0.25"[set status bar style per view controller](https://stackoverflow.com/questions/19013975/ios-7-and-later-set-status-bar-style-per-view-controller)

1. Set "View controller-based status bar appearance" (UIViewControllerBasedStatusBarAppearance) to YES in your Info.plist. (YES is the default, so you can also just leave this value out of your plist.)
2. In your viewDidLoad method, call [self setNeedsStatusBarAppearanceUpdate].
3. Implement preferredStatusBarStyle, returning the status bar style that you want for this view controller.

- (UIStatusBarStyle) preferredStatusBarStyle { return UIStatusBarStyleLightContent; }

# General Settings of your Target

In the General tab for the preferences for your project’s target, you’ll find two fields that adjust the default behaviour of your status bar :

* Status Bar Style — choose between Light(i.e. light text) or Default(i.e. dark text)
* Hide status bar — a checkbox.

When you set firstArray = secondArray; you're actually setting their *pointers* to the same value, rather than setting the items in the first array to be the same as the items in the second array. This means that you have made firstArray and secondArray the exact same object, rather than having the exact same values. If you just want to have a *copy* of the second array into the first, you just need to specify that.

firstArray = [secondArray mutableCopy]; firstArray = [secondArray copy]; // non-mutable copy

I am initializing two NSMutableArrays namely:

NSMutableArray \*firstArray = [[NSMutableArray alloc] init]; NSMutableArray \*secondArray = [[NSMutableArray alloc] init];

firstArray has some values in it and secondArray has some values in it too.

Then I assigned firstArray to secondArray:

firstArray = secondArray;

Now if I make some changes in firstArray it will also affect secondArray. For example, if I replace or remove a value at certain index from firstArray then that value will also get removed from the secondArray.

These fields are derived from the info plist, which would be an alternative approach for adjusting these fields:

# Info plist fields

If you’re using info.plist to adjust the default behaviour of your status bar, you’ll want to add rows for:

## Status bar is initially hidden(UIStatusBarHidden)

Choose from either YES or NO.

## Status bar style(UIStatusBarStyle).

Choose from either:

* default(UIStatusBarStyleDefault)
* light(UIStatusBarStyleLightContent)
* [UIStatusBarStyleDefault](https://developer.apple.com/documentation/uikit/uistatusbarstyle/uistatusbarstyledefault?language=objc)
* A dark status bar, intended for use on light backgrounds.
* [UIStatusBarStyleLightContent](https://developer.apple.com/documentation/uikit/uistatusbarstyle/uistatusbarstylelightcontent?language=objc)
* A light status bar, intended for use on dark backgrounds.
* [UIStatusBarStyleDarkContent](https://developer.apple.com/documentation/uikit/uistatusbarstyle/uistatusbarstyledarkcontent?language=objc)
* A dark status bar, intended for use on light backgrounds.
* Beta
* [~~UIStatusBarStyleBlackTranslucent~~](https://developer.apple.com/documentation/uikit/uistatusbarstyle/uistatusbarstyleblacktranslucent?language=objc)
* A transparent black style.
* Deprecated
* [~~UIStatusBarStyleBlackOpaque~~](https://developer.apple.com/documentation/uikit/uistatusbarstyle/uistatusbarstyleblackopaque?language=objc)
* An opaque black style.
* Deprecated

# Set properties on the UIAppDelegate

Some will suggest you to set properties on the UIAppDelegate:

UIApplication.shared.statusBarStyle = .default //Set Style UIApplication.shared.isStatusBarHidden = false //Set if hidden

These could be set for example in the didFinishLaunchingWithOptionsmethod of your app’s UIAppDelegate, or anywhere you like really — for example, if you wanted to change this temporarily for a view controller you could call this in the viewDidAppear and viewDidDisappear methods.

Oh wait, sorry. I forgot to mention — the [isStatusBarHidden](https://developer.apple.com/documentation/uikit/uiapplication/1622982-isstatusbarhidden) and [statusBarStyle](https://developer.apple.com/documentation/uikit/uiapplication/1622988-statusbarstyle) properties were **deprecated**in iOS 9. Bah!

# Override properties in your view controller classes

Within your view controller classes, you can override the [prefersStatusBarHidden](https://developer.apple.com/documentation/uikit/uiviewcontroller/1621440-prefersstatusbarhidden) and [preferredStatusBarStyle](https://developer.apple.com/documentation/uikit/uiviewcontroller/1621416-preferredstatusbarstyle) properties:

override var prefersStatusBarHidden: Bool {   
 return true   
}   
override var preferredStatusBarStyle: UIStatusBarStyle {   
 return .lightContent   
}

These methods return values by default (prefersStatusBarHidden defaults to false, preferredStatusBarStyle defaults to .default), so if you change either of these default values, when the user navigates to another view controller, if the new view controller doesn’t have its own implementation of these methods, it will revert to these defaults. This is a difference in behavior to setting properties on the app delegate, which changes them for the duration of your app, well, at least until you specifically request they change again.

## Changing mid view controller

By the way, you can use these properties to make this change mid view controller too. Just set up a variable, let’s call it statusBarHidden, to contain the current preference for the status bar.  
We could then return this variable in the view controller property prefersStatusBarHidden:

var statusBarHidden = true   
override var prefersStatusBarHidden: Bool {   
 return statusBarHidden   
}

Now if we change the statusBarHidden property, nothing will happen yet. We need to tell the system that we want the status bar to change. We can do this in a property observer:

var statusBarHidden = true {   
 didSet(newValue) {   
 setNeedsStatusBarAppearanceUpdate()   
 }   
}

Now we should be able to change the statusBarHidden property, and the status bar will automatically update. For example, we could have a button that toggles the status bar, connected to an IBAction:

@IBAction func toggleStatusBar(\_ sender: Any) {   
 statusBarHidden.toggle()   
 //If you’re pre Swift 4.2, use:   
 //statusBarHidden = !statusBarHidden   
}

# Navigation Controllers

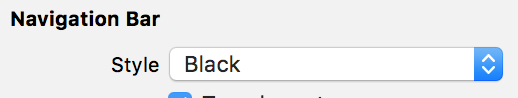
I’m afraid we’re not done on the mysterious case of status bars — embed your view controller in a navigation controller, and if you have View controller-based status bar appearance set to YES, suddenly the navigation controller now takes over control of the style of the status bar, and your view controller’s preferredStatusBarStyle property will be ignored.

If you want to adjust the style of the status bar in a navigation controller, you’ll need to do this in the navigation bar’s style property, either in code, eg:

override func viewDidAppear(\_ animated: Bool) {   
 navigationController?.navigationBar.barStyle = .black   
}

Or in the storyboard:

https://miro.medium.com/max/60/0*XsgZEg_oadZ8OVMh.png?q=20



To make things ultra-confusing, if you want, for example, that your status bar has white text, view controllers **not** embedded in a navigation controller will need to request the .lightContentstyle (i.e. light colored writing), and view controllers that **are** embedded in a navigation controller will need to request the .blackstyle (i.e. a black colored nav bar).

Don’t forget, if you have View controller-based status bar appearance set to NO, any changes you make to the navigation bar style will be ignored.

[UIInterfaceOrientationMaskPortrait](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmaskportrait?language=objc)

The view controller supports a portrait interface orientation.

[UIInterfaceOrientationMaskLandscapeLeft](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmasklandscapeleft?language=objc)

The view controller supports a landscape-left interface orientation.

[UIInterfaceOrientationMaskLandscapeRight](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmasklandscaperight?language=objc)

The view controller supports a landscape-right interface orientation.

[UIInterfaceOrientationMaskPortraitUpsideDown](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmaskportraitupsidedown?language=objc)

The view controller supports an upside-down portrait interface orientation.

[UIInterfaceOrientationMaskLandscape](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmasklandscape?language=objc)

The view controller supports both landscape-left and landscape-right interface orientation.

[UIInterfaceOrientationMaskAll](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmaskall?language=objc)

The view controller supports all interface orientations.

[UIInterfaceOrientationMaskAllButUpsideDown](https://developer.apple.com/documentation/uikit/uiinterfaceorientationmask/uiinterfaceorientationmaskallbutupsidedown?language=objc)

The view controller supports all but the upside-down portrait interface orientation.

# UIInterfaceOrientationMaskAll

The view controller supports all interface orientations.

UIInterfaceOrientationMaskAll = (UIInterfaceOrientationMaskPortrait | UIInterfaceOrientationMaskLandscapeLeft | UIInterfaceOrientationMaskLandscapeRight | UIInterfaceOrientationMaskPortraitUpsideDown)

Bitbucket is more than just Git code management. Bitbucket gives teams one place to plan projects, collaborate on code, test, and deploy.

@try {

brr = [((PT\_ButtonData\*)[\_buttons objectAtIndex:i]).btnSelections objectAtIndex:0];

}

// @catch (NSException \*exception) {

// NSLog(@"%@", exception.reason);

// NSLog(@"Char at index %d cannot be found", index);

// }

@finally {

NSLog(@"Finally condition");

if ([brr.DisplayString isEqualToString:@""]) {

\_cdtData = [NSMutableArray new];

return \_cdtData;

}

Windows Vista (2007), for instance, had 50 million lines of code, while Windows 7 shaved that figure down to 40 million.  lines of source code there are in OS X and how this has grown over the last few releases? Wikipedia suggests that Tiger has 86 million lines of source code - is this correct?

# UITableViewAutomaticDimension

A constant representing the default value for a given dimension.

Return this value from your table view's delegate methods when you want the table view to choose a default value for the given dimension. For example, if you return this constant from [tableView:heightForHeaderInSection:](https://developer.apple.com/documentation/uikit/uitableviewdelegate/1614855-tableview?language=objc) or [tableView:heightForFooterInSection:](https://developer.apple.com/documentation/uikit/uitableviewdelegate/1614967-tableview?language=objc), the table view uses a height that fits the value returned from [tableView:titleForHeaderInSection:](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614850-tableview?language=objc) or [tableView:titleForFooterInSection:](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614994-tableview?language=objc), if the title is not nil.

// Set automatic dimensions for row height // Swift 4.2 onwards table.rowHeight = UITableView.automaticDimension table.estimatedRowHeight = UITableView.automaticDimension // Swift 4.1 and below table.rowHeight = UITableViewAutomaticDimension table.estimatedRowHeight = UITableViewAutomaticDimension

// UITableViewAutomaticDimension calculates height of label contents/text func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat { // Swift 4.2 onwards return UITableView.automaticDimension // Swift 4.1 and below return UITableViewAutomaticDimension }

The situation is slightly more complicated when you want some cells to be autosized and others not. To make this work you should add two methods to your table view controller, heightForRowAt and estimatedHeightForRowAt, then make them both return the special value UITableView.automaticDimension for the cells you want to be sized automatically.

func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat { if indexPath.section == 0 { return UITableView.automaticDimension } else { return 40 } }

override func tableView(\_ tableView: UITableView, estimatedHeightForRowAt indexPath: IndexPath) -> CGFloat { if indexPath.section == 0 { return UITableView.automaticDimension } else { return 40 } }

A few ins-and-outs aside, all you really have to do is:

1. Use Auto Layout for the UI elements inside the table view cells.
2. Set the table view rowHeight to UITableViewAutomaticDimension.
3. Set the estimatedRowHeight or implement the height estimation delegate method.

To get dynamic cell heights working properly, you need to create a custom table view cell and set it up with the right Auto Layout constraints.

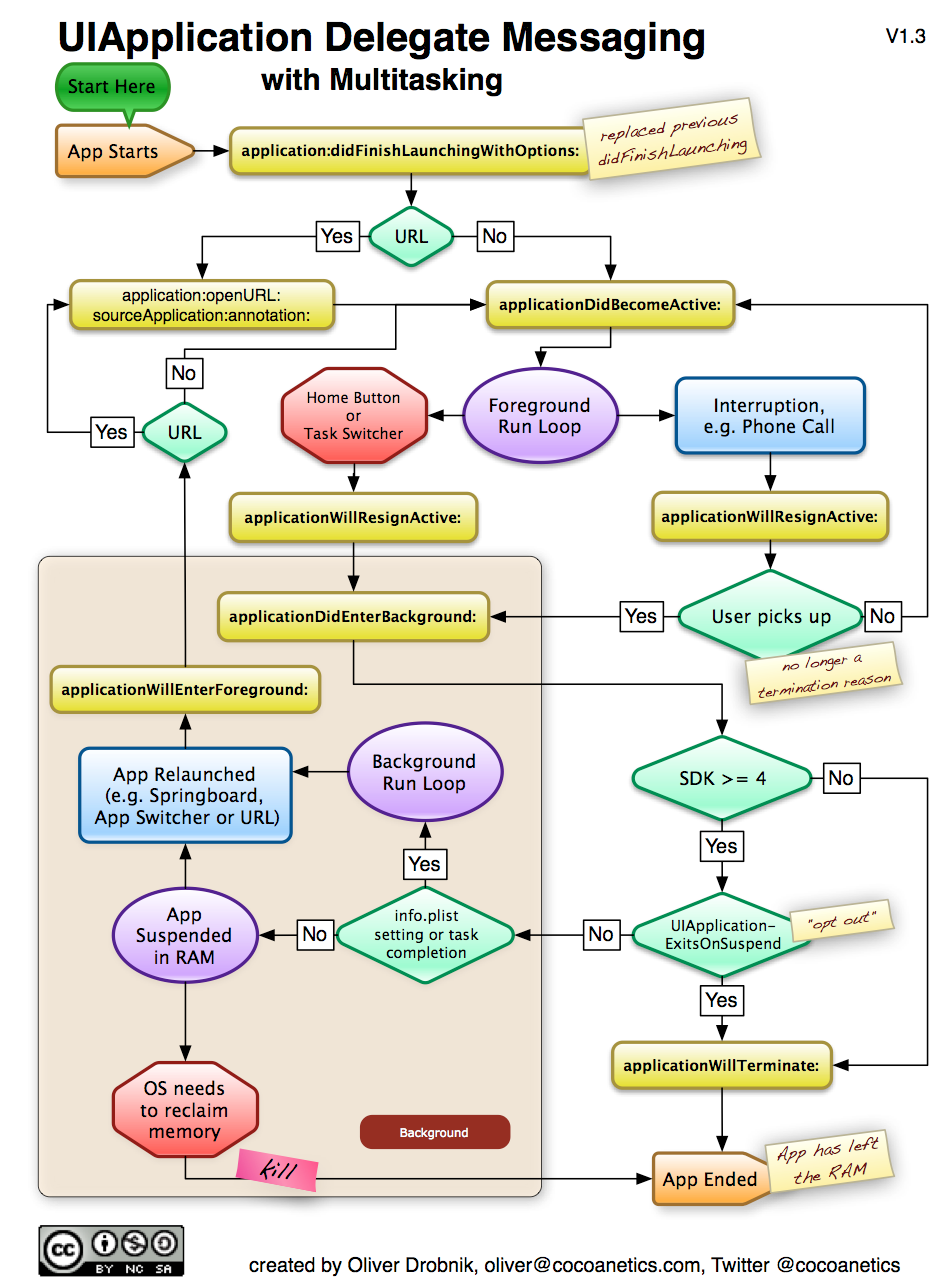
When you set the row height as UITableViewAutomaticDimension, the table view is told to use the Auto Layout constraints and the contents of its cells to determine each cell’s height.

It uses AutoLayout to calculate the height of each cell. If the subviews don't have explicit height NSLayoutConstraints, it will use their intrinsic content sizes. Your example using a UITableViewdoesn't work because it's intrinsic content height is zero. If you want it to work, you will need to use a height constraint that matches the .contentSize.height

A delegate object is an object that gets notified when the object to which it is connected reaches certain events or states. In this case, the Application Delegate is an object which receives notifications when the UIApplication object reaches certain states. In many respects, it is a specialized one-to-one Observer pattern.

This means that the "area of concern" for the AppDelegate is handling special UIApplication states. The most important of these are:

* applicationDidFinishLaunching: - good for handling on-startup configuration and construction
* applicationWillTerminate: - good for cleaning up at the end



# [How do I get a reference to the app delegate in Swift?](https://stackoverflow.com/questions/24046164/how-do-i-get-a-reference-to-the-app-delegate-in-swift)

### Swift 3.x (Introduced with Xcode 8)

let appDelegate = UIApplication.shared.delegate as! AppDelegate

let aVariable = appDelegate.someVariable

1. The app is launched, either explicitly by the user or implicitly by the system.
2. The Xcode-provided main function calls UIKit's [UIApplicationMain](https://developer.apple.com/documentation/uikit/1622933-uiapplicationmain?language=objc) function.
3. The [UIApplicationMain](https://developer.apple.com/documentation/uikit/1622933-uiapplicationmain?language=objc) function creates the [UIApplication](https://developer.apple.com/documentation/uikit/uiapplication?language=objc) object and your app delegate.
4. UIKit loads your app's default interface from the main storyboard or nib file.
5. UIKit calls your app delegate's [application:willFinishLaunchingWithOptions:](https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1623032-application?language=objc)method.
6. UIKit performs state restoration, which calls additional methods of your app delegate and view controllers.
7. UIKit calls your app delegate's [application:didFinishLaunchingWithOptions:](https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1622921-application?language=objc)method .

When initialization is complete, the system uses either your scene delegates or app delegate to display your UI and manage the life cycle for your app.

When an iOS app is launched the first thing called is

**application: willFinishLaunchingWithOptions:-> Bool**. This method is intended for initial application setup. Storyboards have already been loaded at this point but state restoration hasn’t occurred yet.

***Launch***

* **application: didFinishLaunchingWithOptions: -> Bool**is called next. This callback method is called when the application has finished launching and restored state and can do final initialization such as creating UI.
* **applicationWillEnterForeground:**is called after **application:didFinishLaunchingWithOptions:**or if your app becomes active again after receiving a phone call or other system interruption.
* **applicationDidBecomeActive:**is called after **applicationWillEnterForeground:**to finish up the transition to the foreground.

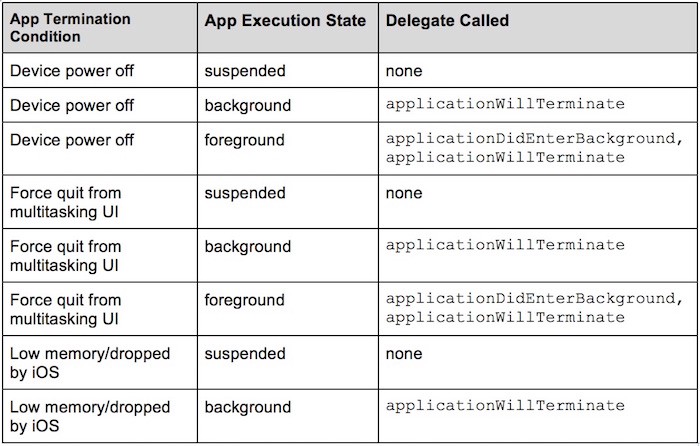
***Termination***

* **applicationWillResignActive:**is called when the app is about to become inactive (for example, when the phone receives a call or the user hits the Home button).
* **applicationDidEnterBackground:**is called when your app enters a background state after becoming inactive. You have approximately five seconds to run any tasks you need to back things up in case the app gets terminated later or right after that.
* **applicationWillTerminate:**is called when your app is about to be purged from memory. Call any final cleanups here.

Both **application: willFinishLaunchingWithOptions:**and **application: didFinishLaunchingWithOptions:**can potentially be launched with options identifying that the app was called to handle a push notification or url or something else. You need to return **true**if your app can handle the given activity or url.

### App Delegate life Cycle Methods

* [application:willFinishLaunchingWithOptions:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/application:willFinishLaunchingWithOptions:)—This method is your app’s first chance to execute code at launch time.
* [application:didFinishLaunchingWithOptions:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/application:didFinishLaunchingWithOptions:)—This method allows you to perform any final initialization before your app is displayed to the user.
* [applicationDidBecomeActive:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/applicationDidBecomeActive:)—Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.
* [applicationWillResignActive:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/applicationWillResignActive:)—Lets you know that your app is transitioning away from being the foreground app. Use this method to put your app into a quiescent state.
* [applicationDidEnterBackground:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/applicationDidEnterBackground:)—Lets you know that your app is now running in the background and may be suspended at any time.
* [applicationWillEnterForeground:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/applicationWillEnterForeground:)—Lets you know that your app is moving out of the background and back into the foreground, but that it is not yet active.
* [applicationWillTerminate:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/applicationWillTerminate:)—Lets you know that your app is being terminated. This method is not called if your app is suspended.



* **Not Running —**Either the application has not started yet or was running and has been terminated by the system.
* **Inactive —**An application is running in the Foreground but is not receiving any events. This could happen in case a Call or Message is received. An application could also stay in this state while in transition to a different state. In this State, we can not interact with app’s UI.
* **Active —**An application is running in the Foreground and receiving the events. This is the normal mode for the Foreground apps. The only way to go to or from the Active state is through the Inactive state. User normally interacts with UI, and can see the response/result for user actions.
* **Background —**An application is running in the background and executing the code. Freshly launching apps directly enter into In-Active state and then to Active state. Apps that are suspended, will come back to this background state, and then transition to In-Active → Active states. In addition, an application being launched directly into the background enters this state instead of the inactive state.
* **Suspended —**An application is in the background but is not executing the code. The system moves the application to this state automatically and does not notify. In case of low memory, the system may purge suspended application without notice to make free space for the foreground application. Usually after 5 secs spent in the background, apps will transition to Suspend state, but we can extend the time if app needs.
* **application:willFinishLaunchingWithOptions —**This method is called after your application has been launched successfully. It is the first method from our app delegate , which will be called. You can execute your code if the launch was successful.
* **application:didFinishLaunchingWithOptions —**This method is called before the app’s window is displayed. You can finalise your interface and can provide the root ViewController to the window.
* **applicationDidBecomeActive —**This method is either called to let your app know that it moved from the inactive to active state or your app was launched by the user or the system or in case user ignores an interruption (such as an incoming phone call or SMS message) that sent the application temporarily to the inactive state. You should use this method to restart any tasks that were paused (or not yet started) while the app was inactive.
* **applicationWillResignActive —**This method is called to let your app know that it is about to move from active to inactive state. This can happen in case of any interruptions (such as an incoming phone call or SMS message or Calendar alerts) or when the user quits the app. You should use this method to pause any ongoing tasks or disable timers etc.
* **applicationDidEnterBackground —**This method is called to let app know that it is not running in the foreground. You have approximately **five**seconds to perform any tasks and return back. In case you need additional time, you can request additional execution time from the system by calling [**beginBackgroundTask(expirationHandler:)**](https://developer.apple.com/documentation/uikit/uiapplication/1623031-beginbackgroundtask). If the method does not return before time runs out your app is terminated and purged from memory.
* **applicationWillEnterForeground —**This method is called as a part of the transition from the background to the active state. You should use this to undo any change you made to your app upon entering the background. **applicationDidBecomeActive** method is called soon after this method has finished its execution which then moves the app from the inactive to the active state.
* **applicationWillTerminate —**This method is called to let you know that your app is about to terminate. You should use this method to perform any final clean-up task. You have approximately **five** seconds to perform any tasks and return back. If the method does not return before time expires, the system may kill the process altogether. This method may be called in situations where the app is running in the background (not suspended) and the system needs to terminate it for some reason. You shouldn’t wait **applicationWillTerminate** to be called in order to save your data. There are some cases when **applicationWillTerminate** won’t be called before app termination. For example the system will not call **applicationWillTerminate** when the device reboots.

# [How to get Location (latitude & longitude value) in variable on iOS?](https://stackoverflow.com/questions/12736086/how-to-get-location-latitude-longitude-value-in-variable-on-ios)

You can use CoreLocation to get the longitude and latitude.

**Include framework:**

Click your project in navigator.

Click the plus button under "link binary with libraries"

Add Corelocation to your project.

**Import the header file:**

#import <CoreLocation/CoreLocation.h>

**Declare CLLocationManager:**

CLLocationManager \*locationManager;

**initialize locationManager:**

- (void)viewDidLoad { locationManager = [[CLLocationManager alloc] init]; locationManager.distanceFilter = kCLDistanceFilterNone; locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters; [locationManager startUpdatingLocation]; }

**Then, use**

float latitude = locationManager.location.coordinate.latitude; float longitude = locationManager.location.coordinate.longitude;

import UIKit import MapKit class ViewController: UIViewController { var locManager = CLLocationManager() var currentLocation: CLLocation! override func viewDidLoad() { super.viewDidLoad() locManager.requestWhenInUseAuthorization() if (CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedWhenInUse || CLLocationManager.authorizationStatus() == CLAuthorizationStatus.authorizedAlways){ guard let currentLocation = locManager.location else { return } print(currentLocation.coordinate.latitude) print(currentLocation.coordinate.longitude) } } }

iOS8+ requires one of these two strings to be set to use locations. Which one you use depends on how you intend ask for the location.

* Use NSLocationAlwaysUsageDescription for apps that want to use the device's location even when the app is not open and being used.
* Use NSLocationWhenInUseUsageDescription for apps that want to use the device's location only when the app is open and in use.

import CoreLocation

class ViewController: UIViewController,

CLLocationManagerDelegate

var seenError : Bool = false

var locationFixAchieved : Bool = false

var locationStatus : NSString = "Not Started"

var locationManager: CLLocationManager!

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

func initLocationManager() { seenError = false locationFixAchieved = false locationManager = CLLocationManager() locationManager.delegate = self locationManager.locationServicesEnabled locationManager.desiredAccuracy = kCLLocationAccuracyBest locationManager.requestAlwaysAuthorization() }

func locationManager(manager: CLLocationManager!, didFailWithError error: NSError!) { locationManager.stopUpdatingLocation() if (error) { if (seenError == false) { seenError = true print(error) } } }

func locationManager(manager: CLLocationManager!, didUpdateLocations locations: AnyObject[]!) { if (locationFixAchieved == false) { locationFixAchieved = true var locationArray = locations as NSArray var locationObj = locationArray.lastObject as CLLocation var coord = locationObj.coordinate println(coord.latitude) println(coord.longitude) } }

func locationManager(manager: CLLocationManager!, didChangeAuthorizationStatus status: CLAuthorizationStatus) { var shouldIAllow = false switch status { case CLAuthorizationStatus.Restricted: locationStatus = "Restricted Access to location" case CLAuthorizationStatus.Denied: locationStatus = "User denied access to location" case CLAuthorizationStatus.NotDetermined: locationStatus = "Status not determined" default: locationStatus = "Allowed to location Access" shouldIAllow = true } NSNotificationCenter.defaultCenter().postNotificationName("LabelHasbeenUpdated", object: nil) if (shouldIAllow == true) { NSLog("Location to Allowed") // Start location services locationManager.startUpdatingLocation() } else { NSLog("Denied access: \(locationStatus)") } }

# CLLocationManagerDelegate

The methods that you use to receive events from an associated location manager object.

### Responding to Location Events

[- locationManager:didUpdateLocations:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423615-locationmanager?language=objc)

Tells the delegate that new location data is available.

[- locationManager:didFailWithError:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423786-locationmanager?language=objc)

Tells the delegate that the location manager was unable to retrieve a location value.

