinator object to coordinate access to a file or directory among the objects of our application and between our application and other processes
  — Instances of NSFileCoordinator are meant to be used on a per-file-operation basis
• The NSFilePresenter protocol should be implemented by objects that allow the user to view or edit the content of files or directories
  — The job of a file presenter is to protect the integrity of its own data structures
  — Class NSDocument is an example of a file presenter that tracks changes to its underlying file or file package
What apps should do to work with iCloud

- Manage each document in iCloud using a file presenter
  - After creating a file presenter, register it
  - Before deleting a file presenter, unregister it

- All file-related operations must be performed through a file coordinator object
  - Create an instance of class NSFileCoordinator and initialize it with the file presenter object that is about to perform the file operation
  - Use the methods of the NSFileCoordinator object to read/write the file
  - When we are done with the operations, release the file coordinator object
SQLite

Different wrappers available (on GitHub)
SQLite.swift

- Swift 3 interface to SQLite 3
- Class Connection helps establish Database connections
  - We can create a writable database in our app’s Documents directory
  - If we omit the path, SQLite.swift will provision an in-memory database
  - SQLite will attempt to create the database file if it does not already exist
  - We can also bundle a database with our app, and then we can establish a read-only connection to it
Core Data
Core Data

- Is a framework that we use to manage the objects in the model layer of our applications
- Provides generalized and automated solutions to common tasks associated with life cycle and graph management of objects, including persistence
- Is not a relational database, and is not an Object-Relational Mapping (ORM) to a relational database either
Let's start

- AppDelegate.swift file
  - quite a significant amount of new code (Core Data stack)
  - Most of these are setting up the Core Data stack and the defaults are fine for now

```swift
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.
    */
```
import UIKit
import CoreData

class ViewController: UIViewController {

    func getContext () -> NSManagedObjectContext {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        return appDelegate.persistentContainer.viewContext
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view

        // Print it to the console
        print(getContext())
    }
}
Entities

- An Entity in the code becomes an NSManagedObject
Entities

- All attributes are objects
- Attributes can be accessed easily
  - NSManagedObject offers valueForKey and setValue
/retrieve the entity that we just created
let entity = NSEntityDescription.entity(forEntityName: "LogItem", in: context)

let item = NSManagedObject(entity: entity!, insertInto: context)

//set the entity values
item.setValue("Wrote Core Data Tutorial", forKey: "title")
item.setValue("Wrote and post", forKey: "text")

//save the object
do {
    try context.save()
    print("saved!")
} catch {
    print("Could not save \(error)")
}
Relationships

- The type of relationships can be either toOne or to Many