[- locationManager:didFinishDeferredUpdatesWithError:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423537-locationmanager?language=objc)

Tells the delegate that updates will no longer be deferred.

[- locationManager:didUpdateToLocation:fromLocation:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423716-locationmanager?language=objc)

Tells the delegate that a new location value is available.

### Pausing Location Updates

[- locationManagerDidPauseLocationUpdates:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621553-locationmanagerdidpauselocationu?language=objc)

Tells the delegate that location updates were paused.

[- locationManagerDidResumeLocationUpdates:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621512-locationmanagerdidresumelocation?language=objc)

Tells the delegate that the delivery of location updates has resumed.

### Responding to Heading Events

[- locationManager:didUpdateHeading:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621555-locationmanager?language=objc)

Tells the delegate that the location manager received updated heading information.

[- locationManagerShouldDisplayHeadingCalibration:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621457-locationmanagershoulddisplayhead?language=objc)

Asks the delegate whether the heading calibration alert should be displayed.

### Responding to Region Events

[- locationManager:didEnterRegion:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423560-locationmanager?language=objc)

Tells the delegate that the user entered the specified region.

[- locationManager:didExitRegion:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423630-locationmanager?language=objc)

Tells the delegate that the user left the specified region.

[- locationManager:didDetermineState:forRegion:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423570-locationmanager?language=objc)

Tells the delegate about the state of the specified region.

[- locationManager:monitoringDidFailForRegion:withError:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423720-locationmanager?language=objc)

Tells the delegate that a region monitoring error occurred.

[- locationManager:didStartMonitoringForRegion:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423842-locationmanager?language=objc)

Tells the delegate that a new region is being monitored.

### Responding to Ranging Events

[- locationManager:didRangeBeacons:satisfyingConstraint:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/3240610-locationmanager?language=objc)

Tells the delegate that a beacon satisfying the constraint was detected.

Beta

[- locationManager:didFailRangingBeaconsForConstraint:error:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/3240609-locationmanager?language=objc)

Tells the delegate that no beacons were detected that satisfy the constraint.

Beta

[~~- locationManager:didRangeBeacons:inRegion:~~](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621501-locationmanager?language=objc)

Tells the delegate that one or more beacons are in range.

Deprecated

[~~- locationManager:rangingBeaconsDidFailForRegion:withError:~~](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621483-locationmanager?language=objc)

Tells the delegate that an error occurred while gathering ranging information for a set of beacons.

Deprecated

### Responding to Visit Events

[- locationManager:didVisit:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1621529-locationmanager?language=objc)

Tells the delegate that a new visit-related event was received.

### Responding to Authorization Changes

[- locationManager:didChangeAuthorizationStatus:](https://developer.apple.com/documentation/corelocation/cllocationmanagerdelegate/1423701-locationmanager?language=objc)

Tells the delegate that the authorization status for the application changed.

I will simply break down to two parts, ios pre 7 and ios 7+:

# iOS version pre 7

The answer is simply 600 seconds (10 minutes), reason is provided by the article above.

# iOS version 7+

The answer is that the time system allocates you is opportunistic. You will have to use @Gary Riches's suggestion

NSLog(@"Time Remaining: %f", [[UIApplication sharedApplication] backgroundTimeRemaining]);

to find out. The reason for it being opportunistic is the way iOS 7+ handles background tasks is completely different, certainly optimised. To be exact, It has an intermittent behaviour, and therefore, if you need background tasks such as downloading a big chuck of data, it will be much more effective if you use NSURLSession instead.

If you want to upload your files when app is in background, you should use Apple's background service. iOS will give your app time of approx. 3 minutes (based on some experience) for completing your task and then it will kill your app.

Apple allows longer run of the app in special cases. For that you will need to use UIBackgroundModes in your info.plist file.

# iOS Backgrounding Techniques

In the following sections, we will explore the following iOS features alongside the existing backgrounding options:

* [Opportunistic Background Tasks](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/ios-backgrounding-with-tasks#background_tasks_in_iOS_7) - Preserve battery life by running background tasks in opportunistic chunks when the device is awake for other processing.
* [Background Transfer Service](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/ios-backgrounding-with-tasks#background-transfers) - Reliably upload and download files regardless of network status or file size.
* [Background Fetch](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/updating-an-application-in-the-background#background_fetch) - Refresh an application from the background at system-determined intervals.
* [Remote Notifications](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/updating-an-application-in-the-background#remote_notifications) - Use push notifications to trigger content updates in the background before the user opens the application, with an option to notify the user or update silently.
* **Background UI Updates** - Prepare the application UI for the user, and update the application's snapshot, all from the background.

# iOS Backgrounding with Tasks

* 03/18/2017
* 7 minutes to read

* + [+1](https://github.com/MicrosoftDocs/xamarin-docs/blob/live/docs/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/ios-backgrounding-with-tasks.md" \o "6 Contributors)

The simplest way to perform backgrounding on iOS is to break your backgrounding requirements into tasks, and run the tasks in the background. Tasks are under a strict time limit, and typically get about 600 seconds (10 minutes) of processing time after an application has moved to the background on iOS 6, and less than 10 minutes on iOS 7+.

Background tasks can be broken up into three categories:

1. **Background-Safe Tasks** - Called anywhere in the application where you have a task you don't want interrupted should the application enter the background.
2. **DidEnterBackground Tasks** - Called during the DidEnterBackground application lifecycle method to assist in cleanup and state saving.
3. **Background Transfers (iOS 7+)** - A special type of background task used to perform network transfers on iOS 7. Unlike regular tasks, background transfers do not have a pre-determined time limit.

Background-safe and DidEnterBackground tasks are safe to use on both iOS 6 and iOS 7, with some minor differences. Let's investigate these two types of tasks in greater detail.

## Creating Background-Safe Tasks

Some applications contain tasks that shouldn't be interrupted by iOS should the application change state. One way to protect these tasks from being interrupted is to register them with iOS as long-running tasks. You can use this pattern anywhere in your application where you don't want a task being interrupted should the user put the app into the background. A great candidate for this pattern would be tasks such as sending a new user's registration information to your server, or verifying login information.

The following code snippet demonstrates registering a task to run in the background:

C#Copy

nint taskID = UIApplication.SharedApplication.BeginBackgroundTask( () => {}); //runs on main or background thread FinishLongRunningTask(taskID); UIApplication.SharedApplication.EndBackgroundTask(taskID);

The registration process pairs a task with a unique identifier, taskID, and then wraps it in matching BeginBackgroundTask and EndBackgroundTask calls. To generate the identifier, we make a call to the BeginBackgroundTask method on the UIApplication object, and then start the long-running task, usually on a new thread. When the task is complete, we call EndBackgroundTaskand pass in the same identifier. This is important because iOS will terminate the application if a BeginBackgroundTask call does not have a matching EndBackgroundTask.

**Important**

Background-safe tasks can run on either the main thread or a background thread, depending on the application's needs.

## Performing Tasks During DidEnterBackground

In addition to making a long-running task background-safe, registration can be used to kick off tasks as an application is being put in the background. iOS provides an event method in the AppDelegate class called DidEnterBackground that can be used to save application state, save user data, and encrypt sensitive content before an application enters the background. An application has approximately five seconds to return from this method or it will get terminated. Therefore, cleanup tasks that might take more than five seconds to complete can be called from inside the DidEnterBackground method. These tasks must be invoked on a separate thread.

The process is nearly identical to that of registering a long-running task. The following code snippet illustrates this in action:

C#Copy

public override void DidEnterBackground (UIApplication application) { nint taskID = UIApplication.SharedApplication.BeginBackgroundTask( () => {}); new Task ( () => { DoWork(); UIApplication.SharedApplication.EndBackgroundTask(taskID); }).Start(); }

We begin by overriding the DidEnterBackground method in the AppDelegate, where we register our task via BeginBackgroundTask as we did in the previous example. Next, we spawn a new thread and perform our long-running task. Note that the EndBackgroundTask call is now made from inside the long-running task, since the DidEnterBackground method will have already returned.

**Important**

iOS uses a [**watchdog mechanism**](https://developer.apple.com/library/ios/qa/qa1693/_index.html) to ensure that an application's UI remains responsive. An application that spends too much time in DidEnterBackground will become unresponsive in the UI. Kicking off tasks to run in the background allows DidEnterBackground to return in a timely manner, keeping the UI responsive and preventing the watchdog from killing the application.

## Handling Background Task Time Limits

iOS places strict limits on how long a background task can run, and if the EndBackgroundTask call is not made within the allotted time, the application will be terminated. By keeping track of the remaining backgrounding time, and using expiration handlers when necessary, we can avoid iOS terminating the application.

### Accessing Background Time Remaining

If an application with registered tasks gets moved to the background, the registered tasks will get about 600 seconds to run. We can check how much time the task has to complete using the static BackgroundTimeRemaining property of the UIApplication class. The following code will give us the time, in seconds, that our background task has left:

C#Copy

double timeRemaining = UIApplication.SharedApplication.BackgroundTimeRemaining;

### Avoiding App Termination With Expiration Handlers

In addition to giving access to the BackgroundTimeRemaining property, iOS provides a graceful way to handle background time expiration through an **Expiration Handler**. This is an optional block of code that will get executed when the time allotted for a task is about to expire. Code in the Expiration Handler calls EndBackgroundTask and passes in the task ID, which indicates that the app is behaving well and prevents iOS from terminating the app even if the task runs out of time. EndBackgroundTask must be called within the expiration handler, as well as in the normal course of execution.

The expiration handler is expressed as an anonymous function using a lambda expression, as illustrated below:

C#Copy

Task.Factory.StartNew( () => { //expirationHandler only called if background time allowed exceeded var taskId = UIApplication.SharedApplication.BeginBackgroundTask(() => { Console.WriteLine("Exhausted time"); UIApplication.SharedApplication.EndBackgroundTask(taskId); }); while(myFlag == true) { Console.WriteLine(UIApplication.SharedApplication.BackgroundTimeRemaining); myFlag = SomeCalculationNeedsMoreTime(); } //Only called if loop terminated due to myFlag and not expiration of time UIApplication.SharedApplication.EndBackgroundTask(taskId); });

While expiration handlers are not required for the code to run, you should always use an expiration handler with a background task.

## Background Tasks in iOS 7+

The biggest change in iOS 7 with regard to background tasks is not how the tasks are implemented, but when they run.

Recall that pre-iOS 7, a task running in the background had 600 seconds to complete. One reason for this limit is that a task running in the background would keep the device awake for the duration of the task:

iOS 7 background processing is optimized for longer battery life. In iOS 7, backgrounding becomes opportunistic: instead of keeping the device awake, tasks respect when the device goes to sleep, and instead do their processing in chunks when the device wakes up to handle phone calls, notifications, incoming emails, and other common interruptions. The following diagram provides insight into how a task might be broken up:

Because the task run time is not longer continuous, tasks that perform network transfers must be handled differently in iOS 7. Developers are encouraged to use the NSURlSession API to handle network transfers. The next section is an overview of background transfers.

## Background Transfers

The backbone of background transfers in iOS 7 is the new NSURLSession API. NSURLSessionallows us to create tasks to:

1. Transfer content through network and device interruptions.
2. Upload and download large files ( Background Transfer Service ).

Let's take a closer look at how this works.

### NSURLSession API

NSURLSession is a powerful API for transferring content over the network. It provides a set of tools to handle transfer of data through network interruptions and changes in application states.

The NSURLSession API creates one or several sessions, which in turn spawn tasks to shuttle blocks of related data across the network. Tasks run asynchronously to transfer data quickly and reliably. Because NSURLSession is asynchronous, every session requires a completion handler block to let the system and application know when a transfer is complete.

To perform a network transfer that is valid on both pre-iOS 7 and post-iOS 7, check if an NSURLSession is available to enqueue transfers, and use a regular background task to perform the transfer if it is not:

C#Copy

if ([NSURLSession class]) { // Create a background session and enqueue transfers } else { // Start a background task and transfer directly // Do NOT make calls to update the UI here! }

**Important**

Avoid making calls to update the UI from the background in iOS 6-compliant code, as iOS 6 does not support background UI updates, and will terminate the application.

The NSURLSession API includes a rich set of features to handle authentication, manage failed transfers, and report client-side - but not server-side - errors. It helps bridge the interruptions in task run time introduced in iOS 7, and also provides support for transferring large files quickly and reliably. The next section explores this second feature.

### Background Transfer Service

Prior to iOS 7, uploading or downloading files in the background was unreliable. Background tasks get a limited time to run, but the time it takes to transfer a file varies with the network and the size of the file. In iOS 7, we can use an NSURLSession to successfully upload and download large files. The particular NSURLSession session type that handles network transfers of large files in the background is known as the Background Transfer Service.

Transfers initiated using the Background Transfer Service are managed by the operating system and provide APIs to handle authentication and errors. Because transfers are not bound by an arbitrary time limit, they can be used to upload or download large files, auto-update content in the background, and more. Refer to the [Background Transfer Walkthrough](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-walkthroughs/background-transfer-walkthrough) for details on how to implement the Service.

The Background Transfer Service is often paired with Background Fetch or Remote Notifications to help applications refresh content in the background. In the next two sections, we introduce the concept of registering entire applications to run in the background on both iOS 6 and iOS 7

# Updating a Xamarin.iOS App in the Background

* 03/18/2017
* 8 minutes to read

Background refresh is the process of waking an application that is suspended or not running, and updating it with new content. iOS provides three options for refreshing content in the background:

1. Region Monitoring and Significant Location Changes Service - Location-aware APIs trigger background updates based on changes in the user's location. These APIs can be used with discretion to refresh content in non-location-based iOS 6 applications, where other options are not available.
2. Background Fetch (iOS 7+) - A temporal approach to refreshing non-critical content that updates frequently .
3. Remote Notifications (iOS 7+) - Applications that receive push notifications can use the notifications to trigger background content refreshes. This method can be used to update with important, time-sensitive content that updates sporadically .

The following sections cover the basics of these options.

## Region Monitoring and Significant Location Changes

iOS provides two location-aware APIs with backgrounding capabilities:

1. Region Monitoring is the process of setting up regions with boundaries, and waking the device when the user enters or exits a region. Regions are circular and can be of varying size. When the user crosses a region boundary, the device will wake up to handle the event, usually by firing a notification or kicking off a task. Region Monitoring requires GPS, and increases battery and data usage.
2. The Significant Location Changes Service is a simpler, power-conserving option available for devices with cellular radios. An application listening for significant location changes will be notified when the device switches cell towers. This service can be used to wake a suspended or terminated application, and provides an opportunity to check for new content in the background. Background activity is limited to about 10 seconds, unless paired with a [Background Task](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/ios-backgrounding-with-tasks) .

An application does not need the location UIBackgroundMode to use these location-aware APIs. Because iOS doesn't track the types of tasks that can run when the device is woken by changes in the user's location, these APIs provide a work-around for updating content in the background on iOS 6. Keep in mind that triggering background updates with location-based APIs will draw on device resources, and may confuse users who don't understand why an application requires access to their location. Use discretion when implementing Region Monitoring or Significant Location Changes for background processing in applications that aren't already using the location APIs.

Apps using location monitoring for background processing expose a flaw in iOS 6: if an application's needs don't fit into a background-necessary category, it has limited backgrounding options. With the introduction of two new APIs, Background Fetch and Remote Notifications, iOS 7 (and greater) provides backgrounding opportunities to more applications. The next two sections introduce these new APIs.

## Background Fetch (iOS 7 and Greater)

In iOS 6, an application entering the foreground needed time to load new content, briefly presenting users with content they've already seen. Background fetch allows applications to load new data before a user launches the application, and provide the user with the most up-to-date content.

To implement background fetch, edit Info.plist and check the **Enable Background Modes** and **Background Fetch** check boxes:

Next, in the AppDelegate, override the FinishedLaunching method to set the minimum fetch interval. In this example, we let the OS decide how often to fetch new content:

C#Copy

public override bool FinishedLaunching (UIApplication application, NSDictionary launchOptions) { UIApplication.SharedApplication.SetMinimumBackgroundFetchInterval (UIApplication.BackgroundFetchIntervalMinimum); return true; }

Finally, perform the fetch by overriding the PerformFetch method in the AppDelegate, and passing in a completion handler. The completion handler is a delegate that takes a UIBackgroundFetchResult:

C#Copy

public override void PerformFetch (UIApplication application, Action<UIBackgroundFetchResult> completionHandler) { // Check for new data, and display it ... // Inform system of fetch results completionHandler (UIBackgroundFetchResult.NewData); }

When we're done updating content, we let the OS know by calling the completion handler with the appropriate status. iOS offers three options for completion handler status:

1. UIBackgroundFetchResult.NewData - Called when new content has been fetched, and the application has been updated.
2. UIBackgroundFetchResult.NoData - Called when the fetch for new content went through, but no content is available.
3. UIBackgroundFetchResult.Failed - Useful for error handling, this is called when the fetch was unable to go through.

Applications using Background Fetch can make calls to update the UI from the background. When the user opens the app, the UI will be up to date and displaying new content. This will also update the application's App Switcher snapshot, so the user can see when the application has new content.

**Important**

Once PerformFetch is called, the application has approximately 30 seconds to kick off download of new content, and call the completion handler block. If this takes too long, the app will be terminated. Consider using Background Fetch with the Background Transfer Service when downloading media or other large files.

### BackgroundFetchInterval

In the sample code above, we let the OS decide how often to fetch new content by setting the minimum fetch interval to BackgroundFetchIntervalMinimum. iOS offers three options for the fetch interval:

1. BackgroundFetchIntervalNever - Tell the system to never fetch new content. Use this to turn off fetching in certain situations, such as when the user is not signed in. This is the default value for the fetch interval.
2. BackgroundFetchIntervalMinimum - Let the system decide how often to fetch based on user patterns, battery life, data usage, and the needs of other applications.
3. BackgroundFetchIntervalCustom - If you know how often an application's content gets updated, you can specify a "sleep" interval after every fetch, during which the application will be prevented from fetching new content. Once that interval is up, the system will determine when to fetch content.

Both BackgroundFetchIntervalMinimum and BackgroundFetchIntervalCustom rely on the system to schedule fetches. This interval is dynamic, adapting to the device's needs as well as the individual user's habits. For example, if one user checks an application every morning, and another checks every hour, iOS will ensure the content is up to date for both users every time they open the application.

Background Fetch should be used for applications that update frequently with non-critical content. For applications with critical updates, Remote Notifications should be used. Remote Notifications are based on Background Fetch, and share the same completion handler. We'll dive into Remote Notifications next.

## Remote Notifications (iOS 7 and Greater)

Push notifications are JSON messages sent from a provider to a device by way of the Apple Push Notification service (APNs).

In iOS 6, an incoming push notifications tells the system to alert the user that something interesting has happened in an application. Clicking on the notification pulls the application out of the suspended or terminated state, and the app would begin updating content. iOS 7 (and greater) extends ordinary push notifications by giving applications a chance to update content in the background before notifying the user, so that the user can open the application and be presented with new content immediately.

To implement remote notifications, edit Info.plist and check the **Enable Background Modes**and **Remote notifications** check boxes:

Next, set the content-available flag on the push notification itself to 1. This lets the application know to fetch new content before displaying the alert:

C#Copy

'aps' { 'content-available': 1, 'alert': 'Something new has happened in your app!'' }

In the AppDelegate, override the DidReceiveRemoteNotification method to check the notification payload for available content, and call the appropriate completion handler block:

C#Copy

public override void DidReceiveRemoteNotification (UIApplication application, NSDictionary userInfo, Action<UIBackgroundFetchResult> completionHandler) { if([content-available]) { // fetch content completionHandler (UIBackgroundFetchResult.NewData); } }

Remote notifications should be used for infrequent updates with content that is crucial to the application's functionality. For more information on remote notifications, see the Xamarin [Push Notifications in iOS](https://docs.microsoft.com/en-us/xamarin/ios/platform/user-notifications/deprecated/remote-notifications-in-ios) guide.

**Important**

Because the update mechanism in Remote Notifications is based on Background Fetch, the application must kick off download of new content and call the completion handler block within 30 seconds of receiving the notification, or iOS will terminate the application. Consider pairing Remote Notifications with Background Transfer Service when downloading media or other large files in the background.

### Silent Remote Notifications

Remote Notifications are a simple way to notify applications of updates and kick off fetching new content, but there are cases where we don't need to notify the user that something has changed. For example, if a user flags a file for synching, we don't need to notify them every time the file updates. File synching is not a surprising event, nor does it require the user's immediate attention. Users just expect the file to be up-to-date when they open it.

For cases like the one above, iOS allows push notifications to be sent silently - that is, without an alert. To turn a regular notification into a silent one, simply remove the alert from the notification payload:

C#Copy

'aps' { 'content-available': 1 }

#### Rate Limits

The biggest difference between normal and silent notifications from a developer perspective is that silent pushes are rate limited. APNs will delay the delivery of silent pushes to the device if the push rate gets too high. This is to ensure that applications don't drain device resources with too many silent notifications.

However, APNs will let silent notifications "piggyback" alongside a normal Remote Notification or keep-alive response. Because regular notifications are not rate limited, they can be used to push stored up silent notifications from the APNs to the device, as illustrated by the following diagram:

**Important**

Apple encourages developers to send silent push notifications whenever the application requires, and let the APNs schedule their delivery.

In this section, we've covered the various options for refreshing content in the background to run tasks that don't fit into a background-necessary category. Now, let's see some of these APIs in action.

# Introduction to Backgrounding in iOS

* 07/24/2018
* 4 minutes to read

iOS regulates background processing very tightly, and offers three approaches to implement it:

* **Register a Background Task** - If an application needs to complete an important task, it can ask iOS not to interrupt the task when the application moves into the background. For example, an application might need to finish logging in a user, or finish downloading a large file.
* **Register as a Background-Necessary Application** - An app can register as a specific type of application that has known, specific backgrounding requirements, such as Audio , VoIP ,External Accessory , Newsstand , and Location . These applications are allowed continuous background processing privileges as long as they are performing tasks that are within the parameters of the registered application type.
* **Enable Background Updates** - Applications can trigger background updates with Region Monitoring or by listening for Significant Location Changes . As of iOS 7, applications can also register to update content in the background using Background Fetch or Remote Notifications .

## Application States and Application Delegate Methods

Before we dive into the code for background processing in iOS, we need to understand how backgrounding affects the lifecycle of an iOS application.

The iOS application lifecycle is a collection of application states and methods for moving between them. An application transitions between states based on the behavior of the user and the backgrounding requirements of the application. The movement is illustrated by the following diagram:

* **Not Running** - The application has not yet been launched on the device.
* **Running/Active** - The application is on the screen, and is executing code in the foreground.
* **Inactive** - The application is interrupted by an incoming phone call, text, or other interruption.
* **Backgrounded** - The application moves into the background and continues executing background code.
* **Suspended** - If the application does not have any code to run in the background, or if all code has completed, the app will be Suspended by the OS. A suspended application's process is kept alive, but the application is unable to execute any code in this state.
* **Return to Not Running/Termination (Rare)** - Occasionally, the application's process is destroyed, and the application returns to the Not Running state. This happens in low-memory situations, or if the user manually terminates the application.

Since the introduction of multitasking support, iOS rarely terminates idle applications, and instead keeps their processes Suspended in memory. Keeping an application's process alive ensures that the application launches quickly the next time the user opens it. It also means applications can move freely from the Suspended state back into the Backgrounded state without drawing on system resources. iOS 7 exploits this feature with new APIs that enable applications to pause background tasks when the device goes to sleep, update content directly from the background without user interaction, and more. We will cover the new APIs in [iOS Backgrounding Techniques](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/index).

## Application Lifecycle Methods

When an app changes state, iOS notifies the application through event methods in the AppDelegate class:

* OnActivated - This is called the first time the application is launched, and every time the app comes back into the foreground. This is the place to put code that needs to run every time the app is opened.
* OnResignActivation - If the user receives an interruption such as a text or phone call, this method gets called and the app is temporarily inactivated. Should the user accept the phone call, the app will be sent to the background.
* DidEnterBackground - Called when the app enters the backgrounded state, this method gives an application about five seconds to prepare for possible termination. Use this time to save user data and tasks, and remove sensitive information from the screen.
* WillEnterForeground - When a user returns to a backgrounded or suspended application, and launches it into the foreground, WillEnterForeground gets called. This is the time to prepare the app to take the foreground by rehydrating any state saved duringDidEnterBackground . OnActivated will be called immediately after this method completes.
* WillTerminate - The application is shut down, and its process is destroyed. This method only gets called if multitasking is not available on the device or the OS version, if memory is low, or if the user manually terminates a backgrounded application. Note that suspended applications that get terminated will not call WillTerminate .

The following diagram illustrates how the application states and lifecycle methods fit together:

## User Controls for Backgrounding in iOS

iOS 7 introduced several features to give users more control over an application's backgrounded state. Both the App Switcher and the Background App Refresh setting affect the Application Lifecycle.

### App Switcher

The App Switcher is an important control feature introduced in iOS 7. It is launched by double-tapping the **Home** button, and shows the applications whose processes are alive:

Using the App Switcher, users can scroll through snapshots of all backgrounded and suspended applications. Tapping an application launches it into the foreground. Swiping up removes the application from the background, terminating its process. We will take a closer look at the App Switcher in the [iOS Application Lifecycle Demo](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/application-lifecycle-demo) in the next section.

**Important**

The App Switcher does not show a difference between backgrounded and suspended applications.

### Background App Refresh Settings

iOS 7 increases user control over the Application Lifecycle by allowing users to opt out of backgrounding for applications [registered for background processing](https://docs.microsoft.com/en-us/xamarin/ios/app-fundamentals/backgrounding/ios-backgrounding-techniques/registering-applications-to-run-in-background). This does not prevent applications from running background tasks.

Users can change this setting by navigating to **Settings > General > Background App Refresh**and editing the backgrounding privileges for a selected application. If Background App Refresh is set to off, the application will be suspended immediately upon entering the background, and prevented from doing any background processing:

Developers can check the Background Refresh Application status with the BackgroundRefreshStatus API. For an example, refer to the [Check Background Refresh Setting recipe](https://github.com/xamarin/recipes/tree/master/Recipes/ios/multitasking/check_background_refresh_setting).

We've covered the basics of the iOS Application Lifecycle, and features for controlling the Application Lifecycle. Next, let's see the iOS Application Lifecycle in action.

See "Background Execution" section of the [iPhoneAppProgrammingGuide](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html). In short, your app must be one of these types:

* Apps that play audible content to the user while in the background, such as a music player app
* Apps that keep users informed of their location at all times, such as a navigation app
* Apps that support Voice over Internet Protocol (VoIP)
* Newsstand apps that need to download and process new content
* Apps that receive regular updates from external accessories

And you must add to the Info.plist as follows: Add the UIBackgroundModes key to your Info.plist file and set its value to an array containing one or more of the following strings:

* audio—The app plays audible content to the user while in the background. (This content includes streaming audio or video content using AirPlay.)
* location—The app keeps users informed of their location, even while it is running in the background.
* voip—The app provides the ability for the user to make phone calls using an Internet connection.
* newsstand-content—The app is aNewsstand app that downloads and processesmagazine or newspaper content in the background.
* external-accessory—The app works with a hardware accessory that needs to deliver updates on a regular schedule through the External Accessory framework.
* bluetooth-central—The app works with a Bluetooth accessory that needs to deliver updates on a regular schedule through the CoreBluetooth framework

Note that part of the review process will be checking to make sure that your app does what it says it's doing with regard to background processing.

you’ll create an app that uses the most common background modes: audio playback, location updates, general tasks, and background fetch.

our app is only allowed to keep running in the background in very specific cases. For example, these include playing audio, getting location updates or fetching the latest content from a server.

If your task does not fall into these categories, backgrounding may not be for you. You may even find yourself with an App Store rejection if you try to cheat the system by using background modes outside the realm of their purposes, so consider yourself warned!

To get to the background modes capability list you:

1. Select the project from the **Project navigator**.
2. Click the app target.
3. Select the **Capabilities** tab.
4. Turn the **Background Modes** switch on.

In this background modes tutorial, you’ll investigate four ways of doing background processing:

* **Play audio**: The app can continue playing and/or recording audio in the background.
* **Receive location updates**: The app can continue to get callbacks as the device’s location changes.
* **Perform finite-length tasks**: The generic “whatever” case, where the app can run arbitrary code for a limited amount of time.
* **Background Fetch**: Get updates to the latest content scheduled by iOS.

iOS determines how much time you get after your app moves to the background. There are no guarantees on the time you’re granted, but you can always check backgroundTimeRemaining on UIApplication.shared. This will tell you how much time you have left.

. In your AppDelegate class please update below function and then run your app in background.

func applicationDidEnterBackground(\_ application: UIApplication) { application.beginBackgroundTask(withName: "") {} }

And below is my ViewController class

import UIKit

import CoreLocation

class ViewController: UIViewController, CLLocationManagerDelegate {

let locationManager = CLLocationManager()

override func viewDidLoad() {

super.viewDidLoad()

locationManager.delegate = self;

locationManager.requestAlwaysAuthorization()

locationManager.desiredAccuracy = kCLLocationAccuracyBest;

locationManager.allowsBackgroundLocationUpdates = true

locationManager.distanceFilter = kCLDistanceFilterNone

Timer.scheduledTimer(withTimeInterval: 1.0, repeats: true) { (timer) in self.locationManager.startUpdatingLocation()

}

}

func locationManager(\_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {

print("didUpdateLocations \(Date())")

self.locationManager.stopUpdatingLocation()

} }

locationManager.distanceFilter = kCLDistanceFilterNone; // whenever we move, location is updated

locationManager.desiredAccuracy = kCLLocationAccuracyBest; // get best current locaton coords

# NSLocationUsageDescription

A message that tells the user why the app is requesting access to the user’s location information.

With Swift 3, UITextViewDelegate provides a [textView(\_:shouldInteractWith:in:interaction:)](https://developer.apple.com/reference/uikit/uitextviewdelegate/1649337-textview)method. textView(\_:shouldInteractWith:in:interaction:) has the following declaration:

Asks the delegate if the specified text view should allow the specified type of user interaction with the given URL in the given range of text.

optional func textView(\_ textView: UITextView, shouldInteractWith URL: URL, in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool

The following code shows how to open UITextView web links in a SFSafariViewController instead of opening them in Safari app:

import UIKit import SafariServices class ViewController: UIViewController, UITextViewDelegate { override func viewDidLoad() { super.viewDidLoad() // Set textView let textView = UITextView() textView.text = "http://www.yahoo.fr http://www.google.fr" textView.isUserInteractionEnabled = true textView.isEditable = false textView.isSelectable = true textView.dataDetectorTypes = UIDataDetectorTypes.link // Add view controller as the textView's delegate textView.delegate = self // auto layout view.addSubview(textView) textView.translatesAutoresizingMaskIntoConstraints = false textView.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true textView.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true textView.heightAnchor.constraint(equalToConstant: 300).isActive = true textView.widthAnchor.constraint(equalToConstant: 300).isActive = true } func textView(\_ textView: UITextView, shouldInteractWith URL: URL, in characterRange: NSRange, interaction: UITextItemInteraction) -> Bool { // Open links with a SFSafariViewController instance and return false to prevent the system to open Safari app let safariViewController = SFSafariViewController(url: URL) present(safariViewController, animated: true, completion: nil) return false } }

The shouldInteractWithURL function should return true only if the link has to be opened up in Safari. If you are handling the link yourself you should return false. Check the changed line in the code below.

You can also use optional chaining instead of the if condition to call pushViewController

func textView(textView: UITextView, shouldInteractWithURL URL: NSURL, inRange characterRange: NSRange) -> Bool { let webViewController = WebViewController() webViewController.urlToLoad = URL navigationController?.pushViewController(webViewController, animated: true) return false // Changed line. }

UIWebView \*webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)]; [webView setDelegate:self]; NSString \*urlAddress = @"http://www.google.com/"; NSURL \*url = [NSURL URLWithString:urlAddress]; NSURLRequest \*requestObj = [NSURLRequest requestWithURL:url]; [webView loadRequest:requestObj]; [self.view addSubview:webView];

# UIWebView

A view that embeds web content in your app.

In apps that run in iOS 8 and later, use the [WKWebView](https://developer.apple.com/documentation/webkit/wkwebview?language=objc) class instead of using UIWebView. Additionally, consider setting the [WKPreferences](https://developer.apple.com/documentation/webkit/wkpreferences?language=objc) property [javaScriptEnabled](https://developer.apple.com/documentation/webkit/wkpreferences/1536203-javascriptenabled?language=objc) to NO if you render files that are not supposed to run JavaScript.

Use the [loadHTMLString:baseURL:](https://developer.apple.com/documentation/uikit/uiwebview/1617979-loadhtmlstring?language=objc) method to begin loading local HTML files or the [loadRequest:](https://developer.apple.com/documentation/uikit/uiwebview/1617957-loadrequest?language=objc) method to begin loading web content. Use the [stopLoading](https://developer.apple.com/documentation/uikit/uiwebview/1617974-stoploading?language=objc) method to stop loading, and the [loading](https://developer.apple.com/documentation/uikit/uiwebview/1617978-loading?language=objc) property to find out if a web view is in the process of loading.

If you allow the user to move back and forward through the webpage history, then you can use the [goBack](https://developer.apple.com/documentation/uikit/uiwebview/1617975-goback?language=objc) and [goForward](https://developer.apple.com/documentation/uikit/uiwebview/1617940-goforward?language=objc) methods as actions for buttons. Use the [canGoBack](https://developer.apple.com/documentation/uikit/uiwebview/1617931-cangoback?language=objc) and [canGoForward](https://developer.apple.com/documentation/uikit/uiwebview/1617951-cangoforward?language=objc) properties to disable the buttons when the user can’t move in a direction.

By default, a web view automatically converts telephone numbers that appear in web content to Phone links. When a Phone link is tapped, the Phone app launches and dials the number. To turn off this default behavior, set the [dataDetectorTypes](https://developer.apple.com/documentation/uikit/uiwebview/1617965-datadetectortypes?language=objc) property with a [UIDataDetectorTypes](https://developer.apple.com/documentation/uikit/uidatadetectortypes?language=objc) bitfield that does not contain the [UIDataDetectorTypePhoneNumber](https://developer.apple.com/documentation/uikit/uidatadetectortypes/uidatadetectortypephonenumber?language=objc) flag.

You can also use the [scalesPageToFit](https://developer.apple.com/documentation/uikit/uiwebview/1617950-scalespagetofit?language=objc) property to programmatically set the scale of web content the first time it is displayed in a web view. Thereafter, the user can change the scale using gestures.

Set the [delegate](https://developer.apple.com/documentation/uikit/uiwebview/1617937-delegate?language=objc) property to an object conforming to the [UIWebViewDelegate](https://developer.apple.com/documentation/uikit/uiwebviewdelegate?language=objc) protocol if you want to track the loading of web content.

After creating a new WKWebView object using the [initWithFrame:configuration:](https://developer.apple.com/documentation/webkit/wkwebview/1414998-initwithframe?language=objc)method, you need to load the web content. Use the [loadHTMLString:baseURL:](https://developer.apple.com/documentation/webkit/wkwebview/1415004-loadhtmlstring?language=objc) method to begin loading local HTML files or the [loadRequest:](https://developer.apple.com/documentation/webkit/wkwebview/1414954-loadrequest?language=objc) method to begin loading web content. Use the [stopLoading](https://developer.apple.com/documentation/webkit/wkwebview/1414981-stoploading?language=objc) method to stop loading, and the [loading](https://developer.apple.com/documentation/webkit/wkwebview/1414964-loading?language=objc) property to find out if a web view is in the process of loading. Set the delegate property to an object conforming to the [WKUIDelegate](https://developer.apple.com/documentation/webkit/wkuidelegate?language=objc) protocol to track the loading of web content.

import UIKit import WebKit class ViewController: UIViewController, WKUIDelegate { var webView: WKWebView! override func loadView() { let webConfiguration = WKWebViewConfiguration() webView = WKWebView(frame: .zero, configuration: webConfiguration) webView.uiDelegate = self view = webView } override func viewDidLoad() { super.viewDidLoad() let myURL = URL(string:"https://www.apple.com") let myRequest = URLRequest(url: myURL!) webView.load(myRequest) }}

# Using Javascript with WKWebView

UIWebView \*webView = [[UIWebView alloc]initWithFrame:CGRectMake(5,bluebaseView.frame.size.height + bluebaseView.frame.origin.y + 10,baseView.frame.size.width - 10 , 230)];

webView.scrollView.scrollEnabled = NO;

webView.userInteractionEnabled = NO;

NSString \*str = @"icanCash is our rewards programme where you can earn cash in your icanstay account simply by referring us to your friends. Every unit of icanCash you earn is equal to one rupee, which you can use to buy a luxury stay voucher or make a last-minute booking on our site or mobile app.";

// [webView loadHTMLString:[NSString stringWithFormat:@"<div align='justify'>%@<div>", str ]baseURL:nil];

[webView loadHTMLString:[NSString stringWithFormat:@"<html><body style=\"text-align:justify\"><font color='Gray'><font size='4'> %@ </font></font></body></Html>", str] baseURL:nil];

[baseView addSubview:webView];

# OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

<https://github.com/rafameyer/my-movies-swift3.0-ios-omdb-api>

# stringWithContentsOfURL:

Returns a string created by reading data from the file named by a given URL.

# [NSString stringWithContentsOfURL is deprecated. What should I do?](https://stackoverflow.com/questions/5638909/nsstring-stringwithcontentsofurl-is-deprecated-what-should-i-do)

Use stringWithContentsOfURL:encoding:error:

EDIT: Here's your code using the above method:

NSError \*error = nil; NSString \*locationString = [NSString stringWithContentsOfURL:[NSURL URLWithString:urlString] encoding:NSUTF8StringEncoding error:&error]; NSArray \*listItems = [locationString componentsSeparatedByString:@","];

The words **synchronization** and **concurrency** are overlapping and sometimes synonymous terms. The word *synchronization* generally means sharing data between multiple processors or threads, while *concurrency* refers to a measure of– or the art of improving– how effectively an application allows multiple jobs required by that application (e.g. serving web page requests from a web server) to run simultaneously.

According to Apple, [“Concurrency is the notion of multiple things happening at the same time.”](https://developer.apple.com/library/content/documentation/General/Conceptual/ConcurrencyProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008091-CH1-SW1)

* *The term*thread*is used to refer to a separate path of execution for code. The underlying implementation for threads in OS X is based on the POSIX threads API.*
* *The term*process*is used to refer to a running executable, which can encompass multiple threads.*
* *The term*task*is used to refer to the abstract concept of work that needs to be performed.*

*With threads, the burden of creating a scalable solution rests squarely on the shoulders of you, the developer. You have to decide how many threads to create and adjust that number dynamically as system conditions change.*

*GCD takes care of creating the needed threads and of scheduling your tasks to run on those threads. Because the thread management is now part of the system, GCD provides a holistic approach to task management and execution, providing better efficiency than traditional threads. …*

*A parallel program is one that uses a multiplicity of computational hardware (e.g., several processor cores) to perform a computation more quickly. The aim is to arrive at the answer earlier, by delegating different parts of the computation to different processors that execute at the same time.*

*By contrast, concurrency is a program-structuring technique in which there are multiple threads of control. Conceptually, the threads of control execute “at the same time”; that is, the user sees their effects interleaved. Whether they actually execute at the same time or not is an implementation detail; a concurrent program can execute on a single processor through interleaved execution or on multiple physical processors. …*

*Dispatch queues are an easy way to perform tasks asynchronously and concurrently in your application. A task is simply some work that your application needs to perform. … A dispatch queue is an object-like structure that manages the tasks you submit to it. All dispatch queues are first-in, first-out data structures. Thus, the tasks you add to a queue are always started in the same order that they were added. GCD provides some dispatch queues for you automatically, but others you can create for specific purposes. …*

*Serial queues (also known as private dispatch queues) execute one task at a time in the order in which they are added to the queue. The currently executing task runs on a distinct thread (which can vary from task to task) that is managed by the dispatch queue. Serial queues are often used to synchronize access to a specific resource. …*

*Concurrent queues (also known as a type of global dispatch queue) execute one or more tasks concurrently, but tasks are still started in the order in which they were added to the queue. The currently executing tasks run on distinct threads that are managed by the dispatch queue. The exact number of tasks executing at any given point is variable and depends on system conditions. …*

*The main dispatch queue is a globally available serial queue that executes tasks on the application’s main thread. This queue works with the application’s run loop (if one is present) to interleave the execution of queued tasks with the execution of other event sources attached to the run loop. Because it runs on your application’s main thread, the main queue is often used as a key synchronization point for an application. …*

**Asynchronous versus synchronous**  
When you submit a task — block of code — to a GCD queue, you need to be aware of what happens after submission. You can write code so that when a task is submitted to a queue, your code must wait until the task completes (this is *synchronous*). On the other hand, you can write your code so that when you submit a task to a queue, your code keeps on executing regardless of the submitted task’s execution (this is *asynchronous*).

* async - concurrent: the code runs on a background thread. Control returns immediately to the main thread (and UI). The block can't assume that it's the only block running on that queue
* async - serial: the code runs on a background thread. Control returns immediately to the main thread. The block *can* assume that it's the only block running on that queue
* sync - concurrent: the code runs on a background thread but the main thread waits for it to finish, blocking any updates to the UI. The block can't assume that it's the only block running on that queue (I could have added another block using async a few seconds previously)
* sync - serial: the code runs on a background thread but the main thread waits for it to finish, blocking any updates to the UI. The block *can* assume that it's the only block running on that queue
* Here are a couple of experiments that i have done to make me understand about these serial, concurrent queues with Grand Central Dispatch.
* func doLongAsyncTaskInSerialQueue() { let serialQueue = DispatchQueue(label: "com.queue.Serial") for i in 1...5 { serialQueue.async { if Thread.isMainThread{ print("task running in main thread") }else{ print("task running in background thread") } let imgURL = URL(string: "https://upload.wikimedia.org/wikipedia/commons/0/07/Huge\_ball\_at\_Vilnius\_center.jpg")! let \_ = try! Data(contentsOf: imgURL) print("\(i) completed downloading") } } }
* Task will run in different thread(other than main thread) when you use async in GCD. Async means execute next line do not wait until the block executes which results non blocking main thread & main queue. Since its serial queue, all are executed in the order they are added to serial queue.Tasks executed serially are always executed one at a time by the single thread associated with the Queue.
* func doLongSyncTaskInSerialQueue() { let serialQueue = DispatchQueue(label: "com.queue.Serial") for i in 1...5 { serialQueue.sync { if Thread.isMainThread{ print("task running in main thread") }else{ print("task running in background thread") } let imgURL = URL(string: "https://upload.wikimedia.org/wikipedia/commons/0/07/Huge\_ball\_at\_Vilnius\_center.jpg")! let \_ = try! Data(contentsOf: imgURL) print("\(i) completed downloading") } } }
* Task may run in main thread when you use sync in GCD. Sync runs a block on a given queue and waits for it to complete which results in blocking main thread or main queue.Since the main queue needs to wait until the dispatched block completes, main thread will be available to process blocks from queues other than the main queue.Therefore there is a chance of the code executing on the background queue may actually be executing on the main thread Since its serial queue, all are executed in the order they are added(FIFO).
* func doLongASyncTaskInConcurrentQueue() { let concurrentQueue = DispatchQueue(label: "com.queue.Concurrent", attributes: .concurrent) for i in 1...5 { concurrentQueue.async { if Thread.isMainThread{ print("task running in main thread") }else{ print("task running in background thread") } let imgURL = URL(string: "https://upload.wikimedia.org/wikipedia/commons/0/07/Huge\_ball\_at\_Vilnius\_center.jpg")! let \_ = try! Data(contentsOf: imgURL) print("\(i) completed downloading") } print("\(i) executing") } }
* Task will run in background thread when you use async in GCD. Async means execute next line do not wait until the block executes which results non blocking main thread. Remember in concurrent queue, task are processed in the order they are added to queue but with different threads attached to the queue. Remember they are not supposed to finish the task as the order they are added to the queue.Order of task differs each time threads are created as necessarily automatically.Task are executed in parallel. With more than that(maxConcurrentOperationCount) is reached, some tasks will behave as a serial until a thread is free.
* func doLongSyncTaskInConcurrentQueue() { let concurrentQueue = DispatchQueue(label: "com.queue.Concurrent", attributes: .concurrent) for i in 1...5 { concurrentQueue.sync { if Thread.isMainThread{ print("task running in main thread") }else{ print("task running in background thread") } let imgURL = URL(string: "https://upload.wikimedia.org/wikipedia/commons/0/07/Huge\_ball\_at\_Vilnius\_center.jpg")! let \_ = try! Data(contentsOf: imgURL) print("\(i) completed downloading") } print("\(i) executed") } }
* Task may run in main thread when you use sync in GCD. Sync runs a block on a given queue and waits for it to complete which results in blocking main thread or main queue.Since the main queue needs to wait until the dispatched block completes, main thread will be available to process blocks from queues other than the main queue.Therefore there is a chance of the code executing on the background queue may actually be executing on the main thread. Since its concurrent queue, tasks may not finish in the order they are added to queue. But with synchronous operation it does although they may be processed by different threads. So, it behaves as this is the serial queue.
* **Here is a summary of this experiments**
* Remember using GCD you are only adding task to the Queue and performing task from that queue. Queue dispatches your task either in main or background thread depending on whether operation is synchronous or asynchronous. Types of queues are Serial,Concurrent,Main dispatch queue.All the task you perform is done by default from Main dispatch queue.There are already four predefined global concurrent queues for your application to use and one main queue(DispatchQueue.main).You can also manually create your own queue and perform task from that queue.
* UI Related task should always be performed from main thread by dispatching the task to Main queue.Short hand utility is *DispatchQueue.main.sync/async* whereas network related/heavy operations should always be done asynchronously no matters which ever thread you are using either main or background
* **EDIT: However, There are cases you need to perform network calls operations synchronously in a background thread without freezing UI(e.g.refreshing OAuth Token and wait if it succeed or not).You need to wrap that method inside a asynchronous operation.This way your heavy operations are executed in the order and without Blocking main thread.**
* func doMultipleSyncTaskWithinAsynchronousOperation() { let concurrentQueue = DispatchQueue(label: "com.queue.Concurrent", attributes: .concurrent) concurrentQueue.async { let concurrentQueue = DispatchQueue.global(qos: DispatchQoS.QoSClass.default) for i in 1...5 { concurrentQueue.sync { let imgURL = URL(string: "https://upload.wikimedia.org/wikipedia/commons/0/07/Huge\_ball\_at\_Vilnius\_center.jpg")! let \_ = try! Data(contentsOf: imgURL) print("\(i) completed downloading") } print("\(i) executed") } } }

<https://stackoverflow.com/questions/19179358/concurrent-vs-serial-queues-in-gcd>

**Option 1:**

NSString \*haystack = @"value:hello World:value"; NSString \*haystackPrefix = @"value:"; NSString \*haystackSuffix = @":value"; NSRange needleRange = NSMakeRange(haystackPrefix.length, haystack.length - haystackPrefix.length - haystackSuffix.length); NSString \*needle = [haystack substringWithRange:needleRange]; NSLog(@"needle: %@", needle); // -> "hello World"

**Option 2:**

NSRegularExpression \*regex = [NSRegularExpression regularExpressionWithPattern:@"^value:(.+?):value$" options:0 error:nil]; NSTextCheckingResult \*match = [regex firstMatchInString:haystack options:NSAnchoredSearch range:NSMakeRange(0, haystack.length)]; NSRange needleRange = [match rangeAtIndex: 1]; NSString \*needle = [haystack substringWithRange:needleRange];

This one might be a bit over the top for your rather trivial case though.

**Option 3:**

NSString \*needle = [haystack componentsSeparatedByString:@":"][1];

<https://stackoverflow.com/questions/5676106/how-to-get-substring-of-nsstring>

[@"abc xyz http://www.abc.com aaa bbb ccc" substringWithRange:NSMakeRange(8, 18)]

p = print

po = print object

p prints value of primitive variable or value of a reference

po try to call -description for that object and print returned string

Asynchronous image downloader with cache support as a UIImageView category [https://sdwebimage.github.io](https://sdwebimage.github.io/)

[IQKeyboardManager](https://github.com/hackiftekhar/IQKeyboardManager)/**IQKeyboardManagerSwift**/

FMDBMigrationManager provides a simple, elegant interface for managing SQLite database schemas with FMDB.

The Core Spotlight (CS) framework is part of a greater collection of APIs, known as *Search APIs*, which give the opportunity to programmers to increase the discoverability, visibility and ease of access of their apps significantly, and in a fashion that wasn’t possible to be used in earlier versions of iOS.

Let’s write now a small custom function, where the data loading will take place. As you’ll see next, we just ensure that the property list file actually exists, and if so, we initialize the array with the contents of that file

|  |  |
| --- | --- |
| 1  2  3  4  5 | func loadMoviesInfo() {      if let path = NSBundle.mainBundle().pathForResource("MoviesData", ofType: "plist") {          moviesInfo = NSMutableArray(contentsOfFile: path)      }  } |

<https://www.appcoda.com/core-spotlight-framework/>

The new search APIs in iOS 9 look quite promising for developers, as they allow apps to become a lot more discoverable and easily accessible by users. In this tutorial we went through all those actions that index an app’s data so they can be found on the Spotlight searches, and how a selected result can be handled by the app so specific data to be displayed to the user. Implementing such features in your apps definitely increase the user experience, so it’s something that you should seriously consider to include in your current and future projects. And as once again we’ve come to the end, I leave you hoping that you found this post useful! Have fun!

## Indexing Data For The Spotlight

By using the Core Spotlight framework in iOS 9 it’s possible to make the data of any application searchable through the Spotlight. The key for doing that is to ask specifically from the Core Spotlight API to index our data so it can be found on searches performed by the user. But neither our app, nor the CS API decides what kind of data this is going to be. It’s our responsibility to prepare that data and provide it to the API in a specific form.

To make that more clear, all the data that we want to make searchable through the Spotlight must be represented as CSSearchableItem objects, and then to be grouped together as an array and to be given to the CS API for indexing. A single CSSearchableItem object contains a set of attributes that make crystal clear to iOS the details of the searchable item, like what pieces of data should be displayed upon searching (for example, the name of the movie, its image and the description), and which the keywords that make data from our app appear on the Spotlight are. All the attributes for a single searchable item are represented by a CSSearchableItemAttributeSet object that provides many properties for assigning the values we need.

For each movie we’ll create a *CSSearchableItemAttributeSet* object, and then we’ll set the attributes for the data we want to be shown in the search results when searching on the Spotlight. In our demo, we’ll specify the movie title, description and image as the pieces of data that will be displayed to the user.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | func setupSearchableContent() {      var searchableItems = [CSSearchableItem]()        for i in 0...(moviesInfo.count - 1) {          let movie = moviesInfo[i] as! [String: String]            let searchableItemAttributeSet = CSSearchableItemAttributeSet(itemContentType: kUTTypeText as String)            // Set the title.          searchableItemAttributeSet.title = movie["Title"]!            // Set the movie image.          let imagePathParts = movie["Image"]!.componentsSeparatedByString(".")          searchableItemAttributeSet.thumbnailURL = NSBundle.mainBundle().URLForResource(imagePathParts[0], withExtension: imagePathParts[1])            // Set the description.          searchableItemAttributeSet.contentDescription = movie["Description"]!      }  } |

## What is iBeacon? What are iBeacons?

The term iBeacon and Beacon are often used interchangeably. iBeacon is the name for Apple’s technology standard, which allows Mobile Apps (running on both iOS and Android devices) to listen for signals from beacons in the physical world and react accordingly. In essence, iBeacon technology allows Mobile Apps [to understand their position](http://www.ibeacon.com/apples-ibeacon-future-mobile-shopping/)on a micro-local scale, and deliver hyper-contextual content to users based on location. The underlying communication technology is Bluetooth Low Energy.

**iBeacon** is a protocol developed by [Apple](https://en.wikipedia.org/wiki/Apple_Inc.) and introduced at the [Apple Worldwide Developers Conference](https://en.wikipedia.org/wiki/Apple_Worldwide_Developers_Conference) in 2013.[[1]](https://en.wikipedia.org/wiki/IBeacon#cite_note-1) Various vendors have since made iBeacon-compatible hardware transmitters – typically called [beacons](https://en.wikipedia.org/wiki/Bluetooth_low_energy_beacon) – a class of [Bluetooth Low Energy](https://en.wikipedia.org/wiki/Bluetooth_Low_Energy)(BLE) devices that broadcast their identifier to nearby [portable electronic](https://en.wikipedia.org/wiki/Mobile_computing) devices. The technology enables [smartphones](https://en.wikipedia.org/wiki/Smartphone), [tablets](https://en.wikipedia.org/wiki/Computer_tablet) and other devices to perform actions when in close proximity to an iBeacon.[[2]](https://en.wikipedia.org/wiki/IBeacon#cite_note-2)[[3]](https://en.wikipedia.org/wiki/IBeacon#cite_note-3)

iBeacon is based on [Bluetooth low energy proximity sensing](https://en.wikipedia.org/wiki/Bluetooth_Low_Energy#Proximity_sensing) by transmitting a [universally unique identifier](https://en.wikipedia.org/wiki/Universally_unique_identifier)[[4]](https://en.wikipedia.org/wiki/IBeacon#cite_note-4) picked up by a compatible app or operating system. The identifier and several bytes sent with it can be used to determine the device's physical location,[[5]](https://en.wikipedia.org/wiki/IBeacon#cite_note-5) track customers, or trigger a [location-based](https://en.wikipedia.org/wiki/Location-based_service) action on the device such as a [check-in on social media](https://en.wikipedia.org/wiki/Check-in#Social_network) or a [push notification](https://en.wikipedia.org/wiki/Mobile_marketing).

iBeacon can also be used with an application as an [indoor positioning system](https://en.wikipedia.org/wiki/Indoor_positioning_system),[[6]](https://en.wikipedia.org/wiki/IBeacon#cite_note-6)[[7]](https://en.wikipedia.org/wiki/IBeacon#cite_note-7)[[8]](https://en.wikipedia.org/wiki/IBeacon#cite_note-8) which helps smartphones determine their approximate location or context. With the help of an iBeacon, a smartphone's software can approximately find its relative location to an iBeacon in a store. [Brick and mortar](https://en.wikipedia.org/wiki/Brick_and_mortar) retail stores use the beacons for [mobile commerce](https://en.wikipedia.org/wiki/Mobile_commerce), offering customers special deals through [mobile marketing](https://en.wikipedia.org/wiki/Mobile_marketing),[[9]](https://en.wikipedia.org/wiki/IBeacon#cite_note-mcforbes-9) and can enable [mobile payments](https://en.wikipedia.org/wiki/Mobile_payment) through [point of sale](https://en.wikipedia.org/wiki/Point_of_sale) systems.

Another application is distributing messages at a specific [Point of Interest](https://en.wikipedia.org/wiki/Point_of_Interest), for example a store, a bus stop, a room or a more specific location like a piece of furniture or a vending machine. This is similar to previously used geopush technology based on [GPS](https://en.wikipedia.org/wiki/GPS), but with a much reduced impact on battery life and better precision.

iBeacon differs from some other location-based technologies as the broadcasting device (beacon) is only a 1-way transmitter to the receiving smartphone or receiving device, and necessitates a specific app installed on the device to interact with the beacons. This ensures that only the installed app (not the iBeacon transmitter) can track users as they walk around the transmitters.

# isEmpty

A Boolean value indicating whether a string has no characters.

if emptyString.isEmpty { print("Nothing to see here") }

**Instamojo** provides free payment gateway in India. Trusted by 800000+ Indian Businesses, 100% Secure, No setup cost, No maintenance cost.

[Skip to end of metadata](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#page-metadata-end)

[Go to start of metadata](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#page-metadata-start)

A *distribution provisioning profile* is used to deploy an app to users via the App Store or other app marketplace, such as the App Catalog. A distribution provisioning profile is associated with a distribution certificate that identifies a team or organization, not an individual team member. It authorizes an app to run on devices without the assistance of Xcode.

A distribution provisioning profile should use a unique app ID associated with one specific application.

Unless your organization is brand new to Apperian or app development in general, it's likely that you'll already have one or more distribution provisioning profiles. However, distribution provisioning profiles expire every year so you will periodically need to renew your distribution provisioning profile to prevent apps from expiring. For more information, see [Re-sign an App](https://help.apperian.com/display/pub/Re-sign+an+App).

When a provisioning profile has expired or is due to expire, you should edit it to generate an updated .mobileprovision file rather than create a new provisioning profile. If you create a new profile and use it to re-sign apps that were previously distributed to users, those users will not be able to install updates of the re-signed apps because iOS will not overwrite an app with an app signed with a different provisioning profile.

**Expired Apps**

Users cannot install expired apps from an App Catalog, and cannot run expired apps that are already installed. To prevent interruption to your users, you must renew credentials and re-sign apps **BEFORE**an app expires.

iOS apps with an expired distribution provisioning profile or distribution certificate are highlighted on the Applications page, and Apperian notifies administrators about apps due to expire. Notification emails are sent 60 days before an app expires, 45 days before, 30 days before, and then every day until the app either expires or is re-signed with new credentials.

The following procedures guide you through the process of creating, renewing, and downloading provisioning profiles. The .mobileprovision file that you'll download at the end is the final certificate used to sign an app.

# Ad Hoc or In House Distribution

During the following procedure you'll need to choose whether to use Ad Hoc or In House distribution. For typical implementations, you will need to use an In House distribution provisioning profile if you want to distribute apps to an an unlimited number of unspecified iOS devices. You can use an Ad Hoc profile when it is sufficient to distribute apps to a few specific devices (during testing and pilot phases, for example).

**Ad Hoc distribution**means that the app is bundled with a distribution provisioning profile that identifies specific iOS devices on which the app can be installed (up to 100 iOS devices). The UDIDs (Unique Device ID) for these devices must be registered through the [iOS Dev Center](https://developer.apple.com/devcenter/ios/index.action).

**In House distribution** means that the app is bundled with a distribution provisioning profile that does not limit distribution to specific iOS devices; the app can be installed on an infinite number of iOS devices.

 Learn how to find a device's UDID.

With a standard iOS Developer Program account, you can create an Ad Hoc distribution provisioning profile. With an iOS Developer Enterprise account, you can create Ad Hoc or In House profiles.

## Installing Apps Bundled with Ad Hoc Profiles

When an app is distributed Ad Hoc, the app is listed in the App Catalog for all users with group access to the app—regardless of whether the user's device is in the list of provisioned UDIDs. If the user's device is not provisioned, however, when the user clicks **Install**, the app downloads and starts to install, but the operating system will terminate the installation and display an “Unable to Download Application” message.

A progress indicator appears on the app icon until the message appears; this may take some time.

After the message appears, the icon is grayed-out. The user cannot complete the installation unless the app is updated to a version with either:

* An Ad Hoc profile that lists the UDID for the device
* An In House profile that does not limit distribution to specific devices.

**Manage Provisioning with Groups**

In the Admin Portal, you can use groups to organize apps so that users on non-provisioned devices do not have access to apps they cannot install.

For example, if you want to distribute the HR app with an Ad Hoc distribution provisioning profile that allows the app to be installed on five specific devices only, you can add a user group called “HR Ad Hoc” that includes the users of the five provisioned devices. You can then assign the HR app to the HR Ad Hoc group so that only those five users see the app in their App Catalog. For more information on using groups, see [Managing Groups](https://help.apperian.com/display/pub/Managing+Groups).

[Back to Top](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#ManageDistributionProvisioningProfiles-TOP)

# Before You Begin

Before you can create a distribution provisioning profile, you need the following:

* A distribution certificate. For more information, see [Manage Distribution Certificates](https://help.apperian.com/display/pub/Manage+Distribution+Certificates).
* A registered App ID. If you are signing the iOS App Catalog, you will need an explicit App ID registered for the App Catalog. If you are signing another app, you can use either an explicit App ID for that app or a wildcard App ID that can be used for any app. For instructions, see [Manage App Identifiers](https://help.apperian.com/display/pub/Manage+App+Identifiers).

[Back to Top](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#ManageDistributionProvisioningProfiles-TOP)

# Create a Distribution Provisioning Profile

The following procedure describes tasks that are performed outside of Apperian's systems. The UI or procedure may change without notice. For the official Apple documentation, see [Maintaining your Signing Credentials and Certificates](https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html#//apple_ref/doc/uid/TP40012582-CH31-SW1). In Apple's official documentation the terms distribution certificate and production certificate are synonymous.

###### To create a distribution provisioning profile

1. Log in to the [iOS Dev Center](http://developer.apple.com/devcenter/ios) with the Apple ID and password for your Apple Developer account.
2. In the left column, click **Certificates, Identifiers & Profiles**.
3. Under the iOS Apps section, click **Provisioning Profiles**.
4. Under Provisioning Profiles, click **Distribution** to display the iOS Provisioning Profiles (Distribution) page.
5. Click **Add (+)** to display the Add iOS Provisioning Profile page where you can select the type of profile you need.
6. If you are registered for the iOS Developer Enterprise Program, choose **In House** or **Ad Hoc** as the distribution method. If you are registered for the standard iOS Developer Program, choose **Ad Hoc** as the distribution method.
7. Click **Continue**.
8. On the Select App ID page, select the App ID and then click **Continue**. You can select an explicit App ID if you want to use the profile with a specific app, or you can select a wildcard App ID if you want to use the profile with any iOS app.
9. On the Select certifications page, select a certificate and then click **Continue**.
10. If you are doing Ad Hoc provisioning, the Select devices page displays. Select the check box next to each of the devices you want to authorize to use the app and then click **Continue**.
11. On the Name this profile and generate page, enter a name in the **Profile Name** field. This is the name that will be used to identify the profile in the portal.
12. Click **Generate** to create the profile.
13. When the page indicates that "Your provisioning profile is ready," click **Download** to download a profilename.mobileprovision file to your **Downloads** folder.
14. Click **Done**.

[Back to Top](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#ManageDistributionProvisioningProfiles-TOP)

# Renew a Distribution Provisioning Profile

###### To renew a distribution provisioning profile

1. Log in to the [iOS Dev Center](http://developer.apple.com/devcenter/ios) with the Apple ID and password for your Apple Developer account.
2. In the left column, click **Certificates, Identifiers & Profiles**.
3. Under the iOS Apps section, click **Provisioning Profiles**.
4. Under Provisioning Profiles, click **Distribution** to display the iOS Provisioning Profiles (Distribution) page.
5. Select the provisioning profile you want to edit. In this example, the profile is expired.
6. Click **Edit** to display the Edit iOS Provisioning Profile page.
7. Select the certificate you want to associate with the profile, if there is more than one.
8. Click **Generate** and then click **Download** to download a new profilename.mobileprovision file to your **Downloads** folder.

You are now ready to upload this updated distribution provisioning profile to Apperian so that you can re-sign apps that were signed with the previous version.

[Back to Top](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#ManageDistributionProvisioningProfiles-TOP)

# Download a Distribution Provisioning Profile

Follow this procedure to download a distribution provisioning profile as a .mobileprovision file.

If you followed the procedure above to [Create a Distribution Provisioning Profile](https://help.apperian.com/display/pub/Manage+Distribution+Provisioning+Profiles#ManageDistributionProvisioningProfiles-CreateaDistributionProvisioningProfile), you will have already done this.

###### To download a distribution provisioning profile

1. Log in to the [iOS Dev Center](http://developer.apple.com/devcenter/ios) with the Apple ID and password for your Apple Developer account.
2. In the left column, click **Certificates, Identifiers & Profiles**.
3. Under the iOS Apps section, click **Provisioning Profiles**.
4. Under Provisioning Profiles, click **Distribution** to display the iOS Provisioning Profiles (Distribution) page.
5. Select the provisioning profile you want to download and then click **Download** to download a profilename.mobileprovision file to your **Downloads** folder.

# NSObject

The group of methods that are fundamental to all Objective-C objects.

SSL stands for Secure Socket Layer, which is a protocol for creating an encrypted connection between client and server. It ensures that all data pass in network will be private and integral.

1. Client connects to server and requests server identify itself.
2. Server sends certificate to client (include public key)
3. Client checks if that certificate is valid. If it is, client creates a symmetric key (session key), encrypts with public key, then sends back to server
4. Server receives encrypted symmetric key, decrypts by its private key, then sends acknowledge packet to client
5. Client receives ACK and starts the session

Using SSL, the client will allow the connection only from trusted sources that have the valid certificate. And it looks good for most cases. But what if someone stands between client and server, and acts like they are the real server? Let's call client is C, server is S and the attacker is A.

In step 1, instead of sending packet to S, A can catch the packet and pretend it as S. What if instead of receiving certificate from S, client C will receive fake certificate from A and believe it's valid. A can make C think that it is communicating with S, but actually all connection flows will be directed to attacker A.

Hence, SSL pinning can be the solution to prevent Man-In-The-Middle (MITM) attack. SSL pinning will ensure that client connect with designated server. The main key of SSL pinning that server certificate will be saved in app bundle. Then, when client receives certificate from server, it then compares 2 certificates to make sure that they are the same before establishing the connection.

Now, I will show how to implement SSL pinning in iOS.

1. **NSURLSession**

For NSURLSession, the main method to handle SSL pinning is **URLSession:didReceiveChallenge:completionHandler:delegate**. Set your class to conform URLSessionDelegate and paste this function to your class:

|  |
| --- |
| func urlSession(\_ session: URLSession, didReceive challenge: URLAuthenticationChallenge, completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) { |
|  | if (challenge.protectionSpace.authenticationMethod == NSURLAuthenticationMethodServerTrust) { |
|  | if let serverTrust = challenge.protectionSpace.serverTrust { |
|  | var secresult = SecTrustResultType.invalid |
|  | let status = SecTrustEvaluate(serverTrust, &secresult) |
|  |  |
|  | if (errSecSuccess == status) { |
|  | if let serverCertificate = SecTrustGetCertificateAtIndex(serverTrust, 0) { |
|  | let serverCertificateData = SecCertificateCopyData(serverCertificate) |
|  | let data = CFDataGetBytePtr(serverCertificateData); |
|  | let size = CFDataGetLength(serverCertificateData); |
|  | let cert1 = NSData(bytes: data, length: size) |
|  | let file\_der = Bundle.main.path(forResource: "name-of-cert-file", ofType: "cer") |
|  |  |
|  | if let file = file\_der { |
|  | if let cert2 = NSData(contentsOfFile: file) { |
|  | if cert1.isEqual(to: cert2 as Data) { |
|  | completionHandler(URLSession.AuthChallengeDisposition.useCredential, URLCredential(trust:serverTrust)) |
|  | return |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Pinning failed |
|  | completionHandler(URLSession.AuthChallengeDisposition.cancelAuthenticationChallenge, nil) |
|  | } |

This function will “*requests credentials from the delegate in response to an authentication request from the remote server.*” We will compare the certificate from server with the one that saved in app bundle. If 2 certificates are identical, the authentication will let it pass and client can connect to server.

One disadvantage of SSL pinning is that you have to save the certificate in the app. Whenever the certificate is updated, we need to release new version of app. But this also leads to another problem: what we do with older version?

* Maintain the old certificate for a time, until we make sure all users have downloaded new version already.
* A special flow to download new certificate. But what if this flow is attacked?

While AFNetworking talks only to the servers whose certificates you have pinned, *Alamofire does it differently* - you pin a certificate per domain, so the certificate check will occur only if you talk to the list of predetermined domains. For all other domains, no check will be enforced. AFNetworking, on the other hand, will block all requests that don't pass checks for the certificates you have pinned in your app.

SSL pinning plays a major role in building highly secure mobile apps which users will be able to use even in countless insecure wireless networks that they encounter every day while using their mobile devices

SSL (Secure Socket Layer) ensures encrypted client-server communication over HTTP - specified by HTTPS (HTTP over SSL). The encryption is based on PKI (Public Key Infrastructure) and a session key. The session key was introduced because encrypting and decrypting a public/private key uses a lot of processing power and it would slow down the whole communication process.

## SSL pinning

We use SSL pinning to ensure that the app communicates only with the designated server itself. One of the prerequisites for SSL pinning is saving the target's server SSL certificate within the app bundle. The saved certificate is used when defining the pinned certificate(s) upon session configuration.

The main goal of TLS (Transport Security layer) is to add privacy and integrity to messages exchanged between two parties. In other words, TLS allows you to transmit data over a network without exposing that data to untrusted third parties.

A certificate is a file that encapsulates information about the server that owns the certificate. It’s similar to an identification card, such as a passport or a driver license.

A [**Certificate Authority (CA)**](https://en.wikipedia.org/wiki/Certificate_authority) can issue a certificate or it can be self-signed. In the first case, the CA must validate the identity of the certificate holder both before it issues the certificate and when your app uses the certificate. In the second case, the same entity whose identity it certifies signs the certificate.

### The Structure of a Digital Certificates

The structure of a certificate uses [X.509 standard](https://en.wikipedia.org/wiki/X.509). Here are the main fields:

* **Subject**: Provides the name of the entity (computer, user, network device, etc.) that the CA issued the certificate to.
* **Serial Number**: Provides a unique identifier for each certificate that a CA issues.
* **Issuer**: Provides a unique name for the CA that issued the certificate.
* **Valid From**: Provides the date and time when the certificate becomes valid.
* **Valid To**: Provides the date and time when the certificate is no longer considered valid.
* **Public Key**: Contains the public key of the key pair that goes with the certificate.
* **Algorithm Identifier**: Indicates the algorithm used to sign the certificate.
* [**Digital Signature**](https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq): A bit string used to verify the authenticity of the certificate.

X.509 certificates can be encoded differently, which will affect their appearance. The most common are:

* [**Privacy Enhanced Mail (PEM)**](https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail): A **Base-64** encoding, whose file extension is **.pem**.
* [**Distinguished Encoding Rules (DER)**](https://en.wikipedia.org/wiki/X.690#DER_encoding): A binary encoding, whose file extensions are **.cer**, **.der** and **.crt**.
* [**Public Key Cryptography Standards (PKCS)**](https://en.wikipedia.org/wiki/PKCS): Used to exchange public and private objects in a single file. Its extensions are **.p7b**, **.p7c**, **.p12**, **.pfx** etc.

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), a **certificate authority** or **certification authority** (**CA**) is an entity that issues [digital certificates](https://en.wikipedia.org/wiki/Public_key_certificate). A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon [signatures](https://en.wikipedia.org/wiki/Digital_signature) or on assertions made about the private key that corresponds to the certified public key. A CA acts as a [trusted third party](https://en.wikipedia.org/wiki/Trusted_third_party)—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by the [X.509](https://en.wikipedia.org/wiki/X.509) standard.

# App Store Review Guidelines

### 1. Safety

When people install an app from the App Store, they want to feel confident that it’s safe to do so—that the app doesn’t contain upsetting or offensive content, won’t damage their device, and isn’t likely to cause physical harm from its use. We’ve outlined the major pitfalls below, but if you’re looking to shock and offend people, the App Store isn’t the right place for your app.

* **1.1 Objectionable Content**

Apps should not include content that is offensive, insensitive, upsetting, intended to disgust, in exceptionally poor taste, or just plain creepy. Examples of such content include:

* + **1.1.1** Defamatory, discriminatory, or mean-spirited content, including references or commentary about religion, race, sexual orientation, gender, national/ethnic origin, or other targeted groups, particularly if the app is likely to humiliate, intimidate, or place a targeted individual or group in harm’s way. Professional political satirists and humorists are generally exempt from this requirement.
  + **1.1.2** Realistic portrayals of people or animals being killed, maimed, tortured, or abused, or content that encourages violence. “Enemies” within the context of a game cannot solely target a specific race, culture, real government, corporation, or any other real entity.
  + **1.1.3** Depictions that encourage illegal or reckless use of weapons and dangerous objects, or facilitate the purchase of firearms or ammunition.
  + **1.1.4** Overtly sexual or pornographic material, defined by Webster’s Dictionary as "explicit descriptions or displays of sexual organs or activities intended to stimulate erotic rather than aesthetic or emotional feelings."
  + **1.1.5** Inflammatory religious commentary or inaccurate or misleading quotations of religious texts.
  + **1.1.6** False information and features, including inaccurate device data or trick/joke functionality, such as fake location trackers. Stating that the app is “for entertainment purposes” won’t overcome this guideline. Apps that enable anonymous or prank phone calls or SMS/MMS messaging will be rejected.
  + **1.1.7** App Store Reviews:
    - App Store customer reviews can be an integral part of the app experience, so you should treat customers with respect when responding to their comments. Keep your responses targeted to the user’s comments and do not include personal information, spam, or marketing in your response.
    - Use the provided API to prompt users to review your app; this functionality allows customers to provide an App Store rating and review without the inconvenience of leaving your app, and we will disallow custom review prompts.
* **1.2 User Generated Content**

Apps with user-generated content present particular challenges, ranging from intellectual property infringement to anonymous bullying. To prevent abuse, apps with user-generated content or social networking services must include:

* + A method for filtering objectionable material from being posted to the app
  + A mechanism to report offensive content and timely responses to concerns
  + The ability to block abusive users from the service
  + Published contact information so users can easily reach you

Apps with user-generated content or services that end up being used primarily for pornographic content, objectification of real people (e.g. “hot-or-not” voting), making physical threats, or bullying do not belong on the App Store and may be removed without notice. If your app includes user-generated content from a web-based service, it may display incidental mature “NSFW” content, provided that the content is hidden by default and only displayed when the user turns it on via your website.

* **1.3 Kids Category**

The Kids Category is a great way for people to easily find apps that are designed for children. If you want to participate in the Kids Category, you should focus on creating a great experience specifically for younger users. These apps must not include links out of the app, purchasing opportunities, or other distractions to kids unless reserved for a designated area behind a parental gate. Keep in mind that once customers expect your app to follow the Kids Category requirements, it will need to continue to meet these guidelines in subsequent updates, even if you decide to deselect the category. Learn more about [parental gates](https://developer.apple.com/app-store/parental-gates/).

Apps in the Kids Category may not include third-party advertising or analytics. You should also pay particular attention to privacy laws around the world relating to the collection of data from children online. Be sure to review the [Privacy section](https://developer.apple.com/app-store/review/guidelines/#privacy) of these guidelines for more information.

* **1.4 Physical Harm**

If your app behaves in a way that risks physical harm, we may reject it. For example:

* + **1.4.1** Medical apps that could provide inaccurate data or information, or that could be used for diagnosing or treating patients may be reviewed with greater scrutiny.
    - Apps must clearly disclose data and methodology to support accuracy claims relating to health measurements, and if the level of accuracy or methodology cannot be validated, we will reject your app. For example, apps that claim to take x-rays, measure blood pressure, body temperature, blood glucose levels, or blood oxygen levels using only the sensors on the device are not permitted.
    - Apps should remind users to check with a doctor in addition to using the app and before making medical decisions.

If your medical app has received regulatory clearance, please submit a link to that documentation with your app.

* + **1.4.2** Drug dosage calculators must come from the drug manufacturer, a hospital, university, health insurance company, pharmacy or other approved entity, or receive approval by the FDA or one of its international counterparts. Given the potential harm to patients, we need to be sure that the app will be supported and updated over the long term.
  + **1.4.3** Apps that encourage consumption of tobacco and vape products, illegal drugs, or excessive amounts of alcohol are not permitted on the App Store. Apps that encourage minors to consume any of these substances will be rejected. Facilitating the sale of marijuana, tobacco, or controlled substances (except for licensed pharmacies) isn’t allowed.
  + **1.4.4** Apps may only display DUI checkpoints that are published by law enforcement agencies, and should never encourage drunk driving or other reckless behavior such as excessive speed.
  + **1.4.5** Apps should not urge customers to participate in activities (like bets, challenges, etc.) or use their devices in a way that risks physical harm to themselves or others.
* **1.5 Developer Information**

People need to know how to reach you with questions and support issues. Make sure your app and its Support URL include an easy way to contact you; this is particularly important for apps that may be used in the classroom. Failure to include accurate and up-to-date contact information not only frustrates customers, but may violate the law in some countries. Also ensure that Wallet passes include valid contact information from the issuer and are signed with a dedicated certificate assigned to the brand or trademark owner of the pass.

* **1.6 Data Security**

Apps should implement appropriate security measures to ensure proper handling of user information collected pursuant to the Apple Developer Program License Agreement and these Guidelines (see Guideline 5.1 for more information) and prevent its unauthorized use, disclosure, or access by third parties.

### 2. Performance

* **2.1 App Completeness**

Submissions to App Review, including apps you make available for pre-order, should be final versions with all necessary metadata and fully functional URLs included; placeholder text, empty websites, and other temporary content should be scrubbed before submission. Make sure your app has been tested on-device for bugs and stability before you submit it, and include demo account info (and turn on your back-end service!) if your app includes a login. If you offer in-app purchases in your app, make sure they are complete, up-to-date, and visible to the reviewer, or that you explain why not in your review notes. Please don’t treat App Review as a software testing service. We will reject incomplete app bundles and binaries that crash or exhibit obvious technical problems.

* **2.2 Beta Testing**

Demos, betas, and trial versions of your app don’t belong on the App Store – use TestFlight instead. Any app submitted for beta distribution via TestFlight should be intended for public distribution and should comply with the App Review Guidelines. Note, however, that apps using TestFlight cannot be distributed to testers in exchange for compensation of any kind, including as a reward for crowd-sourced funding. Significant updates to your beta build should be submitted to TestFlight App Review before being distributed to your testers. To learn more, visit the [TestFlight Beta Testing](https://developer.apple.com/testflight/).

* **2.3 Accurate Metadata**

Customers should know what they’re getting when they download or buy your app, so make sure your app description, screenshots, and previews accurately reflect the app’s core experience and remember to keep them up-to-date with new versions.

* + **2.3.1** Don’t include any hidden or undocumented features in your app; your app’s functionality should be clear to end-users and App Review. Similarly, you should not market your app on the App Store or offline as including content or services that it does not actually offer (e.g. iOS-based virus and malware scanners). Egregious or repeated behavior is grounds for removal from the Developer Program. We work hard to make the App Store a trustworthy ecosystem and expect our app developers to follow suit; if you’re dishonest, we don’t want to do business with you.
  + **2.3.2** If your app includes in-app purchases, make sure your app description, screenshots, and previews clearly indicate whether any featured items, levels, subscriptions, etc. require additional purchases. If you decide to promote in-app purchases on the App Store, ensure that the in-app purchase Display Name, Screenshot and Description are appropriate for a public audience, that you follow the guidance found in [Promoting Your In-App Purchases](https://developer.apple.com/app-store/promoting-in-app-purchases/), and that your app properly handles the [SKPaymentTransactionObserver method](https://developer.apple.com/documentation/storekit/skpaymenttransactionobserver/2877502-paymentqueue) so that customers can seamlessly complete the purchase when your app launches.
  + **2.3.3** Screenshots should show the app in use, and not merely the title art, log-in page, or splash screen. They may also include text and image overlays (e.g. to demonstrate input mechanisms, such as an animated touch point or Apple Pencil) and show extended functionality on device, such as Touch Bar.
  + **2.3.4** Previews are a great way for customers to see what your app looks like and what it does. To ensure people understand what they’ll be getting with your app, previews may only use video screen captures of the app itself. Stickers and iMessage extensions may show the user experience in the Messages app. You can add narration and video or textual overlays to help explain anything that isn’t clear from the video alone.
  + **2.3.5** Select the most appropriate category for your app, and check out the [App Store Category Definitions](https://developer.apple.com/app-store/categories/) if you need help. If you’re way off base, we may change the category for you.
  + **2.3.6** Answer the age rating questions in App Store Connect honestly so that your app aligns properly with parental controls. If your app is mis-rated, customers might be surprised by what they get, or it could trigger an inquiry from government regulators. If your app includes media that requires the display of content ratings or warnings (e.g. films, music, games, etc.), you are responsible for complying with local requirements in each territory where your app is available.
  + **2.3.7** Choose a unique app name, assign keywords that accurately describe your app, and don’t try to pack any of your metadata with trademarked terms, popular app names, or other irrelevant phrases just to game the system. App names must be limited to 30 characters and should not include prices, terms, or descriptions that are not the name of the app. App subtitles are a great way to provide additional context for your app; they must follow our standard metadata rules and should not include inappropriate content, reference other apps, or make unverifiable product claims. Apple may modify inappropriate keywords at any time or take other appropriate steps to prevent abuse.
  + **2.3.8** Metadata should be appropriate for all audiences, so make sure your app and in-app purchase icons, screenshots, and previews adhere to a 4+ age rating even if your app is rated higher. For example, if your app is a game that includes violence, select images that don’t depict a gruesome death or a gun pointed at a specific character. Use of terms like “For Kids” and “For Children” in app metadata is reserved for the Kids Category. Remember to ensure your metadata, including app name and icons (small, large, Apple Watch app, alternate icons, etc.), are similar to avoid creating confusion.
  + **2.3.9** You are responsible for securing the rights to use all materials in your app icons, screenshots, and previews, and you should display fictional account information instead of data from a real person.
  + **2.3.10** Make sure your app is focused on the iOS, Mac, Apple TV or Apple Watch experience, and don’t include names, icons, or imagery of other mobile platforms in your app or metadata, unless there is specific, approved interactive functionality. Make sure your app metadata is focused on the app itself and its experience. Don’t include irrelevant information, including but not limited to information about Apple or the development process.
  + **2.3.11** Apps you submit for pre-order on the App Store must be complete and deliverable as submitted. Ensure that the app you ultimately release is not materially different from what you advertise while the app is in a pre-order state. If you make material changes to the app (e.g. change business models), you should restart your pre-order sales.
  + **2.3.12** Apps must clearly describe new features and product changes in their “What’s New” text. Simple bug fixes, security updates, and performance improvements may rely on a generic description, but more significant changes must be listed in the notes.
* **2.4 Hardware Compatibility**
  + **2.4.1** To ensure people get the most out of your app, iPhone apps should run on iPad whenever possible. We encourage you to consider building universal apps so customers can use them on all of their devices. Learn more about [Universal apps](https://developer.apple.com/go/?id=xcode-universal).
  + **2.4.2** Design your app to use power efficiently and be used in a way that does not risk damage to the device. Apps should not rapidly drain battery, generate excessive heat, or put unnecessary strain on device resources. For example, apps should not encourage placing the device under a mattress or pillow while charging or perform excessive write cycles to the solid state drive. Apps, including any third-party advertisements displayed within them, may not run unrelated background processes, such as cryptocurrency mining.
  + **2.4.3** People should be able to use your Apple TV app without the need for hardware inputs beyond the Siri remote or third-party game controllers, but feel free to provide enhanced functionality when other peripherals are connected. If you require a game controller, make sure you clearly explain that in your metadata so customers know they need additional equipment to play.
  + **2.4.4** Apps should never suggest or require a restart of the device or modifications to system settings unrelated to the core functionality of the application. For example, don’t encourage users to turn off Wi-Fi, disable security features, etc.
  + **2.4.5** Apps distributed via the Mac App Store have some additional requirements to keep in mind:
    - **(i)** They must be appropriately sandboxed, and follow [macOS File System Documentation](https://developer.apple.com/library/archive/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010672). They should also only use the appropriate macOS APIs for modifying user data stored by other Apps (e.g. bookmarks, Address Book, or Calendar entries).
    - **(ii)** They must be packaged and submitted using technologies provided in Xcode; no third-party installers allowed. They must also be self-contained, single application installation bundles and cannot install code or resources in shared locations.
    - **(iii)** They may not auto-launch or have other code run automatically at startup or login without consent nor spawn processes that continue to run without consent after a user has quit the app. They should not automatically add their icons to the Dock or leave short cuts on the user desktop.
    - **(iv)** They may not download or install standalone apps, kexts, additional code, or resources to add functionality or significantly change the app from what we see during the review process.
    - **(v)** They may not request escalation to root privileges or use setuid attributes.
    - **(vi)** They may not present a license screen at launch, require license keys, or implement their own copy protection.
    - **(vii)** They must use the Mac App Store to distribute updates; other update mechanisms are not allowed.
    - **(viii)** Apps should run on the currently shipping OS and may not use deprecated or optionally installed technologies (e.g. Java, Rosetta)
    - **(ix)** Apps must contain all language and localization support in a single app bundle.
* **2.5 Software Requirements**
  + **2.5.1** Apps may only use public APIs and must run on the currently shipping OS. Learn more about [public APIs](https://developer.apple.com/documentation/). Keep your apps up-to-date and make sure you phase out any deprecated features, frameworks or technologies that will no longer be supported in future versions of an OS. Apps should use APIs and frameworks for their intended purposes and indicate that integration in their app description. For example, the HomeKit framework should provide home automation services; and HealthKit should be used for health and fitness purposes and integrate with the Health app.
  + **2.5.2** Apps should be self-contained in their bundles, and may not read or write data outside the designated container area, nor may they download, install, or execute code which introduces or changes features or functionality of the app, including other apps. Educational apps designed to teach, develop, or allow students to test executable code may, in limited circumstances, download code provided that such code is not used for other purposes. Such apps must make the source code provided by the Application completely viewable and editable by the user.
  + **2.5.3** Apps that transmit viruses, files, computer code, or programs that may harm or disrupt the normal operation of the operating system and/or hardware features, including Push Notifications and Game Center, will be rejected. Egregious violations and repeat behavior will result in removal from the Developer Program.
  + **2.5.4** Multitasking apps may only use background services for their intended purposes: VoIP, audio playback, location, task completion, local notifications, etc. If your app uses location background mode, include a reminder that doing so may dramatically decrease battery life.
  + **2.5.5** Apps must be fully functional on IPv6-only networks.
  + **2.5.6** Apps that browse the web must use the appropriate WebKit framework and WebKit Javascript.
  + **2.5.7** Video streaming content over a cellular network longer than 10 minutes must use HTTP Live Streaming and include a baseline 192 kbps HTTP Live stream.
  + **2.5.8** Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected.
  + **2.5.9** Apps that alter or disable the functions of standard switches, such as the Volume Up/Down and Ring/Silent switches, or other native user interface elements or behaviors will be rejected. For example, apps should not block links out to other apps or other features that users would expect to work a certain way. Learn more about proper handling of [links](https://developer.apple.com/design/).
  + **2.5.10** Apps should not be submitted with empty ad banners or test advertisements.
  + **2.5.11** SiriKit and Shortcuts
    - **(i)** Apps integrating SiriKit and Shortcuts should only sign up for intents they can handle without the support of an additional app and that users would expect from the stated functionality. For example, if your app is a meal planning app, you should not incorporate an intent to start a workout, even if the app shares integration with a fitness app.
    - **(ii)** Ensure that the vocabulary and phrases in your plist pertains to your app and the Siri functionality of the intents the app has registered for. Aliases must relate directly to your app or company name and should not be generic terms or include third-party app names or services.
    - **(iii)** Resolve the Siri request or Shortcut in the most direct way possible and do not insert ads or other marketing between the request and its fulfillment. Only request a disambiguation when required to complete the task (e.g. asking the user to specify a particular type of workout).
  + **2.5.12** Apps using CallKit or including an SMS Fraud Extension should only block phone numbers that are confirmed spam. Apps that include call-, SMS-, and MMS- blocking functionality or spam identification must clearly identify these features in their marketing text and explain the criteria for their blocked and spam lists. You may not use the data accessed via these tools for any purpose not directly related to operating or improving your app or extension (e.g. you may not use, share, or sell it for tracking purposes, creating user profiles, etc.).
  + **2.5.13** Apps using facial recognition for account authentication must use [LocalAuthentication](https://developer.apple.com/documentation/localauthentication/) (and not ARKit or other facial recognition technology) where possible, and must use an alternate authentication method for users under 13 years old.
  + **2.5.14** Apps must request explicit user consent and provide a clear visual and/or audible indication when recording, logging, or otherwise making a record of user activity. This includes any use of the device camera, microphone, screen recordings, or other user inputs.
  + **2.5.15** Apps that enable users to view and select files should include items from the Files app and the user’s iCloud documents.

### 3. Business

There are many ways to monetize your app on the App Store. If your business model isn’t obvious, make sure to explain in its metadata and App Review notes. If we can’t understand how your app works or your in-app purchases aren’t immediately obvious, it will delay your review and may trigger a rejection. And while pricing is up to you, we won’t distribute apps and in-app purchase items that are clear rip-offs. We’ll reject expensive apps that try to cheat users with irrationally high prices.

If we find that you have attempted to manipulate reviews, inflate your chart rankings with paid, incentivized, filtered, or fake feedback, or engage with third-party services to do so on your behalf, we will take steps to preserve the integrity of the App Store, which may include expelling you from the Developer Program.

* **3.1 Payments**
  + **3.1.1 In-App Purchase:**
    - If you want to unlock features or functionality within your app, (by way of example: subscriptions, in-game currencies, game levels, access to premium content, or unlocking a full version), you must use in-app purchase. Apps may not use their own mechanisms to unlock content or functionality, such as license keys, augmented reality markers, QR codes, etc. Apps and their metadata may not include buttons, external links, or other calls to action that direct customers to purchasing mechanisms other than in-app purchase.
    - Apps may use in-app purchase currencies to enable customers to “tip” digital content providers in the app.
    - Any credits or in-game currencies purchased via in-app purchase may not expire, and you should make sure you have a restore mechanism for any restorable in-app purchases.
    - Remember to assign the correct purchasability type or your app will be rejected.
    - Apps may enable gifting of items that are eligible for in-app purchase to others. Such gifts may only be refunded to the original purchaser and may not be exchanged.
    - Apps distributed via the Mac App Store may host plug-ins or extensions that are enabled with mechanisms other than the App Store.
    - Apps offering “loot boxes” or other mechanisms that provide randomized virtual items for purchase must disclose the odds of receiving each type of item to customers prior to purchase.
    - Non-subscription apps may offer a free time-based trial period before presenting a full unlock option by setting up a Non-Consumable IAP item at Price Tier 0 that follows the naming convention: “XX-day Trial.” Prior to the start of the trial, your app must clearly identify its duration, the content or services that will no longer be accessible when the trial ends, and any downstream charges the user would need to pay for full functionality. Learn more about managing content access and the duration of the trial period using [Receipts](https://developer.apple.com/library/archive/releasenotes/General/ValidateAppStoreReceipt/Introduction.html) and [Device Check](https://developer.apple.com/documentation/devicecheck).
  + **3.1.2 Subscriptions:** Apps may offer auto-renewing in-app purchase subscriptions, regardless of category on the App Store. When incorporating auto-renewable subscriptions into your app, be sure to follow the guidelines below.
  + **3.1.2(a) Permissible uses:** If you offer an auto-renewing subscription, you must provide ongoing value to the customer, and the subscription period must last at least seven days and be available across all of the user’s devices. While the following list is not exhaustive, examples of appropriate subscriptions include: new game levels; episodic content; multi-player support; apps that offer consistent, substantive updates; access to large collections of, or continually updated, media content; software as a service (“SAAS”); and cloud support. In addition:
    - Subscriptions may be offered alongside a la carte offerings (e.g. you may offer a subscription to an entire library of films as well the purchase or rental of a single movie).
    - You may offer a single subscription that is shared across your own apps and services, but these subscriptions may not extend to third-party apps or services. Games offered in a game subscription must be owned or exclusively licensed by the developer (e.g. not part of a game publishing platform). Each game must be downloaded directly from the App Store, must be designed to avoid duplicate payment by a subscriber, and should not disadvantage non-subscriber customers.
    - Subscriptions must work on all of the user’s devices where the app is available. Learn more about [sharing a subscription across your apps](https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/StoreKitGuide/Chapters/Subscriptions.html).
    - Apps must not force users to rate the app, review the app, download other apps, or other similar actions in order to access functionality, content, or use of the app.
    - As with all apps, those offering subscriptions should allow a user to get what they’ve paid for without performing additional tasks, such as posting on social media, uploading contacts, checking in to the app a certain number of times, etc.
    - Subscriptions may include consumable credits, gems, in-game currencies, etc., and you may offer subscriptions that include access to discounted consumable goods (e.g. a platinum membership that exposes gem-packs for a reduced price).
    - If you are changing your existing app to a subscription-based business model, you should not take away the primary functionality existing users have already paid for. For example, let customers who have already purchased a “full game unlock” continue to access the full game after you introduce a subscription model for new customers.
    - Auto-renewing subscription apps may offer a free trial period to customers by providing the relevant information set forth in App Store Connect.
    - Apps that attempt to scam users will be removed from the App Store. This includes apps that attempt to trick users into purchasing a subscription under false pretenses or engage in bait-and-switch and scam practices will be removed from the App Store and you may be removed from the Apple Developer Program. Learn more about [Subscription Free Trials](https://developer.apple.com/app-store/subscriptions/).
  + **3.1.2(b) Upgrades and Downgrades:** Users should have a seamless upgrade/downgrade experience and should not be able to inadvertently subscribe to multiple variations of the same thing. Review [best practices](https://developer.apple.com/app-store/subscriptions/) on managing your subscription upgrade and downgrade options.
  + **3.1.2(c) Subscription Information:** Before asking a customer to subscribe, you should clearly describe what the user will get for the price. How many issues per month? How much cloud storage? What kind of access to your service? Ensure you clearly communicate the requirements described in Schedule 2 of the Apple Developer Program License Agreement, found in [Agreements, Tax, and Banking](http://www.apple.com/itunes/go/itunesconnect/contracts).
  + **3.1.3(a) “Reader” Apps:** Apps may allow a user to access previously purchased content or content subscriptions (specifically: magazines, newspapers, books, audio, music, video, access to professional databases, VoIP, cloud storage, and approved services such as classroom management apps), provided that you agree not to directly or indirectly target iOS users to use a purchasing method other than in-app purchase, and your general communications about other purchasing methods are not designed to discourage use of in-app purchase.
  + **3.1.3(b) Multiplatform Services:** Apps that operate across multiple platforms may allow users to access content, subscriptions, or features they have acquired elsewhere, including consumable items in multi-platform games, provided those items are also available as in-app purchases within the app. You must not directly or indirectly target iOS users to use a purchasing method other than in-app purchase, and your general communications about other purchasing methods must not discourage use of in-app purchase.
  + **3.1.4 Hardware-Specific Content:** In limited circumstances, such as when features are dependent upon specific hardware to function, the app may unlock that functionality without using in-app purchase (e.g. an astronomy app that adds features when synced with a telescope). App features that work in combination with an approved physical product (such as a toy) on an optional basis may unlock functionality without using in-app purchase, provided that an in-app purchase option is available as well. You may not, however, require users to purchase unrelated products or engage in advertising or marketing activities to unlock app functionality.
  + **3.1.5(a) Goods and Services Outside of the App:** If your app enables people to purchase goods or services that will be consumed outside of the app, you must use purchase methods other than in-app purchase to collect those payments, such as Apple Pay or traditional credit card entry.
  + **3.1.5(b) Cryptocurrencies:**
    - (i) Wallets: Apps may facilitate virtual currency storage, provided they are offered by developers enrolled as an organization.
    - (ii) Mining: Apps may not mine for cryptocurrencies unless the processing is performed off device (e.g. cloud-based mining).
    - (iii) Exchanges: Apps may facilitate transactions or transmissions of cryptocurrency on an approved exchange, provided they are offered by the exchange itself.
    - (iv) Initial Coin Offerings: Apps facilitating Initial Coin Offerings (“ICOs”), cryptocurrency futures trading, and other crypto-securities or quasi-securities trading must come from established banks, securities firms, futures commission merchants (“FCM”), or other approved financial institutions and must comply with all applicable law.
    - (v) Cryptocurrency apps may not offer currency for completing tasks, such as downloading other apps, encouraging other users to download, posting to social networks, etc.
  + **3.1.6 Apple Pay:** Apps using Apple Pay must provide all material purchase information to the user prior to sale of any good or service and must use Apple Pay branding and user interface elements correctly, as described in the [Apple Pay Identity Guidelines](https://developer.apple.com/apple-pay/marketing/) and [Human Interface Guidelines](https://developer.apple.com/design/human-interface-guidelines/apple-pay/). Apps using Apple Pay to offer recurring payments must, at a minimum, disclose the following information:
    - The length of the renewal term and the fact that it will continue until canceled
    - What will be provided during each period
    - The actual charges that will be billed to the customer
    - How to cancel
  + **3.1.7 Advertising:** Ads displayed in an app must be appropriate for the app’s age rating, allow the user to see all information used to target them for that ad (without requiring the user to leave the app), and may not engage in targeted or behavioral advertising based on sensitive user data such as health/medical data (e.g. from the HealthKit APIs), school and classroom data (e.g. from ClassKit), or from kids (e.g. from apps in the Kids Category), etc. Interstitial ads or ads that interrupt or block the user experience must clearly indicate that they are an ad, must not manipulate or trick users into tapping into them, and must provide easily accessible and visible close/skip buttons large enough for people to easily dismiss the ad.
* **3.2 Other Business Model Issues**

The lists below are not exhaustive, and your submission may trigger a change or update to our policies, but here are some additional do’s and don’ts to keep in mind:

* + **3.2.1 Acceptable**
    - **(i)** Displaying your own apps for purchase or promotion within your app, provided the app is not merely a catalog of your apps.
    - **(ii)** Displaying or recommending a collection of third-party apps that are designed for a specific approved need (e.g. health management, aviation, accessibility). Your app should provide robust editorial content so that it doesn’t seem like a mere storefront.
    - **(iii)** Disabling access to specific approved rental content (e.g. films, television programs, music, books) after the rental period has expired; all other items and services may not expire.
    - **(iv)** Wallet passes can be used to make or receive payments, transmit offers, or offer identification (such as movie tickets, coupons, and VIP credentials). Other uses may result in the rejection of the app and the revocation of Wallet credentials.
    - **(v)** Insurance apps must be free, in legal-compliance in the regions distributed, and cannot use in-app purchase.
    - **(vi)** Approved nonprofits may fundraise directly within their own apps or third-party apps, provided those fundraising campaigns adhere to all App Review Guidelines and offer Apple Pay support. These apps must disclose how the funds will be used, abide by all required local and federal laws, and ensure appropriate tax receipts are available to donors. Additional information shall be provided to App Review upon request. Nonprofit platforms that connect donors to other nonprofits must ensure that every nonprofit listed in the app has also gone through the nonprofit approval process. Learn more about becoming an [approved nonprofit](https://developer.apple.com/support/apple-pay-nonprofits/).
    - **(vii)** Apps may enable individual users to give a monetary gift to another individual without using in-app purchase, provided that (a) the gift is a completely optional choice by the giver, and (b) 100% of the funds go to the receiver of the gift. However, a gift that is connected to or associated at any point in time with receiving digital content or services must use in-app purchase.
    - **(viii)** Apps used for financial trading, investing, or money management should come from the financial institution performing such services or must use a public API offered by the institution in compliance with its Terms & Conditions.
  + **3.2.2 Unacceptable**
    - **(i)** Creating an interface for displaying third-party apps, extensions, or plug-ins similar to the App Store or as a general-interest collection.
    - **(ii)** Monetizing built-in capabilities provided by the hardware or operating system, such as Push Notifications, the camera, or the gyroscope; or Apple services, such as Apple Music access or iCloud storage.
    - **(iii)** Artificially increasing the number of impressions or click-throughs of ads, as well as apps that are designed predominantly for the display of ads.
    - **(iv)** Unless you are an approved nonprofit or otherwise permitted under Section 3.2.1 (vi) above, collecting funds within the app for charities and fundraisers. Apps that seek to raise money for such causes must be free on the App Store and may only collect funds outside of the app, such as via Safari or SMS.
    - **(v)** Arbitrarily restricting who may use the app, such as by location or carrier.
    - **(vi)** Apps should allow a user to get what they’ve paid for without performing additional tasks, such as posting on social media, uploading contacts, checking in to the app a certain number of times, etc. Apps should not require users to rate the app, review the app, watch videos, download other apps, tap on advertisements, or take other similar actions in order to access functionality, content, use the app, or receive monetary or other compensation, including but not limited to gift cards and codes.
    - **(vii)** Artificially manipulating a user’s visibility, status, or rank on other services unless permitted by that service’s Terms and Conditions.
    - **(viii)** Apps that facilitate binary options trading are not permitted on the App Store. Consider a web app instead. Apps that facilitate trading in contracts for difference (“CFDs”) or other derivatives (e.g. FOREX) must be properly licensed in all jurisdictions where the service is available.
    - **(ix)** Apps must not force users to rate the app, review the app, download other apps, or perform other similar actions in order to access functionality, content, or use of the app.

### 4. Design

Apple customers place a high value on products that are simple, refined, innovative, and easy to use, and that’s what we want to see on the App Store. Coming up with a great design is up to you, but the following are minimum standards for approval to the App Store. And remember that even after your app has been approved, you should update your app to ensure it remains functional and engaging to new and existing customers. Apps that stop working or offer a degraded experience may be removed from the App Store at any time.

* **4.1 Copycats**

Come up with your own ideas. We know you have them, so make yours come to life. Don’t simply copy the latest popular app on the App Store, or make some minor changes to another app’s name or UI and pass it off as your own. In addition to risking an intellectual property infringement claim, it makes the App Store harder to navigate and just isn’t fair to your fellow developers.

* **4.2 Minimum Functionality**

Your app should include features, content, and UI that elevate it beyond a repackaged website. If your app is not particularly useful, unique, or “app-like,” it doesn’t belong on the App Store. If your App doesn’t provide some sort of lasting entertainment value, it may not be accepted. Apps that are simply a song or movie should be submitted to the iTunes Store. Apps that are simply a book or game guide should be submitted to the Apple Books Store.

* + **4.2.1** Apps using ARKit should provide rich and integrated augmented reality experiences; merely dropping a model into an AR view or replaying animation is not enough.
  + **4.2.2** Other than catalogs, apps shouldn’t primarily be marketing materials, advertisements, web clippings, content aggregators, or a collection of links.
  + **4.2.3**
    - (i) Your app should work on its own without requiring installation of another app to function.
    - (ii) Make sure you include sufficient content in the binary for the app to function at launch.
    - (iii) If your app needs to download additional resources, disclose the size of the download and prompt users before doing so. Existing apps must comply with this guideline in any update submitted after January 1, 2019.
  + **4.2.4** Apple Watch apps that appear to be a watch face are confusing, because people will expect them to work with device features such as swipes, notifications, and third-party complications. Creative ways of expressing time as an app interface is great (say, a tide clock for surfers), but if your app comes too close to resembling a watch face, we will reject it.
  + **4.2.5** Apps that are primarily iCloud and iCloud Drive file managers need to include additional app functionality to be approved.
  + **4.2.6** Apps created from a commercialized template or app generation service will be rejected unless they are submitted directly by the provider of the app’s content. These services should not submit apps on behalf of their clients and should offer tools that let their clients create customized, innovative apps that provide unique customer experiences. Another acceptable option for template providers is to create a single binary to host all client content in an aggregated or “picker” model, for example as a restaurant finder app with separate customized entries or pages for each client restaurant, or as an event app with separate entries for each client event.
  + **4.2.7 Remote Desktop Clients:** If your remote desktop app acts as a mirror of specific software or services rather than a generic mirror of the host device, it must comply with the following:
    - (a) The app must only connect to a user-owned host device that is a personal computer or dedicated game console owned by the user, and both the host device and client must be connected on a local and LAN-based network.
    - (b) Any software or services appearing in the client are fully executed on the host device, rendered on the screen of the host device, and may not use APIs or platform features beyond what is required to stream the Remote Desktop.
    - (c) All account creation and management must be initiated from the host device.
    - (d) The UI appearing on the client does not resemble an iOS or App Store view, does not provide a store-like interface, or include the ability to browse, select, or purchase software not already owned or licensed by the user. For the sake of clarity, transactions taking place within mirrored software do not need to use in-app purchase, provided the transactions are processed on the host device.
    - (e) Thin clients for cloud-based apps are not appropriate for the App Store.
* **4.3 Spam**

Don’t create multiple Bundle IDs of the same app. If your app has different versions for specific locations, sports teams, universities, etc., consider submitting a single app and provide the variations using in-app purchase. Also avoid piling on to a category that is already saturated; the App Store has enough fart, burp, flashlight, and Kama Sutra apps, etc. already. Spamming the store may lead to your removal from the Developer Program.

* **4.4 Extensions**

Apps hosting or containing extensions must comply with the [App Extension Programming Guide](https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/index.html#//apple_ref/doc/uid/TP40014214) or the [Safari Extensions Development Guide](https://developer.apple.com/library/archive/documentation/Tools/Conceptual/SafariExtensionGuide/Introduction/Introduction.html) and should include some functionality, such as help screens and settings interfaces where possible. You should clearly and accurately disclose what extensions are made available in the app’s marketing text, and the extensions may not include marketing, advertising, or in-app purchases.

* + **4.4.1** Keyboard extensions have some additional rules.

They must:

* + - Provide keyboard input functionality (e.g. typed characters);
    - Follow Sticker guidelines if the keyboard includes images or emoji;
    - Provide a method for progressing to the next keyboard;
    - Remain functional without full network access and without requiring full access;
    - Collect user activity only to enhance the functionality of the user’s keyboard extension on the iOS device.

They must not:

* + - Launch other apps besides Settings; or
    - Repurpose keyboard buttons for other behaviors (e.g. holding down the “return” key to launch the camera).
  + **4.4.2** Safari extensions must run on the current version of Safari on macOS. They may not interfere with System or Safari UI elements and must never include malicious or misleading content or code. Violating this rule will lead to removal from the Developer Program. Safari extensions should not claim access to more websites than strictly necessary to function.
  + **4.4.3** Stickers

Stickers are a great way to make Messages more dynamic and fun, letting people express themselves in clever, funny, meaningful ways. Whether your app contains a sticker extension or you’re creating free-standing sticker packs, its content shouldn’t offend users, create a negative experience, or violate the law.

* + - **(i)** In general, if it wouldn’t be suitable for the App Store, it doesn’t belong in a sticker.
    - **(ii)** Consider regional sensitivities, and do not make your sticker pack available in a country where it could be poorly received or violate local law.
    - **(iii)** If we don’t understand what your stickers mean, include a clear explanation in your review notes to avoid any delays in the review process.
    - **(iv)** Ensure your stickers have relevance beyond your friends and family; they should not be specific to personal events, groups, or relationships.
    - **(v)** You must have all the necessary copyright, trademark, publicity rights, and permissions for the content in your stickers, and shouldn’t submit anything unless you’re authorized to do so. Keep in mind that you must be able to provide verifiable documentation upon request. Apps with sticker content you don’t have rights to use will be removed from the App Store and repeat offenders will be removed from the Developer Program. If you believe your content has been infringed by another provider, [submit a claim here](https://www.apple.com/legal/internet-services/itunes/appstorenotices/#?lang=en).
* **4.5 Apple Sites and Services**
  + **4.5.1** Apps may use approved Apple RSS feeds such as the iTunes Store RSS feed, but may not scrape any information from Apple sites (e.g. apple.com, the iTunes Store, App Store, App Store Connect, developer portal, etc.) or create rankings using this information.
  + **4.5.2** Apple Music
    - **(i)** The MusicKit APIs let customers access their subscription while using your app. They are intended for simple music playback by Apple Music subscribers. Users must initiate the playback of an Apple Music stream and be able to navigate using standard media controls such as “play,” “pause,” and “skip.” Moreover, your app may not require payment or indirectly monetize access to the Apple Music service (e.g. in-app purchase, advertising, requesting user info, etc.). Do not download, upload, or enable sharing of music files sourced from the MusicKit APIs, except as explicitly permitted in [MusicKit](https://developer.apple.com/musickit/) documentation.
    - **(ii)** Using the MusicKit APIs is not a replacement for securing the licenses you might need for a deeper or more complex music integration. For example, if you want your app to play a specific song at a particular moment, or to create audio or video files that can be shared to social media, you’ll need to contact rights-holders directly to get their permission (e.g. synchronization or adaptation rights) and assets. Cover art and other metadata may only be used in connection with music playback or playlists (including App Store screenshots displaying your app’s functionality), and should not be used in any marketing or advertising without getting specific authorization from rights-holders. Make sure to follow the [Apple Music Identity Guidelines](https://www.apple.com/itunes/marketing-on-music/identity-guidelines.html) when integrating Apple Music services in your app.
    - **(iii)** Apps that access Apple Music user data, such as playlists and favorites, must clearly disclose this access in the purpose string. Any data collected may not be shared with third parties for any purpose other than supporting or improving the app experience. This data may not be used to identify users or devices, or to target advertising.
  + **4.5.3** Do not use Apple Services to spam, phish, or send unsolicited messages to customers, including Game Center, Push Notifications, etc. Do not attempt to reverse lookup, trace, relate, associate, mine, harvest, or otherwise exploit Player IDs, aliases, or other information obtained through Game Center, or you will be removed from the Developer Program.
  + **4.5.4** Push Notifications must not be required for the app to function, and should not be used for advertising, promotions, or direct marketing purposes or to send sensitive personal or confidential information. Abuse of these services may result in revocation of your privileges.
  + **4.5.5** Only use Game Center Player IDs in a manner approved by the Game Center terms and do not display them in the app or to any third-party.
  + **4.5.6** Apps may use Unicode characters that render as Apple emoji in their app and app metadata. Apple emoji may not be used on other platforms or embedded directly in your app binary.
* **4.6 Alternate App Icons**

Apps may display customized icons, for example, to reflect a sports team preference, provided that each change is initiated by the user and the app includes settings to revert to the original icon. All icon variants must relate to the content of the app and changes should be consistent across all system assets, so that the icons displayed in Settings, Notifications, etc. match the new springboard icon. This feature may not be used for dynamic, automatic, or serial changes, such as to reflect up-to-date weather information, calendar notifications, etc.

* **4.7 HTML5 Games, Bots, etc.**

Apps may contain or run code that is not embedded in the binary (e.g. HTML5-based games, bots, etc.), as long as code distribution isn’t the main purpose of the app, the code is not offered in a store or store-like interface, and provided that the software (1) is free or purchased using in-app purchase; (2) only uses capabilities available in a standard WebKit view (e.g. it must open and run natively in Safari without modifications or additional software); your app must use WebKit and JavaScript Core to run third-party software and should not attempt to extend or expose native platform APIs to third-party software; (3) is offered by developers that have joined the Apple Developer Program and signed the Apple Developer Program License Agreement; (4) does not provide access to real money gaming, lotteries, or charitable donations; (5) adheres to the terms of these App Review Guidelines (e.g. does not include objectionable content); and (6) does not support digital commerce. Upon request, you must provide an index of software and metadata available in your app. It must include Apple DeveloperProgram Team IDs for the providers of the software along with a URL which App Review can use to confirm that the software complies with the requirements above.

### 5. Legal

Apps must comply with all legal requirements in any location where you make them available (if you’re not sure, check with a lawyer). We know this stuff is complicated, but it is your responsibility to understand and make sure your app conforms with all local laws, not just the guidelines below. And of course, apps that solicit, promote, or encourage criminal or clearly reckless behavior will be rejected. In extreme cases, such as apps that are found to facilitate human trafficking and/or the exploitation of children, appropriate authorities will be notified.

* **5.1 Privacy**

Protecting user privacy is paramount in the Apple ecosystem, and you should use care when handling personal data to ensure you’ve complied with [privacy best practices](https://developer.apple.com/documentation/uikit/core_app/protecting_the_user_s_privacy), applicable laws and the terms of the [Apple Developer Program License Agreement](https://developer.apple.com/terms/), not to mention customer expectations. More particularly:

* + **5.1.1 Data Collection and Storage**
    - **(i) Privacy Policies:**All apps must include a link to their privacy policy in the App Store Connectmetadata field and within the app in an easily accessible manner. The privacy policy must clearly and explicitly:
      * Identify what data, if any, the app/service collects, how it collects that data, and all uses of that data.
      * Confirm that any third-party with whom an app shares user data (in compliance with these Guidelines) — such as analytics tools, advertising networks and third-party SDKs, as well as any parent, subsidiary or other related entities that will have access to user data — will provide the same or equal protection of user data as stated in the app’s privacy policy and required by these Guidelines.
      * Explain its data retention/deletion policies and describe how a user can revoke consent and/or request deletion of the user’s data.
    - **(ii) Permission**Apps that collect user or usage data must secure user consent for the collection, even if such data is considered to be anonymous at the time of or immediately following collection. Paid functionality must not be dependent on or require a user to grant access to this data. Apps must also provide the customer with an easily accessible and understandable way to withdraw consent. Ensure your purpose strings clearly and completely describe your use of the data. Apps that collect data for a legitimate interest without consent by relying on the terms of the European Union’s General Data Protection Regulation (“GDPR”) or similar statute must comply with all terms of that law. Learn more about [Requesting Permission](https://developer.apple.com/documentation/uikit/core_app/protecting_the_user_s_privacy).
    - **(iii) Data Minimization:**Apps should only request access to data relevant to the core functionality of the app and should only collect and use data that is required to accomplish the relevant task. Where possible, use the out-of-process picker or a share sheet rather than requesting full access to protected resources like Photos or Contacts.
    - **(iv) Access**Apps must respect the user’s permission settings and not attempt to manipulate, trick, or force people to consent to unnecessary data access. For example, apps that include the ability to post photos to a social network must not also require microphone access before allowing the user to upload photos. Where possible, provide alternative solutions for users who don’t grant consent. For example, if a user declines to share Location, offer the ability to manually enter an address.
    - **(v) Account Sign-In:**If your app doesn’t include significant account-based features, let people use it without a log-in. Apps may not require users to enter personal information to function, except when directly relevant to the core functionality of the app or required by law. If your core app functionality is not related to a specific social network (e.g. Facebook, WeChat, Weibo, Twitter, etc.), you must provide access without a login or via another mechanism. Pulling basic profile information, sharing to the social network, or inviting friends to use the app are not considered core app functionality. The app must also include a mechanism to revoke social network credentials and disable data access between the app and social network from within the app. An app may not store credentials or tokens to social networks off of the device and may only use such credentials or tokens to directly connect to the social network from the app itself while the app is in use.
    - **(vi)**Developers that use their apps to surreptitiously discover passwords or other private data will be removed from the Developer Program.
    - **(vii)**SafariViewController must be used to visibly present information to users; the controller may not be hidden or obscured by other views or layers. Additionally, an app may not use SafariViewController to track users without their knowledge and consent.
    - **(viii)**Apps that compile personal information from any source that is not directly from the user or without the user’s explicit consent, even public databases, are not permitted on the App Store.
  + **5.1.2 Data Use and Sharing**
    - **(i)**Unless otherwise permitted by law, you may not use, transmit, or share someone’s personal data without first obtaining their permission. You must provide access to information about how and where the data will be used. Data collected from apps may only be shared with third parties to improve the app or serve advertising (in compliance with the [Apple Developer Program License Agreement](https://developer.apple.com/terms/).). Apps that share user data without user consent or otherwise complying with data privacy laws may be removed from sale and may result in your removal from the Apple DeveloperProgram.
    - **(ii)**Data collected for one purpose may not be repurposed without further consent unless otherwise explicitly permitted by law.
    - **(iii)**Apps should not attempt to surreptitiously build a user profile based on collected data and may not attempt, facilitate, or encourage others to identify anonymous users or reconstruct user profiles based on data collected from Apple-provided APIs or any data that you say has been collected in an “anonymized,” “aggregated,” or otherwise non-identifiable way.
    - **(iv)**Do not use information from Contacts, Photos, or other APIs that access user data to build a contact database for your own use or for sale/distribution to third parties, and don’t collect information about which other apps are installed on a user’s device for the purposes of analytics or advertising/marketing.
    - **(v)**Do not contact people using information collected via a user’s Contacts or Photos, except at the explicit initiative of that user on an individualized basis; do not include a Select All option or default the selection of all contacts. You must provide the user with a clear description of how the message will appear to the recipient before sending it (e.g. What will the message say? Who will appear to be the sender?).
    - **(vi)**Data gathered from the HomeKit API, HealthKit, Consumer Health Records API, MovementDisorder APIs, ClassKit or from depth and/or facial mapping tools (e.g. ARKit, Camera APIs, or Photo APIs) may not be used for marketing, advertising or use-based data mining, including by third parties. Learn more about best practices for implementing [CallKit](https://developer.apple.com/documentation/callkit), [HealthKit](https://developer.apple.com/documentation/healthkit), [ClassKit](https://developer.apple.com/documentation/classkit), and [ARKit](https://developer.apple.com/documentation/arkit).
    - **(vii)**Apps using Apple Pay may only share user data acquired via Apple Pay with third parties to facilitate or improve delivery of goods and services.
  + **5.1.3 Health and Health Research**

Health, fitness, and medical data are especially sensitive and apps in this space have some additional rules to make sure customer privacy is protected:

* + - **(i)** Apps may not use or disclose to third parties data gathered in the health, fitness, and medical research context—including from the Clinical Health Records API, HealthKit API, Motion and Fitness, MovementDisorderAPIs, or health-related human subject research—for advertising, marketing, or other use-based data mining purposes other than improving health management, or for the purpose of health research, and then only with permission. Apps may, however, use a user’s health or fitness data to provide a benefit directly to that user (such as a reduced insurance premium), provided that the app is submitted by the entity providing the benefit, and the data is not be shared with a third party. You must disclose the specific health data that you are collecting from the device.
    - **(ii)** Apps must not write false or inaccurate data into HealthKit or any other medical research or health management apps, and may not store personal health information in iCloud.
    - **(iii)** Apps conducting health-related human subject research must obtain consent from participants or, in the case of minors, their parent or guardian. Such consent must include the (a) nature, purpose, and duration of the research; (b) procedures, risks, and benefits to the participant; (c) information about confidentiality and handling of data (including any sharing with third parties); (d) a point of contact for participant questions; and (e) the withdrawal process.
    - **(iv)** Apps conducting health-related human subject research must secure approval from an independent ethics review board. Proof of such approval must be provided upon request.
  + **5.1.4 Kids**

For many reasons, it is critical to use care when dealing with personal data from kids, and we encourage you to carefully review all the requirements for complying with laws like the Children’s Online Privacy Protection Act (“COPPA”), the European Union’s General Data Protection Regulation (“GDPR”), and any international or local equivalents.

Apps may ask for birthdate and parental contact information only for the purpose of complying with these statutes, but must include some useful functionality or entertainment value regardless of a person’s age.

Apps intended for kids may not include third-party advertising or analytics.

Moreover, apps in the Kids Category or those that collect, transmit, or have the capability to share personal information (e.g. name, address, email, location, photos, videos, drawings, the ability to chat, other personal data, or persistent identifiers used in combination with any of the above) from a minor must include a privacy policy and must comply with all applicable children’s privacy statutes. For the sake of clarity, the [parental gate requirement](https://developer.apple.com/app-store/review/guidelines/#kids-category) for the Kid’s Category is generally not the same as securing parental consent to collect personal data under these privacy statutes. Collecting and transmitting data to third parties from apps in the Kids category is not allowed.

* + **5.1.5 Location Services**

Use Location services in your app only when it is directly relevant to the features and services provided by the app. Location-based APIs shouldn’t be used to provide emergency services or autonomous control over vehicles, aircraft, and other devices, except for small devices such as lightweight drones and toys, or remote control car alarm systems, etc. Ensure that you notify and obtain consent before collecting, transmitting, or using location data. If your app uses location services, be sure to explain the purpose in your app; refer to the [Human Interface Guidelines](https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/requesting-permission/) for best practices on doing so.

* **5.2 Intellectual Property**

Make sure your app only includes content that you created or that you have a license to use. Your app may be removed if you’ve stepped over the line and used content without permission. Of course, this also means someone else’s app may be removed if they’ve “borrowed” from your work. If you believe your intellectual property has been infringed by another developer on the App Store, submit a claim via our [web form](http://www.apple.com/legal/internet-services/itunes/appstorenotices/). Laws differ in different countries, but at the very least, make sure to avoid the following common errors:

* + **5.2.1** Generally: Don’t use protected third-party material such as trademarks, copyrighted works, or patented ideas in your app without permission, and don’t include misleading, false, or copycat representations, names, or metadata in your app bundle or developer name. Apps should be submitted by the person or legal entity that owns or has licensed the intellectual property and other relevant rights and is responsible for offering any services provided by the app.
  + **5.2.2** Third-Party Sites/Services: If your app uses, accesses, monetizes access to, or displays content from a third-party service, ensure that you are specifically permitted to do so under the service’s terms of use. Authorization must be provided upon request.
  + **5.2.3** Audio/Video Downloading: Apps should not facilitate illegal file sharing or include the ability to save, convert, or download media from third-party sources (e.g. Apple Music, YouTube, SoundCloud, Vimeo, etc.) without explicit authorization from those sources. Streaming of audio/video content may also violate Terms of Use, so be sure to check before your app accesses those services. Documentation must be provided upon request.
  + **5.2.4** Apple Endorsements: Don’t suggest or infer that Apple is a source or supplier of the App, or that Apple endorses any particular representation regarding quality or functionality. If your app is selected as an “Editor’s Choice,” Apple will apply the badge automatically.
  + **5.2.5** Apple Products: Don’t create an app that appears confusingly similar to an existing Apple product, interface (e.g. Finder), app (such as the App Store, iTunes Store, or Messages) or advertising theme. Apps and extensions, including third-party keyboards and Sticker packs, may not include Apple emoji. iTunes music previews may not be used for their entertainment value (e.g. as the background music to a photo collage or the soundtrack to a game) or in any other unauthorized manner. If your app displays Activity rings, they should not visualize Move, Exercise, or Stand data in a way that resembles the Activity control. The [Human Interface Guidelines](https://developer.apple.com/design/human-interface-guidelines/watchos/system-capabilities/health-and-fitness/#activity-rings) have more information on how to use Activity rings.
* **5.3 Gaming, Gambling, and Lotteries**

Gambling, gaming, and lotteries can be tricky to manage and tend to be one of the most regulated offerings on the App Store. Only include this functionality if you’ve fully vetted your legal obligations everywhere you make your app available and are prepared for extra time during the review process. Some things to keep in mind:

* + **5.3.1** Sweepstakes and contests must be sponsored by the developer of the app.
  + **5.3.2** Official rules for sweepstakes, contests, and raffles must be presented in the app and make clear that Apple is not a sponsor or involved in the activity in any manner.
  + **5.3.3** Apps may not use in-app purchase to purchase credit or currency for use in conjunction with real money gaming of any kind, and may not enable people to purchase lottery or raffle tickets or initiate fund transfers in the app.
  + **5.3.4** Apps that offer real money gaming (e.g. sports betting, poker, casino games, horse racing) or lotteries must have necessary licensing and permissions in the locations where the App is used, must be geo-restricted to those locations, and must be free on the App Store. Illegal gambling aids, including card counters, are not permitted on the App Store. Lottery apps must have consideration, chance, and a prize.
* **5.4 VPN Apps**

Apps offering VPN services must utilize the [NEVPNManager API](https://developer.apple.com/documentation/networkextension/nevpnmanager) and may only be offered by developers enrolled as an organization. You must make a clear declaration of what user data will be collected and how it will be used on an app screen prior to any user action to purchase or otherwise use the service. Apps offering VPN services may not sell, use, or disclose to third parties any data for any purpose, and must commit to this in their privacy policy. VPN apps must not violate local laws, and if you choose to make your VPN app available in a territory that requires a VPN license, you must provide your license information in the App Review Notes field. Parental control, content blocking, and security apps, among others, from approved providers may also use the NEVPNManager API. Apps that do not comply with this guideline will be removed from the App Store and you may be removed from the Apple Developer Program.

* **5.5 Mobile Device Management**

Mobile Device Management Apps that offer Mobile Device Management (MDM) services must request this capability from Apple. Such apps may only be offered by commercial enterprises (such as business organizations, educational institutions, or government agencies), and in limited cases, companies using MDM for parental control services. You must make a clear declaration of what user data will be collected and how it will be used on an app screen prior to any user action to purchase or otherwise use the service. MDM apps must not violate local laws. Apps offering MDM services may not sell, use, or disclose to third parties any data for any purpose, and must commit to this in their privacy policy. Apps that do not comply with this guideline will be removed from the App Store and you may be removed from the Apple Developer Program.

* **5.6 Developer Code of Conduct**

Please treat everyone with respect, whether in your responses to App Store reviews, customer support requests, or when communicating with Apple, including your responses in Resolution Center. Do not engage in harassment of any kind, discriminatory practices, intimidation, bullying, and don’t encourage others to engage in any of the above.

Customer trust is the cornerstone of the App Store’s success. Apps should never prey on users or attempt to rip-off customers, trick them into making unwanted purchases, force them to share unnecessary data, raise prices in a tricky manner, charge for features or content that are not delivered, or engage in any other manipulative practices within or outside of the app.

Last Updated: 3 June 2019

### After You Submit

Once you’ve submitted your app and metadata in App Store Connect and you’re in the review process, here are some things to keep in mind:

* **Timing**: App Review will examine your app as soon as we can. However, if your app is complex or presents new issues, it may require greater scrutiny and consideration. And remember that if your app is repeatedly rejected for the same guideline violation or you’ve attempted to manipulate the App Review process, review of your app will take longer to complete. Learn more about [App Review](https://developer.apple.com/support/app-review/).
* **Status Updates**: The current status of your app will be reflected in App Store Connect, so you can keep an eye on things from there.
* **Expedite Requests**: If you have a critical timing issue, you can [request an expedited review](https://developer.apple.com/contact/app-store/?topic=expedite). Please respect your fellow developers by seeking expedited review only when you truly need it. If we find you’re abusing this system, we may reject your requests going forward.
* **Release Date**: If your release date is set for the future, the app will not appear on the App Store until that date, even if it is approved by App Review. And remember that it can take up to 24-hours for your app to appear on all selected storefronts.
* **Rejections**: Our goal is to apply these guidelines fairly and consistently, but nobody’s perfect. If your app has been rejected and you have questions or would like to provide additional information, please use the Resolution Center to communicate directly with the App Review team. This may help get your app on the store, and it can help us improve the App Review process or identify a need for clarity in our policies. If you still disagree with the outcome, please [submit an appeal](https://developer.apple.com/contact/app-store/?topic=appeal).

We’re excited to see what you come up with next!

**Authorized Use of Apple Trademarks**

**1. Advertising, Promotional, and Sales Materials:** Only Apple and its authorized resellers and licensees may use the Apple Logo in advertising, promotional, and sales materials. Such authorized parties may use the Apple Logo only as specified in their agreement with Apple and any associated Guidelines and such use must always be in conjunction with the appropriate terms that define the relationship authorized by their contract with Apple. For example:

Authorized Reseller

Authorized Value Added Reseller

Authorized Service Provider

Authorized Wholesaler

**2. Compatibility:** Developers may use Apple, Macintosh, iMac, or any other Apple word mark (but not the Apple Logo or other Apple-owned graphic symbol/logo) in a **referential phrase** on packaging or promotional/advertising materials to describe that the third party product is compatible with the referenced Apple product or technology, provided they comply with the following requirements.

a. The Apple word mark is not part of the product name.

b. The Apple word mark is used in a referential phrase such as “runs on,” “for use with,” “for,” or “compatible with.”

c. The Apple word mark appears less prominent than the product name.

d. The product is in fact compatible with, or otherwise works with, the referenced Apple product.

e. The reference to Apple does not create a sense of endorsement, sponsorship, or false association with Apple or Apple products or services.

f. The use does not show Apple or its products in a false or derogatory light.

**3. Publications, Seminars, and Conferences:** You may use an Apple word mark in connection with book titles, magazines, periodicals, seminars, or conferences provided you comply with the following requirements:

a. The use is referential and less prominent than the rest of the title. Acceptable: XYZ CONFERENCE for Macintosh Computer Users.

b. The use reflects favorably on both Apple and Apple products or technology.

c. Your name and logo appear more prominent than the Apple word mark on all printed materials related to the publication, seminar or conference.

d. The Apple logo or any other Apple-owned graphic symbol, logo, icon or image does not appear on or in the publication or on any materials related to the publication, seminar, or conference without express written permission from Apple.

e. A disclaimer of sponsorship, affiliation, or endorsement by Apple, similar to the following, is included on the publication and on all related printed materials: “(Title) is an independent (publication) and has not been authorized, sponsored, or otherwise approved by Apple Inc.”

f. A trademark attribution notice is included in the credit section giving notice of Apple’s ownership of its trademark(s). Please refer to the section below titled “Proper Trademark Notice and Attribution.”

**4. Web Sites:** Web sites that serve only as noncommercial electronic informational forums concerning an Apple product or technology may use the appropriate Apple word mark, provided such use complies with the guidelines set forth in Section 3 above.

**5. Apple Web Badge Licensing Program:** Web sites may use one of the Apple Web Badges if the site uses or was created using Apple-branded hardware or software and you comply with the terms of the Apple Web Badges License Agreement and Guidelines. For more information see [Apple Web Badges](https://www.apple.com/about/webbadges/).

**Unauthorized Use of Apple Trademarks**

**1. Company, Product, or Service Name:** You may not use or register, in whole or in part, Apple, iPod, iTunes, Macintosh, iMac, or any other Apple trademark, including Apple-owned graphic symbols, logos, icons, or an alteration thereof, as or as part of a company name, trade name, product name, or service name except as specifically noted in these guidelines.

**2. Apple Logo and Apple-owned Graphic Symbols:** You may not use the Apple Logo or any other Apple-owned graphic symbol, logo, or icon on or in connection with web sites, products, packaging, manuals, promotional/advertising materials, or for any other purpose except pursuant to an express written trademark license from Apple, such as a reseller agreement.

**3. Variations, Takeoffs or Abbreviations:** You may not use an image of a real apple or other variation of the Apple logo for any purpose. Third parties cannot use a variation, phonetic equivalent, foreign language equivalent, takeoff, or abbreviation of an Apple trademark for any purpose. For example:

Not acceptable:   Appletree      Jackintosh      Apple Cart      iPodMart

**4. Disparaging Manner:** You may not use an Apple trademark or any other Apple-owned graphic symbol, logo, or icon in a disparaging manner.

**5. Endorsement or Sponsorship:** You may not use Apple, Macintosh, iMac, or any other Apple trademark, including Apple-owned graphic symbols/logos, or icons, in a manner that would imply Apple’s affiliation with or endorsement, sponsorship, or support of a third party product or service.

**6. Merchandise Items:** You may not manufacture, sell or give-away merchandise items, such as T-shirts and mugs, bearing Apple, Macintosh, iMac or any other Apple trademark, including symbols, logos, or icons, except pursuant to an express written trademark license from Apple.

# viewWithTag:

Returns the view whose tag matches the specified value.

**Apple Wallet** (**Wallet** for short, formerly **Apple Passbook**) is a [mobile app](https://en.wikipedia.org/wiki/Mobile_app) included with the [iOS](https://en.wikipedia.org/wiki/IOS) operating system that allows users to store *Wallet-passes*, meaning [coupons](https://en.wikipedia.org/wiki/Coupon), [boarding passes](https://en.wikipedia.org/wiki/Boarding_pass), [student ID](https://en.wikipedia.org/wiki/Campus_card) cards, event [tickets](https://en.wikipedia.org/wiki/Ticket_(admission)), [movie](https://en.wikipedia.org/wiki/Movie) tickets, [public transportation](https://en.wikipedia.org/wiki/Public_transport) tickets, [store](https://en.wikipedia.org/wiki/Retail) cards, and – starting with [iOS 8.1](https://en.wikipedia.org/wiki/IOS_8) – [credit cards](https://en.wikipedia.org/wiki/Credit_card), [debit cards](https://en.wikipedia.org/wiki/Debit_card), [prepaid cards](https://en.wikipedia.org/wiki/Prepaid_debit_card), and [loyalty cards](https://en.wikipedia.org/wiki/Loyalty_program) via [Apple Pay](https://en.wikipedia.org/wiki/Apple_Pay).[[1]](https://en.wikipedia.org/wiki/Apple_Wallet#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Apple_Wallet#cite_note-2) It was designed by [Apple Inc.](https://en.wikipedia.org/wiki/Apple_Inc.) and was presented at the 2012 [Apple Worldwide Developers Conference](https://en.wikipedia.org/wiki/Apple_Worldwide_Developers_Conference)(WWDC) on June 11, 2012.[[3]](https://en.wikipedia.org/wiki/Apple_Wallet#cite_note-enwwdc-3) The app first appeared on [iOS 6](https://en.wikipedia.org/wiki/IOS_6) on September 19, 2012. Apple Wallet will also be the main interface with [Apple Card](https://en.wikipedia.org/wiki/Apple_Card), Apple's newly announced credit card service.

**Apple Pay** is a [mobile payment](https://en.wikipedia.org/wiki/Mobile_payment) and [digital wallet](https://en.wikipedia.org/wiki/Digital_wallet) service by [Apple Inc.](https://en.wikipedia.org/wiki/Apple_Inc.) that allows users to make [payments](https://en.wikipedia.org/wiki/Payment) in person, in [iOS apps](https://en.wikipedia.org/wiki/IOS_app), and on the web. It is supported on the [iPhone](https://en.wikipedia.org/wiki/IPhone), [Apple Watch](https://en.wikipedia.org/wiki/Apple_Watch), [iPad](https://en.wikipedia.org/wiki/IPad), and [Mac](https://en.wikipedia.org/wiki/Macintosh). It digitizes and can replace a [credit](https://en.wikipedia.org/wiki/Credit_card) or [debit](https://en.wikipedia.org/wiki/Debit_card)card [chip and PIN](https://en.wikipedia.org/wiki/Chip_and_PIN) transaction at a contactless-capable [point-of-sale terminal](https://en.wikipedia.org/wiki/Point-of-sale_terminal). Apple Pay does not require Apple Pay-specific [contactless payment](https://en.wikipedia.org/wiki/Contactless_payment) terminals; it works with any merchant that accepts contactless payments.[[1]](https://en.wikipedia.org/wiki/Apple_Pay#cite_note-pocket-lint.com-1) It is similar to contactless payments used in other countries, with the addition of [two-factor authentication](https://en.wikipedia.org/wiki/Two-factor_authentication) via Touch ID, [Face ID](https://en.wikipedia.org/wiki/Face_ID), PIN, or passcode. Devices wirelessly communicate with point of sale systems using [near field communication](https://en.wikipedia.org/wiki/Near_field_communication) (NFC), in conjunction with an embedded secure element (eSE) to securely store payment data and perform cryptographic functions, and Apple's [Touch ID](https://en.wikipedia.org/wiki/Touch_ID)and [Face ID](https://en.wikipedia.org/wiki/Face_ID) for biometric authentication.

Use Apple Pay on the App Store to buy apps and games, or within apps to pay for a ride, a pizza delivery, or a new pair of sneakers — with just a touch or a glance. You can also use Apple Pay for a subscription to Apple Music, upgraded iCloud storage, and other Apple services.

## Where can you use Apple Pay?

Apple Pay works anywhere that accepts contactless payments. If you see the contactless payments symbol or the Apple Pay symbol near readers at the checkout, they will accept Apple Pay.

It's supported by hundreds of thousands of stores and restaurants worldwide. Over 75 per cent of stores and restaurants in the US support Apple Pay and over 85 per cent in the UK. Australia has support in 99 per cent of its stores and restaurants.

A few of the US stores and restaurants that accept Apple Pay include: Bloomingdales, Disney, Duane Reade, Macy's, McDonalds, Nike, Petco, Staples, Subway, Unleashed, Walgreens, Whole Foods, etc. You can view a [complete list of stores here](http://www.apple.com/apple-pay/where-to-use-apple-pay/).

The UK has an established contactless payment system and you'll be able to use Apple pay at all of the stores and restaurants you already use contactless at.

You can also use Apple Pay within many apps and through the Safari web browser.

Apple Pay is a contactless payment technology for [Apple devices](https://www.pocket-lint.com/phones/reviews/apple). It was designed to move consumers away from physical wallets into a world where your debit and credit cards are on your [iPhone](https://www.pocket-lint.com/phones/reviews/apple) or [Apple Watch](https://www.pocket-lint.com/smartwatches/reviews/apple), allowing you to pay using your device instead of a card.

<https://medium.com/appcoda-tutorials/integrating-basic-apple-pay-into-your-ios-app-71f17d48fc9b>

## 4GB's is the maximum size your iOS app can be.

[iOS 13 removes 200 MB file size limit for app downloads over cellular](https://9to5mac.com/2019/06/03/ios-13-removes-200-mb-file-size-limit-for-app-downloads-over-cellular/)

Apple upped the iPhone cellular limit from 150 MB to 200 MB [just last week](https://9to5mac.com/2019/05/31/apple-iphone-cellular-limit-increased/) .. and now that limit is being removed altogether.

With iOS 13, Apple will warn users when they attempt to start a large download on a mobile network but you are allowed to override it.

In this case, the App Store version of PUBG is over 2 gigabytes in size. When attempting to download the game on iOS 12, it simply wouldn’t be allowed on mobile data.

In iOS 13, the system now shows a dialog warning that the app size is large, noting the file size. Users can choose to continue downloading or schedule it to happen later when the device returns to a Wi-Fi connection.

This behavior is even customizable in Settings. There is a new preference in iTunes & App Store named ‘App Downloads’. The default option is ‘Ask if Over 200 MB’ and shows the above pictured alert.

4GB's is the maximum size your iOS app can be.

Your app will not get rejected for being to small, they can get rejected for having no functionality though! I have had apps which were under 1 MB back in the early days. If you are concerned because you have a bunch of images in the app and expect it to be much larger than 3.1 MB, I would do some investigation to see if the images were added correctly and are in fact going into the app when you archive it.

The main part of the app is without a doubt the executable file. The executable is usually not very large, because it's just the compiled code that the machine runs. In small, trivial, apps, this is usually only a few kilobytes (KB). In more complex apps it can make it up to a few megabytes (MB). When compiling for multiple architectures the size of your app will inevitably increase (see [my question on app size when compiling for 64-bit iOS devices](https://stackoverflow.com/questions/19646492/ios-64-bit-app-doubles-in-size)).

Generally (not always), the bulk of your app's size is composed of interface files, images, videos, sounds, resources, etc. In other words, **no, the executable is not the whole app**.

Open up any app (in some kind of file viewer) and take a look at it's contents, it has four folders:

* The bundle - with the executable and resources
* The documents folder - which can also take up substantial space
* The cache and temp directories

Many games are very large in size, Infinity Blade for example is about 1.2 GB for the initial download. But Infinity Blade's size is because of the copious amount of images / graphics, not the executable.

I think that Apple is trying to prevent malicious software and iOS / Device memory limits. Honestly, I think it'd be very hard to get the executable itself above 60 MB. Like I said before, the bulk of app size is comprised of resources.

Apple has increased the over-the-air download limit for the [App Store](https://www.macrumors.com/roundup/app-store/) to 200MB, up from 150MB.

[Apple's](https://www.cnet.com/apple/) cellular download limits for apps increased this week from 150MB to 200MB in the App Store. The change was spotted earlier by [9to5Mac](https://9to5mac.com/2019/05/31/apple-iphone-cellular-limit-increased/). This means users will not see a message telling them to connect to [Wi-Fi](https://www.cnet.com/tags/wi-fi/) to download apps below the new file size limit.

The 100 MB download size limits are designed to improve the experience for users over cellular. Large apps can take a long time and may need to be restarted under poor network conditions. This can lead to significant amounts of data being used on their cellular plan.

Their are also total limits on app size (currently 4 GB) are there to avoid apps that crowd out other apps and media from the device.

All of these limits have been moved up over time as cellular plans, speeds and device storage have increased over time.

There are plenty of techniques out there to decrease app size, app thinning, compression, it is best for users, the cell phone networks and Apple when developers provide the smallest possible files for their apps.

It’s also good for a developer to not be the largest app on a users device. When they need space the big ones tend to get deleted first.

**iOS App binary files can be as large as 2 GB**, but the executable file (app\_name.app/app\_name) cannot exceed 60 MB. However, consider download times when determining your app’s size. Minimize the file’s size as much as possible, keeping in mind that **there is a 100 MB limit for over-the-air downloads**.

Instead, according to an Apple developer note, "Starting June 1, 2016 all apps submitted to the [App Store must support IPv6-only networking](https://developer.apple.com/news/?id=05042016a)." That's because, starting with [iOS 9](https://www.zdnet.com/article/ios-9-and-os-x-el-capitan-september-release-date-revealed/), iOS has been migrating up IPv4 to IPv6.

**IPv4** and **IPv6** are the versions of internet protocol where IPv6 is the enhanced version of IPv4. There are various differences between IPv4 and IPv6 protocol including their features, but the crucial one is the number of addresses (Address space) it generates.

IP version 4 (IPv4) generates 4.29 x 109 unique network addresses which is insufficient in quantity and as a result Internet is running out of space. Whereas IP version 6 (IPv6) produces 3.4 x 1038 addresses and is a scalable and flexible solution to the current problem.

An IPv4 address is a 32- bit binary value, which can be displayed as four decimal digits. The IPv4 address space offers about 4.3 billion addresses. Only 3.7 billion addresses can only be assigned out of 4.3 billion address. The other addresses are conserved for specific purposes such as multicasting, private address space, loopback testing, and research.  
IP version 4 (IPv4) uses Broadcasting for transferring packets from one computer to all computers; this probably generates problems sometimes.

WebKit provides a set of classes to display web content in windows, and implements browser features such as following links when clicked by the user, managing a back-forward list, and managing a history of pages recently visited. WebKit greatly simplifies the complicated process of loading webpages—that is, asynchronously requesting web content from an HTTP server where the response may arrive incrementally, in random order, or partially due to network errors. WebKit also simplifies the process of displaying that content which can contain various MIME types, and compound frame elements each with their own set of scroll bars.

**Important**

Call WebKit functions and methods only from your app’s main thread or main dispatch queue.

### sort and sorted

Imagine we have an array of integers:



|  |  |
| --- | --- |
| 1 | var intArray = [34,13,14,584,1,34,69,3,596,2] |

If the members of the array have a type that confirms to the Comparable protocol, then you can use the function sorted() to get an ordered array. Int does confirm to Comparable, so let’s do it:



|  |  |
| --- | --- |
| 1  2  3 | let result = intArray.sorted()    print(result) //[1, 2, 3, 13, 14, 34, 34, 69, 584, 596] |

So, that was very easy! Alternatively, we can also use the function sort(). In contrast tosorted(), it doesn’t return a new array but modifies the existing one. Of course this is only possible if the array is not a constant.



|  |  |
| --- | --- |
| 1  2  3 | intArray.sort()    print(intArray) //[1, 2, 3, 13, 14, 34, 34, 69, 584, 596] |

However, not all types confirm to the Comparable protocol.

### Sorting arrays of dates

A good example for a type that doesn’t confirm to the Comparable protocol is Date. So if we want to sort an array of Date, then we can use



|  |  |
| --- | --- |
| 1 | public func sorted(by areInIncreasingOrder: (Element, Element) -> Bool) -> [Element] |

or



|  |  |
| --- | --- |
| 1 | public mutating func sort(by areInIncreasingOrder: (Element, Element) -> Bool) |

The only difference to the functions in the previous paragraph is, that we have to hand over a closure that has two array elements as parameters and returns a boolean. If the two elements are already in the right order, we have to return true. Sounds very straightforward. Let’s take a look at an example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var dateArray = [Date]()    let dateFormatter = DateFormatter()  dateFormatter.dateFormat = "MM/dd/yyyy"    dateArray.append(dateFormatter.date(from:"02/03/1985")!)  dateArray.append(dateFormatter.date(from: "12/01/2025")!)  dateArray.append(dateFormatter.date(from: "07/08/1964")!)  dateArray.append(dateFormatter.date(from: "09/04/2016")!)  dateArray.append(dateFormatter.date(from: "01/01/2000")!)  dateArray.append(dateFormatter.date(from: "12/12/1903")!)  dateArray.append(dateFormatter.date(from: "04/23/2222")!)  dateArray.append(dateFormatter.date(from: "08/06/1957")!)  dateArray.append(dateFormatter.date(from: "11/11/1911")!)  dateArray.append(dateFormatter.date(from: "02/05/1961")!)    dateArray.sort { (date1, date2) -> Bool in      return date1.compare(date2) == ComparisonResult.orderedAscending  }    for date in dateArray {      print(dateFormatter.string(from: date))  } |

First convert NSMutableArray to the Array by using below line of code.

let swiftArray = mutableArray as AnyObject as! [String]

Use below line of code to sort the Array.

var sortedArray = names.sorted { $0.localizedCaseInsensitiveCompare($1) == NSComparisonResult.OrderedAscending }

let sortedNames = names.sort { $0.name < $1.name }

let testArray = ["25 Jun, 2016", "30 Jun, 2016", "28 Jun, 2016", "2 Jul, 2016"]

var convertedArray: [Date] = []

var dateFormatter = DateFormatter()

dateFormatter.dateFormat = "dd MM, yyyy"// yyyy-MM-dd"

for dat in testArray {

let date = dateFormatter.date(from: dat)

if let date = date {

convertedArray.append(date)

}

}

var ready = convertedArray.sorted(by: { $0.compare($1) == .orderedDescending })

print(ready)

##### **MongoDB is a document database, which means it stores data in JSON-like documents. We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.**

**MongoDB** is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [document-oriented database](https://en.wikipedia.org/wiki/Document-oriented_database) program. Classified as a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) database program, MongoDB uses [JSON](https://en.wikipedia.org/wiki/JSON)-like documents with [schema](https://en.wikipedia.org/wiki/Database_schema). MongoDB is developed by [MongoDB Inc.](https://en.wikipedia.org/wiki/MongoDB_Inc.) and licensed under the Server Side Public License (SSPL).

These are tips that you’ll always want to implement in any app you develop.

1. [Use ARC to Manage Memory](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#arc)
2. [Use a reuseIdentifier Where Appropriate](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#reuse)
3. [Set Views as Opaque When Possible](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#opaque)
4. [Avoid Fat XIBs](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#smallxibs)
5. [Don’t Block the Main Thread](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#mainthread)
6. [Size Images to Image Views](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#imageviews)
7. [Choose the Correct Collection](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#collection)
8. [Enable GZIP Compression](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#gzip)

**Intermediate**

These are tips you should use when you run into slightly more complicated issues.

1. [Reuse and Lazy Load Views](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#lazyviews)
2. [Cache, Cache, Cache](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#cache)
3. [Consider Drawing](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#drawing)
4. [Handle Memory Warnings](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#memwarnings)
5. [Reuse Expensive Objects](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#reuseobjects)
6. [Use Sprite Sheets](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#spritesheets)
7. [Avoid Re-Processing Data](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#avoidreprocess)
8. [Choose the Right Data Format](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#chooserightformat)
9. [Set Background Images Appropriately](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#bgimages)
10. [Reduce Your Web Footprint](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#reduceweb)
11. [Set the Shadow Path](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#shadowpath)
12. [Optimize Your Table Views](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#tableviews)
13. [Choose Correct Data Storage Option](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#datastorage)

**Advanced**

These are tips you should use only when you’re positive they’ll fix the issue, and you feel comfortable using them.

1. [Speed up Launch Time](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#launchtime)
2. [Use Autorelease Pool](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#autoreleasepool)
3. [Cache Images – Or Not](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#cacheimages)
4. [Avoid Date Formatters Where Possible](https://www.raywenderlich.com/2752-25-ios-app-performance-tips-tricks#avoidformatters)

JSON is faster to parse, and is generally smaller than XML, which means less data to transfer.

Xcode includes a performance tuning application named Instruments that you can use to profile your application using all sorts of different metrics. They have tools to inspect CPU usage, memory usage, leaks, file/network activity, and energy usage just to name a few. It’s really easy to start profiling your app from Xcode, but it’s sometimes not as easy to understand what you see when it’s profiling, which deters some developers from being able to use this tool to it’s fullest potential.

**Bitbucket** is a web-based [version control](https://en.wikipedia.org/wiki/Version_control) [repository](https://en.wikipedia.org/wiki/Repository_(version_control)) [hosting service](https://en.wikipedia.org/wiki/Shared_web_hosting_service) owned by [Atlassian](https://en.wikipedia.org/wiki/Atlassian), for [source code](https://en.wikipedia.org/wiki/Source_code) and development projects that use either [Mercurial](https://en.wikipedia.org/wiki/Mercurial) (since launch till June 1, 2020[[2]](https://en.wikipedia.org/wiki/Bitbucket#cite_note-sunsetting-mercurial-2)) or [Git](https://en.wikipedia.org/wiki/Git_(software)) (since October 2011[[3]](https://en.wikipedia.org/wiki/Bitbucket#cite_note-3)) [revision control](https://en.wikipedia.org/wiki/Revision_control) systems. Bitbucket offers both commercial plans and free accounts. It offers free accounts with an unlimited number of private repositories (which can have up to five users in the case of free accounts) as of September 2010. Bitbucket integrates with other Atlassian software like [Jira](https://en.wikipedia.org/wiki/Jira_(software)), [HipChat](https://en.wikipedia.org/wiki/HipChat), [Confluence](https://en.wikipedia.org/wiki/Confluence_(software)) and [Bamboo](https://en.wikipedia.org/wiki/Bamboo_(software)).

It is similar to [GitHub](https://en.wikipedia.org/wiki/GitHub), which primarily uses [Git](https://en.wikipedia.org/wiki/Git_(software)). Bitbucket has traditionally marketed its services to professional developers with private [proprietary software](https://en.wikipedia.org/wiki/Proprietary_software) code, especially since being acquired by Atlassian in 2010.[[4]](https://en.wikipedia.org/wiki/Bitbucket#cite_note-4) In February 2017, Bitbucket announced it had reached 6 million developers and 1 million teams on its platform.[[5]](https://en.wikipedia.org/wiki/Bitbucket#cite_note-5) In April 2019 Atlassian announced that Bitbucket reached **10 million** registered users and over 28 million repositories.[[6]](https://en.wikipedia.org/wiki/Bitbucket#cite_note-6)

*“****What actually makes Bitbucket better than Github?”***

## ****1. Bitbucket is more flexible than Github****

## ****2. Bitbucket gives you unlimited private repositories****

## ****3. Bitbucket has better pricing for private work****

## ****4. Continuous Integration / Delivery is built-in by default****

## ****5. Bitbucket is trustworthy****

## ****6. Smarter semantic searching****

## ****7. Bitbucket has a powerful Jira integration****

## ****8. Bitbucket has a powerful Trello integration****

The purpose of [default synchronize]; is to make the user defaults get written on disk immediately. You don't need to call it explicitly, iOS already does it at appropriate moments. So you can remove that line. In fact, it's a performance problem if you call synchronize every time you set a default.

Prior to iOS 7, the user defaults were always synchronized when the application transitioned into background. As of iOS 7, that is no longer the case, so you might want to call synchronize in your app delegate's applicationDidEnterBackground: or register to the UIApplicationDidEnterBackgroundNotification notification to do that.

From [the documentation for -[NSUserDefaults synchronize]:](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/nsuserdefaults_Class/Reference/Reference.html#//apple_ref/occ/instm/NSUserDefaults/synchronize)

Because this method is automatically invoked at periodic intervals, use this method only if you cannot wait for the automatic synchronization (for example, if your application is about to exit) or if you want to update the user defaults to what is on disk even though you have not made any changes.

let myDictionary: [String: [Int]] = ["Hello": [1, 2, 3], "World": [4, 5, 6]]

In fact, you don't even need the explicit type declaration if you assign an initial value in place. It can go as simple as:

let myDictionary = ["Hello": [1, 2, 3], "World": [4, 5, 6]]

If you want to store for example an array of strings:

var dict: [String: [String]]

or without syntactic sugar:

var dict: Dictionary<String, Array<String>>

Dictionaries, like arrays and more generally whatever uses generics, can handle anything that is a swift type, including tuples, closures, dictionaries, dictionaries of dictionaries, arrays of dictionaries, etc. - unless conditions are specified for the generic type (for instance, a dictionary key can be any type that implements the Hashable protocol), and in that case types must conform to the constraints.

A type that provides an integer hash value.

You can use any type that conforms to the Hashable protocol in a set or as a dictionary key. Many types in the standard library conform to Hashable: strings, integers, floating-point and Boolean values, and even sets provide a hash value by default. Your own custom types can be hashable as well. When you define an enumeration without associated values, it gains Hashable conformance automatically, and you can add Hashable conformance to your other custom types by adding a single hashValue property.

A hash value, provided by a type's hashValue property, is an integer that is the same for any two instances that compare equally. That is, for two instances a and b of the same type, if a == b then a.hashValue == b.hashValue. The reverse is not true: Two instances with equal hash values are not necessarily equal to each other.

**Important:** Hash values are not guaranteed to be equal across different executions of your program. Do not save hash values to use during a future execution.

# Conforming to the Hashable Protocol

To use your own custom type in a set or as the key type of a dictionary, add Hashable conformance to your type by providing a hashValue property. The Hashable protocol inherits from the Equatable protocol, so you must also add an equal-to operator (==) function for your custom type.

As an example, consider a GridPoint type that describes a location in a grid of buttons. Here's the initial declaration of the GridPoint type:

* /// A point in an x-y coordinate system.
* struct GridPoint {
* var x: [Int](https://swiftdoc.org/v3.0/type/Int)
* var y: [Int](https://swiftdoc.org/v3.0/type/Int)
* }

You'd like to create a set of the grid points where a user has already tapped. Because the GridPoint type is not hashable yet, it can't be used as the Element type for a set. To add Hashable conformance, provide an == operator function and a hashValue property.

* extension GridPoint: [Hashable](https://swiftdoc.org/v3.0/protocol/Hashable) {
* var hashValue: [Int](https://swiftdoc.org/v3.0/type/Int) {
* return x.hashValue ^ y.hashValue
* }
* static func == (lhs: GridPoint, rhs: GridPoint) -> [Bool](https://swiftdoc.org/v3.0/type/Bool) {
* return lhs.x == rhs.x && lhs.y == rhs.y
* }
* }

The hashValue property in this example combines the hash values of a grid point's x and y values using the bitwise XOR operator (^). The ^ operator is one way to combine two integer values into a single value.

**Note:** Set and dictionary performance depends on hash values that minimize collisions for their associated element and key types, respectively.

Now that GridPoint conforms to the Hashable protocol, you can create a set of previously tapped grid points.

* var tappedPoints: [Set](https://swiftdoc.org/v3.0/type/Set) = [GridPoint(x: 2, y: 3), GridPoint(x: 4, y: 1)]
* let nextTap = GridPoint(x: 0, y: 1)
* if tappedPoints.contains(nextTap) {
* print("Already tapped at (\(nextTap.x), \(nextTap.y)).")
* } else {
* tappedPoints.insert(nextTap)
* print("New tap detected at (\(nextTap.x), \(nextTap.y)).")
* }
* // Prints "New tap detected at (0, 1).")

|  |  |
| --- | --- |
| Inheritance | [Equatable](https://swiftdoc.org/v3.0/protocol/Equatable)[**VIEW PROTOCOL HIERARCHY →**](https://swiftdoc.org/v3.0/protocol/hashable/hierarchy/) |
| Import | * import Swift |

### Instance Variables

[var hashValue: Int](https://swiftdoc.org/v3.0/protocol/hashable/#comment-var-hashvalue_-int) REQUIRED

### Instance Methods

[func ==(\_:rhs:)](https://swiftdoc.org/v3.0/protocol/hashable/#comment-func-eqeq_rhs_) REQUIRED

You can use any type that conforms to the Hashable protocol in a set or as a dictionary key. Many types in the standard library conform to Hashable: Strings, integers, floating-point and Boolean values, and even sets are hashable by default. Some other types, such as optionals, arrays and ranges automatically become hashable when their type arguments implement the same.

Your own custom types can be hashable as well. When you define an enumeration without associated values, it gains Hashable conformance automatically, and you can add Hashableconformance to your other custom types by implementing the hash(into:) method. For structs whose stored properties are all Hashable, and for enum types that have all-Hashableassociated values, the compiler is able to provide an implementation of hash(into:)automatically.

Hashing a value means feeding its essential components into a hash function, represented by the Hasher type. Essential components are those that contribute to the type’s implementation of Equatable. Two instances that are equal must feed the same values to Hasher in hash(into:), in the same order.

### Conforming to the Hashable Protocol

To use your own custom type in a set or as the key type of a dictionary, add Hashableconformance to your type. The Hashable protocol inherits from the Equatable protocol, so you must also satisfy that protocol’s requirements.

The compiler automatically synthesizes your custom type’s Hashable and requirements when you declare Hashable conformance in the type’s original declaration and your type meets these criteria:

* For a struct, all its stored properties must conform to Hashable.
* For an enum, all its associated values must conform to Hashable. (An enum without associated values has Hashable conformance even without the declaration.)

To customize your type’s Hashable conformance, to adopt Hashable in a type that doesn’t meet the criteria listed above, or to extend an existing type to conform to Hashable, implement the hash(into:) method in your custom type.

In your hash(into:) implementation, call combine(\_:) on the provided Hasher instance with the essential components of your type. To ensure that your type meets the semantic requirements of the Hashable and Equatable protocols, it’s a good idea to also customize your type’s Equatable conformance to match.

As an example, consider a GridPoint type that describes a location in a grid of buttons. Here’s the initial declaration of the GridPoint type:

/// A point in an x-y coordinate system. struct GridPoint { var x: Int var y: Int }

You’d like to create a set of the grid points where a user has already tapped. Because the GridPoint type is not hashable yet, it can’t be used in a set. To add Hashable conformance, provide an == operator function and implement the hash(into:) method.

extension GridPoint: Hashable { static func == (lhs: GridPoint, rhs: GridPoint) -> Bool { return lhs.x == rhs.x && lhs.y == rhs.y } func hash(into hasher: inout Hasher) { hasher.combine(x) hasher.combine(y) } }

The hash(into:) method in this example feeds the grid point’s x and y properties into the provided hasher. These properties are the same ones used to test for equality in the ==operator function.

Now that GridPoint conforms to the Hashable protocol, you can create a set of previously tapped grid points.

var tappedPoints: Set = [GridPoint(x: 2, y: 3), GridPoint(x: 4, y: 1)] let nextTap = GridPoint(x: 0, y: 1) if tappedPoints.contains(nextTap) { print("Already tapped at (\(nextTap.x), \(nextTap.y)).") } else { tappedPoints.insert(nextTap) print("New tap detected at (\(nextTap.x), \(nextTap.y)).") } // Prints "New tap detected at (0, 1).")

array in array

array in dictionary

dictionary in dictionary

dictionary in array

## What is a set?

Sets is simply a container that can hold multiple value of data type in an unordered list and ensures unique element in the container (i.e each data appears only once).

Unordered list means you won't get the elements in the same order as you defined the items in the Set.

The main advantage of using Sets over arrays is when you need to ensure that an item only appears once and when the order of items is not important.

Values stored in a set must be **hashable**. This means it has to provide a hashValueproperty. This is important because sets are unordered and it uses hashValue is used to access the elements of the sets.

All of Swift's basic types (such as String, Int, Double, and Bool) are hashable by default, and can be used as set value types. However, you can also create your Hashable Type in Swift that can be stored in a set.

## How to declare a set in Swift?

You can create an empty set by specifying the type as Set followed by the type of Data it can store within **< >**.

### Example 1: Declaring an empty set

1. let emptyIntSet:Set = []
2. print(emptyIntSet)

**OR**

1. let emptyIntSet:Set = Set()
2. print(emptyIntSet)

When you run the program, the output will be:

[ ]

In the above program, we have declared a constant emptyIntSet of type Set that can store multiple values of integer and initialized with 0 values.

Since, Swift is a type inference language, you can also create set directly without specifying the Data Type but must initialize with some values so that compiler can infer its type as:

### Example 2: Declaring a set with some values

1. let someIntSet:Set = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2. print(someIntSet)

When you run the program, the output will be:

[2, 4, 9, 5, 6, 7, 3, 1, 8]

In the above program, we have declared a constant someIntSet that can store sets of Integer without specifying the type explicitly. However, we need to write :Set when defining the variable, otherwise Swift will create an array for us.

Also, as arrays, we have initialized the set with **1, 2 ,3 ,4, 5, 6, 7, 8, 9** values using the [] brackets.

As you've learnt, when you try to print the values inside the set as print(someIntSet), you will get the results in a different order than you have defined the items in the set because it stores value with no defined ordering. Therefore, each time when you access the order changes.

### Example 3: Declaring a set with duplicate values

1. let someStrSet:Set = ["ab","bc","cd","de","ab"]
2. print(someStrSet)

When you run the program, the output will be:

["de", "ab", "cd", "bc"]

In the above program, we have defined a duplicate value **ab** in the set. And. when we try to access the value inside the set using print(someStrSet), the duplicate value is automatically removed from the set. Therefore, set guarantees unique elements/values inside it.

You can also declare a set with your own custom Hashable type in Swift. To learn more, visit Swift Hashable.

## How to access set elements in Swift?

You cannot access elements of a set using subscript syntax as arrays. This is because sets are unordered and do not have indices to access the elements.

So, you need to access the set using its methods and properties or using for-in loops.

#### Example 4: Accessing elements of a set

1. var someStrSet:Set = ["ab", "bc", "cd", "de"]
2. for val in someStrSet {
3. print(val)
4. }

When you run the program, the output will be:

de ab cd bc

In the above program, we get the val in different order than elements of a set because sets are unordered unlike arrays.

You can also access element of a set directly removing the value from the set as below:

#### Example 5: Accessing elements of a set using remove()

1. var someStrSet:Set = ["ab", "bc", "cd", "de"]
2. let someVal = someStrSet.remove("cd")
3. print(someVal)
4. print(someStrSet)

When you run the program, the output will be:

Optional("cd") ["de", "ab", "bc"]

In the above program, you can see the remove method returns an optional string. Therefore, it's recommended you to do optional handling as below. To learn more about optionals, visit [Swift Optionals](https://www.programiz.com/swift-programming/optionals).

#### Example 6: Optional handling for remove()

1. var someStrSet:Set = ["ab", "bc", "cd", "de"]
2. if let someVal = someStrSet.remove("cd") {
3. print(someVal)
4. print(someStrSet)
5. } else {
6. print("cannot find element to remove")
7. }

When you run the program, the output will be:

cd ["de", "ab", "bc"]

## How to add new element in a set?

You can add a new element to a set using insert() method in Swift.

### Example 7: Add new element using insert()

1. var someStrSet:Set = ["ab", "bc", "cd", "de"]
2. someStrSet.insert("ef")
3. print(someStrSet)

When you run the program, the output will be:

["ab", "de", "cd", "ef", "bc"]

In the above program, we used the set's insert() method to add a new element to a set. Since, sets are unordered, the position of the inserted element isn't known.

You use a set instead of an array when you need to test efficiently for membership and you aren’t concerned with the order of the elements in the collection, or when you need to ensure that each element appears only once in a collection.

You can create a set with any element type that conforms to the Hashable protocol. By default, most types in the standard library are hashable, including strings, numeric and Boolean types, enumeration cases without associated values, and even sets themselves.

Swift makes it as easy to create a new set as to create a new array. Simply assign an array literal to a variable or constant with the Set type specified.

let ingredients: Set = ["cocoa beans", "sugar", "cocoa butter", "salt"] if ingredients.contains("sugar") { print("No thanks, too sweet.") } // Prints "No thanks, too sweet."

### Set Operations

Sets provide a suite of mathematical set operations. For example, you can efficiently test a set for membership of an element or check its intersection with another set:

* Use the contains(\_:) method to test whether a set contains a specific element.
* Use the “equal to” operator (==) to test whether two sets contain the same elements.
* Use the isSubset(of:) method to test whether a set contains all the elements of another set or sequence.
* Use the isSuperset(of:) method to test whether all elements of a set are contained in another set or sequence.
* Use the isStrictSubset(of:) and isStrictSuperset(of:) methods to test whether a set is a subset or superset of, but not equal to, another set.
* Use the isDisjoint(with:) method to test whether a set has any elements in common with another set.

You can also combine, exclude, or subtract the elements of two sets:

* Use the union(\_:) method to create a new set with the elements of a set and another set or sequence.
* Use the intersection(\_:) method to create a new set with only the elements common to a set and another set or sequence.
* Use the symmetricDifference(\_:) method to create a new set with the elements that are in either a set or another set or sequence, but not in both.
* Use the subtracting(\_:) method to create a new set with the elements of a set that are not also in another set or sequence.

You can modify a set in place by using these methods’ mutating counterparts: formUnion(\_:), formIntersection(\_:), formSymmetricDifference(\_:), and subtract(\_:).

for var shoppingList in shoppingLists { let items = getGroceryItemsByShoppingList(shoppingList) shoppingList.groceryItems = getGroceryItemsByShoppingList(shoppingList) shoppingList.title = "BLAH" // copied by value print("ShoppingList \(shoppingList.title) has \(shoppingList.groceryItems.count) items") // THIS PRINT BLAH } print("shoppingLists[0].groceryItems.count \(shoppingLists[0].groceryItems.count)") // THIS PRINTS THE ORIGINAL CONTENT

# containsString:

Returns a Boolean value indicating whether the string contains a given string by performing a case-sensitive, locale-unaware search.

# localizedCaseInsensitiveContainsString:

Returns a Boolean value indicating whether the string contains a given string by performing a case-insensitive, locale-aware search.

let filteredArray = self.arrCountry.filter { $0.localizedCaseInsensitiveContains("india") }

**localizedCaseInsensitiveContains**  
Returns a Boolean value indicating whether the given string is non-empty and contained within this string by case-insensitive, non-literal search, taking into account the current locale. Locale-independent case-insensitive operation, and other needs, can be achieved by calling range(of:options:range:locale:).

Equivalent to: range(of: other, options: .caseInsensitiveSearch, locale: Locale.current) != nil

It's better to use

.filter { $0.range(of: "india", options: .caseInsensitive) != nil }

1) You should request access to the user's calendar via "requestAccessToEntityType:completion:" and execute the event handling inside of a block.

2) You need to commit your event now or pass the "commit" param to your save/remove call

Everything else stays the same...

Add the EventKit framework and #import <EventKit/EventKit.h> to your code.

In my example, I have a NSString \*savedEventId instance property.

To add an event:

EKEventStore \*store = [EKEventStore new]; [store requestAccessToEntityType:EKEntityTypeEvent completion:^(BOOL granted, NSError \*error) { if (!granted) { return; } EKEvent \*event = [EKEvent eventWithEventStore:store]; event.title = @"Event Title"; event.startDate = [NSDate date]; //today event.endDate = [event.startDate dateByAddingTimeInterval:60\*60]; //set 1 hour meeting event.calendar = [store defaultCalendarForNewEvents]; NSError \*err = nil; [store saveEvent:event span:EKSpanThisEvent commit:YES error:&err]; self.savedEventId = event.eventIdentifier; //save the event id if you want to access this later }];

Remove the event:

EKEventStore\* store = [EKEventStore new]; [store requestAccessToEntityType:EKEntityTypeEvent completion:^(BOOL granted, NSError \*error) { if (!granted) { return; } EKEvent\* eventToRemove = [store eventWithIdentifier:self.savedEventId]; if (eventToRemove) { NSError\* error = nil; [store removeEvent:eventToRemove span:EKSpanThisEvent commit:YES error:&error]; } }];

This adds events to your default calendar, if you have multiple calendars then you'll have find out which one that is

*Swift version*

You need to import the EventKit framework

import EventKit

Add event

let store = EKEventStore() store.requestAccessToEntityType(.Event) {(granted, error) in if !granted { return } var event = EKEvent(eventStore: store) event.title = "Event Title" event.startDate = NSDate() //today event.endDate = event.startDate.dateByAddingTimeInterval(60\*60) //1 hour long meeting event.calendar = store.defaultCalendarForNewEvents do { try store.saveEvent(event, span: .ThisEvent, commit: true) self.savedEventId = event.eventIdentifier //save event id to access this particular event later } catch { // Display error to user } }

Remove event

let store = EKEventStore() store.requestAccessToEntityType(EKEntityTypeEvent) {(granted, error) in if !granted { return } let eventToRemove = store.eventWithIdentifier(self.savedEventId) if eventToRemove != nil { do { try store.removeEvent(eventToRemove, span: .ThisEvent, commit: true) } catch { // Display error to user } } }

The EventKit framework provides access to calendar and reminders data so you and your users can create, retrieve, and edit calendar items in your app. In iOS, [EventKitUI](https://developer.apple.com/documentation/eventkitui?language=objc) provides user interfaces you can implement in your app to let users create and edit calendar items.

You can use EventKit to set up alarms and create recurring events. And if a change to the Calendar database occurs from outside your app, EventKit detects the change and sends a notification, allowing you to stay up to date.

[EKEventStore](https://developer.apple.com/documentation/eventkit/ekeventstore?language=objc)

An object that accesses the user’s calendar and reminder events and supports the scheduling of new events.

[EKEvent](https://developer.apple.com/documentation/eventkit/ekevent?language=objc)

A class that represents an event added to a calendar.

[EKReminder](https://developer.apple.com/documentation/eventkit/ekreminder?language=objc)

A class that represents a reminder added to a calendar.

[EKCalendar](https://developer.apple.com/documentation/eventkit/ekcalendar?language=objc)

A class that represents a calendar in Event Kit.

[EKParticipant](https://developer.apple.com/documentation/eventkit/ekparticipant?language=objc)

A class that represents a participant in a calendar event.

he EKEventStore class is an application’s point of contact for accessing calendar and reminder data.

In iOS, you must request access to an entity type after the event store is initialized with [requestAccessToEntityType:completion:](https://developer.apple.com/documentation/eventkit/ekeventstore/1507547-requestaccesstoentitytype?language=objc) for data to return.

**Important**

An iOS app linked on or after iOS 10.0 must include in its Info.plist file the usage description keys for the types of data it needs to access or it will crash. To access Reminders and Calendar data specifically, it must include [NSRemindersUsageDescription](https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW16)and [NSCalendarsUsageDescription](https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW15), respectively.

To access the user’s Calendar data, all sandboxed macOS apps must include the com.apple.security.personal-information.calendars entitlement. To learn more about entitlements related to App Sandbox, see [Enabling App Sandbox](https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html#//apple_ref/doc/uid/TP40011195-CH4-SW4).

Use the [eventWithEventStore:](https://developer.apple.com/documentation/eventkit/ekevent/1507483-eventwitheventstore?language=objc) method to create a new event. Use the properties in the class to get and modify certain information about an event. Other properties, such as the event’s title and calendar, are inherited from the parent class [EKCalendarItem](https://developer.apple.com/documentation/eventkit/ekcalendaritem?language=objc).

# EventKitUI

Display an interface for viewing, selecting, and editing calendar events and reminders.

On iOS, use the EventKitUI framework to show calendar and reminder information to the user modally. EventKitUI provides view controllers for viewing and editing calendar and reminder information, choosing which calendar to view, and for determining whether to present calendars as read-only or readable and writeable.

The view controllers you'll use on iOS are:

* [EKEventViewController](https://developer.apple.com/documentation/eventkitui/ekeventviewcontroller?language=objc), for displaying existing events.
* [EKEventEditViewController](https://developer.apple.com/documentation/eventkitui/ekeventeditviewcontroller?language=objc), for creating, editing, or deleting events.
* [EKCalendarChooser](https://developer.apple.com/documentation/eventkitui/ekcalendarchooser?language=objc), for selecting one or more calendars, and to determine whether a calendar has read-only or read-write access.

You present these interfaces from within your app. Upon presentation, the system manages all interactions with the user, notifying you when the interfaces are dismissed.

EventKitUI also provides several configurable classes for selecting a default calendar, displaying buttons, or to enabling the user to select one or more calendars.

**Note**

To access the event store, which contains calendar and reminder data, use EventKit. For more information, see *Accessing the Event Store*.

[EKEventViewController](https://developer.apple.com/documentation/eventkitui/ekeventviewcontroller?language=objc)

A view controller for displaying existing calendar and reminder events, and for optionally editing those events.

[EKEventEditViewController](https://developer.apple.com/documentation/eventkitui/ekeventeditviewcontroller?language=objc)

A view controller for creating, editing, and deleting calendar events.

[EKCalendarChooser](https://developer.apple.com/documentation/eventkitui/ekcalendarchooser?language=objc)

A view controller for determining whether a user may select one or more calendars.

[EventKitUIBundle](https://developer.apple.com/documentation/eventkitui/2866511-eventkituibundle?language=objc)

Use to access resources within the app bundle.

UIKit has one main class for sharing content from our apps, and it has a flexible design that lets other services connect to it. This class, UIActivityViewController, can handle sharing text, URLs, image, and more, and is also able to add our app’s own services alongside the others.

import UIKit class ViewController: UIViewController { // share text @IBAction func shareTextButton(\_ sender: UIButton) { // text to share let text = "This is some text that I want to share." // set up activity view controller let textToShare = [ text ] let activityViewController = UIActivityViewController(activityItems: textToShare, applicationActivities: nil) activityViewController.popoverPresentationController?.sourceView = self.view // so that iPads won't crash // exclude some activity types from the list (optional) activityViewController.excludedActivityTypes = [ UIActivityType.airDrop, UIActivityType.postToFacebook ] // present the view controller self.present(activityViewController, animated: true, completion: nil) } // share image @IBAction func shareImageButton(\_ sender: UIButton) { // image to share let image = UIImage(named: "Image") // set up activity view controller let imageToShare = [ image! ] let activityViewController = UIActivityViewController(activityItems: imageToShare, applicationActivities: nil) activityViewController.popoverPresentationController?.sourceView = self.view // so that iPads won't crash // exclude some activity types from the list (optional) activityViewController.excludedActivityTypes = [ UIActivityType.airDrop, UIActivityType.postToFacebook ] // present the view controller self.present(activityViewController, animated: true, completion: nil) } }

<https://stackoverflow.com/questions/35931946/basic-example-for-sharing-text-or-image-with-uiactivityviewcontroller-in-swift>

The system provides several standard services, such as copying items to the pasteboard, posting content to social media sites, sending items via email or SMS, and more. Apps can also define custom services.

Your app is responsible for configuring, presenting, and dismissing this view controller. Configuration for the view controller involves specifying the data objects on which the view controller should act. (You can also specify the list of custom services your app supports.) When presenting the view controller, you must do so using the appropriate means for the current device. On iPad, you must present the view controller in a popover. On iPhone and iPod touch, you must present it modally.

Framework

# Messages

Create app extensions that allow users to send text, stickers, media files, and interactive messages.

You can use the Messages framework to create two types of app extensions: Sticker packs and iMessage apps. Both extension types can be created as standalone apps or as app extensions within a containing iOS app.

### Sticker Packs

Sticker packs simply provide static sets of stickers, images that users can send inline as messages or peel off and attach to message bubbles in the transcript. Sticker packs do not require any code. You add stickers by dragging image files into the Sticker Pack folder inside the Stickers asset catalog. Valid image files must meet the following specifications:

* The image must be a PNG, APNG, GIF, or JPEG file.
* The file must be less than 500 KB.
* For the best results, the image should not be smaller than 100 x 100 points or larger than 206 x 206 points.

**Note**

Always provide @3x images (300 x 300 pixels to 618 x 618 pixels). The system generates the @2x and @1x versions by downscaling the @3x images at runtime.

Messages supports three sticker sizes, which are displayed in a grid-based browser. In the Xcode Attributes inspector, pick one of the following sizes for your sticker pack:

* **Small.** 100 x 100 points @3x (300 x 300 pixels).
* **Medium.** 136 x 136 points @3x (408 x 408 pixels).
* **Large.** 206 x 206 points @3x (618 x 618 pixels).

For optimal quality and performance, provide sticker images of the size you choose.

To learn more about creating animated stickers, see [Creating Stickers with Motion](https://developer.apple.com/support/stickers/motion/).

To learn more about submitting sticker packs to the App Store, see [Preparing Sticker Submissions](https://developer.apple.com/app-store/sticker-submissions/).

### iMessage Apps

iMessage apps leverage the full framework to interact with the Messages app.

**Note**

An iMessage app linked on or after iOS 10.0 must include usage description keys for the device features it needs to access in its Info.plist file, or it will crash. Specifically, it must include [NSCameraUsageDescription](https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/plist/info/NSCameraUsageDescription) to access the device’s camera, and it must include [NSMicrophoneUsageDescription](https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/plist/info/NSMicorphoneUsageDescription) to access the device’s microphones.

Use iMessage apps to:

* Present a custom user interface inside the Messages app; see [MSMessagesAppViewController](https://developer.apple.com/documentation/messages/msmessagesappviewcontroller?language=objc).
* Create a custom or dynamic sticker browser; see [MSStickerBrowserViewController](https://developer.apple.com/documentation/messages/msstickerbrowserviewcontroller?language=objc).
* Insert text, stickers, or media files into the Messages app’s input field; see [MSConversation](https://developer.apple.com/documentation/messages/msconversation?language=objc).
* Create interactive messages that carry app-specific data; see [MSMessage](https://developer.apple.com/documentation/messages/msmessage?language=objc).
* Update interactive messages (for example, to create games or collaborative apps); see [MSSession](https://developer.apple.com/documentation/messages/mssession?language=objc).

Framework

# MessageUI

Create a user interface for composing email and text messages, so users can edit and send messages without leaving your app.

## Overview

The Message UI framework provides specialized view controllers for presenting standard composition interfaces for email and SMS (Short Messaging Service) text messages. Use these interfaces to add message delivery capabilities, without requiring the user to leave your app.

To display a composition interface, present the corresponding view controller modally from your app. Once presented, the user has the option to customize the contents before sending or canceling the message. Your custom delegate object then handles the dismissal of the view controller based on the user’s action. For information on how to present and dismiss view controllers, see [View Controller Programming Guide for iOS](https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple_ref/doc/uid/TP40007457).

**Important**

The view controllers in this framework provide methods for determining if you can send a given message type on the current iOS device. If you cannot send a message, do not display the corresponding view controller.

Class

# MFMailComposeViewController

A standard view controller, whose interface lets the user manage, edit, and send email messages.

# MFMessageComposeViewController

A standard view controller whose interface lets the user compose and send SMS or MMS messages.

## Overview

Use an [MFMessageComposeViewController](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller?language=objc) object to display the standard message composition interface inside your app. Before presenting the interface, populate the fields with the set of initial recipients and the message you want to send. After presenting the interface, the user can edit your initial values before sending the message.

The composition interface does not guarantee the delivery of your message; it only lets you construct the initial message and present it for user approval. The user may opt to cancel the composition interface, in which case the message and its contents are discarded. If the user opts to send the message, the Messages app takes on the responsibility of sending the message.

**Important**

You must not modify the view hierarchy presented by this view controller. However, you can customize the appearance of the interface using the [UIAppearance](https://developer.apple.com/documentation/uikit/uiappearance?language=objc) protocol.

An alternate way to compose SMS messages is to create and open a URL that uses the smsscheme. URLs of that type are directed to the Messages app, which uses your URL to configure the message. For information about the structure of sms URLs, see [Apple URL Scheme Reference](https://developer.apple.com/library/archive/featuredarticles/iPhoneURLScheme_Reference/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007899).

### Checking the Availability of the Composition Interface

Before presenting the message compose view controller, always call the the [canSendText](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller/1614072-cansendtext?language=objc)method to see if the current device is configured to send messages. If the user’s device is not set up to send or receive messages, you can notify the user or simply disable the messaging features in your application. You should not attempt to use this interface if the [canSendText](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller/1614072-cansendtext?language=objc)method returns NO. If messaging is available, you can also use the [canSendAttachments](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller/1614076-cansendattachments?language=objc) and [canSendSubject](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller/1614065-cansendsubject?language=objc) methods to determine if those specific messaging features are available.

**Listing 1**

Checking the availability of message services

if (![MFMessageComposeViewController canSendText]) { NSLog(@"Message services are not available."); }

### Configuring and Displaying the Composition Interface

After verifying that message services are available, you can create and configure the message composition view controller and then present it like you would any other view controller. Use the methods of this class to specify the message’s recipients and the contents of the message. If attachments or a subject line are supported, you can set values for them as well. Listing 2shows how to configure the composition interface and present it modally. Always assign a delegate to the [messageComposeDelegate](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontroller/1614070-messagecomposedelegate?language=objc) property, because the delegate is responsible for dismissing the composition interface later. The delegate object must conform to the [MFMessageComposeViewControllerDelegate](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontrollerdelegate?language=objc) protocol.

**Listing 2**

Configuring and presenting the composition interface

MFMessageComposeViewController\* composeVC = [[MFMessageComposeViewController alloc] init]; composeVC.messageComposeDelegate = self; // Configure the fields of the interface. composeVC.recipients = @[@"14085551212"]; composeVC.body = @"Hello from California!"; // Present the view controller modally. [self presentViewController:composeVC animated:YES completion:nil];

The message compose view controller is not dismissed automatically. When the user taps the buttons to send the message or cancel the interface, the message compose view controller calls the [messageComposeViewController:didFinishWithResult:](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontrollerdelegate/1614061-messagecomposeviewcontroller?language=objc) method of its delegate. Your implementation of that method must dismiss the view controller explicitly, as shown in Listing 3. You can also use this method to check the result of the operation.

**Listing 3**

Dismissing the message compose view controller

- (void)messageComposeViewController:(MFMessageComposeViewController \*)controller didFinishWithResult:(MessageComposeResult)result { // Check the result or perform other tasks. // Dismiss the message compose view controller. [self dismissViewControllerAnimated:YES completion:nil];}

For more information on how to present and dismiss view controllers, see [View Controller Programming Guide for iOS](https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple_ref/doc/uid/TP40007457).

### Detecting Changes to the Availability of Messaging

To be notified of changes to the messaging capabilities of the current device, add an observer to the [MFMessageComposeViewControllerTextMessageAvailabilityDidChangeNotification](https://developer.apple.com/documentation/messageui/mfmessagecomposeviewcontrollertextmessageavailabilitydidchangenotification?language=objc) notification. The system delivers that notification to your observer when the status of messaging changes.

\

Framework

# Social

Post content to supported social networking services, using standard system interfaces.

## Overview

On iOS and macOS, this framework provides a template for creating HTTP requests. On iOS only, the Social framework provides a generalized interface for posting requests on behalf of the user.

A common way to use this framework is:

* Create a network session.
* Get the activity feed for a user.
* Make a new post.
* Set properties on a post, add attachments, etc.
* Publish a post to an activity feed.

# SLComposeServiceViewController

A view controller that you present from your share app extension, allowing the user to compose social media posts.

Class

# SLComposeViewController

A view controller that allows the user to compose social media posts

A .PSD file is a layered image file used in Adobe PhotoShop. PSD, which stands for Photoshop Document, is the default format that Photoshop uses for saving data.  PSD is a proprietary file that allows the user to work with the images’ individual layers even after the file has been saved.

When an image is complete, Photoshop allows the user to flatten the layers and convert the flat image into a [.JPG](https://whatis.techtarget.com/definition/JPEG-Joint-Photographic-Experts-Group), [.GIF](https://searchwindevelopment.techtarget.com/definition/GIF), [.TIFF](https://whatis.techtarget.com/definition/TIFF-Tag-Image-File-Format) or other non-proprietary file format so it can be shared.  Once a PSD image has been flatten by conversion, however, it cannot be converted back to PSD and the user can no longer work with the image's layers. It is important, therefore, so always save the .PSD file and not overwrite it during conversion.

A PSD [file](https://www.lifewire.com/what-is-a-file-2625878) is used mainly in [Adobe Photoshop as the default format for saving data](https://www.lifewire.com/save-photoshop-file-for-older-version-1702241). Files with this [file extension](https://www.lifewire.com/what-is-a-file-extension-2625879) are called Adobe Photoshop Document files, and are in a proprietary format developed by Adobe.

Although some PSD files contain just one single image and nothing else, the common use for a PSD file includes much more than just storing an image file. They support multiple pictures, objects, filters, text, and more, as well as using layers, vector paths and shapes, and transparency.

For example, say you have five images within one PSD file, each on its own separate layer. Together, the pictures look as if they're on a single, flat image, but in reality, they're moveable and fully editable within their own layers — just as if you were working with separate pictures. You could reopen this PSD file as many times as you want and make changes to single layers without affecting the others.

iCloud is built into every Apple device. That means all your stuff — photos, files, notes and more — is safe, up to date and available wherever you are. And it works automatically, so all you have to do is keep doing what you love. Everyone gets 5GB of free iCloud storage to start and it’s easy to add more at any time.

**iCloud** is a [cloud storage](https://en.wikipedia.org/wiki/Cloud_storage) and [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) service[[1]](https://en.wikipedia.org/wiki/ICloud#cite_note-cloudconundrum-1)[[2]](https://en.wikipedia.org/wiki/ICloud#cite_note-troublecloud-2)[[3]](https://en.wikipedia.org/wiki/ICloud#cite_note-fromto-3) from [Apple Inc.](https://en.wikipedia.org/wiki/Apple_Inc.) launched on October 12, 2011. As of 2018, the service had an estimated 850 million users, up from 782 million users in 2016.[[4]](https://en.wikipedia.org/wiki/ICloud#cite_note-4)[[5]](https://en.wikipedia.org/wiki/ICloud#cite_note-5)[[6]](https://en.wikipedia.org/wiki/ICloud#cite_note-6)

iCloud enables users to store data such as documents, photos, and music on remote [servers](https://en.wikipedia.org/wiki/Server_(computing)) for download to [iOS](https://en.wikipedia.org/wiki/IOS), [macOS](https://en.wikipedia.org/wiki/MacOS) or [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) devices, to share and send data to other users, and to manage their Apple devices if lost or stolen.

**Cloud storage** is a model of [computer data storage](https://en.wikipedia.org/wiki/Computer_data_storage) in which the [digital data](https://en.wikipedia.org/wiki/Digital_data) is stored in logical [pools](https://en.wikipedia.org/wiki/Pool_(computer_science)). The [physical storage](https://en.wikipedia.org/wiki/Storage_virtualization) spans multiple [servers](https://en.wikipedia.org/wiki/Server_(computing)) (sometimes in multiple locations), and the physical environment is typically owned and managed by a [hosting](https://en.wikipedia.org/wiki/Internet_hosting_service) company. These cloud storage providers are responsible for keeping the data [available](https://en.wikipedia.org/wiki/Availability) and [accessible](https://en.wikipedia.org/wiki/Data_access), and the physical environment protected and running. People and organizations buy or lease storage capacity from the providers to store user, organization, or application data.

Cloud storage services may be accessed through a [colocated](https://en.wikipedia.org/wiki/Colocation_centre) [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) service, a [web service](https://en.wikipedia.org/wiki/Web_service) [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface) (API) or by applications that utilize the API, such as [cloud desktop](https://en.wikipedia.org/wiki/Cloud_desktop) storage, a [cloud storage gateway](https://en.wikipedia.org/wiki/Cloud_storage_gateway) or [Web](https://en.wikipedia.org/wiki/World_Wide_Web)-based [content management systems](https://en.wikipedia.org/wiki/Content_management_system).

CloudKit is Apple’s remote data storage service based on iCloud. It provides a low-cost option to store and share app data using your users’ iCloud accounts as a back-end storage service.

CloudKit is a good option for iOS-only applications that use a lot of data, but don’t require a great deal of server-side logic. In addition, CloudKit can be used for web and server applications.

**Note:** To work with the sample app in this CloudKit tutorial you’ll need an active iOS developer account. Without one, you won’t be able to enable the iCloud entitlements or access the CloudKit dashboard.

Simply registering for the iOS Developer Program makes you eligible to use CloudKit. You don’t have to register for additional services or create new accounts. When you enable CloudKit capabilities in your app all necessary server setup magic happens automatically.

Basically, iCloud is a service that helps user synchronize the data across devices. The main purpose is to let users easily put their data, whether it’s a file or document, so that they can access the data on any of their iOS devices. While you can use other cloud services for saving file or data, the core idea behind iCloud is to eliminate explicit synchronization between devices. Apple do not want users to think of the cloud servers and the syncing. Everything simply works seamlessly.

Store structured app and user data in iCloud containers that can be shared by all users of your app.

The CloudKit framework provides interfaces for moving data between your app and your iCloud containers. You use CloudKit to take your app’s existing data and store it in the cloud so that the user can access it on multiple devices. You can also store data in a public area where all users can access it.

CloudKit is Apple’s dedicated back-end service for your apps, allowing you to store user data and assets remotely entirely using Swift. It works similar to Core Data, although it’s much simpler in practice – you can save any kind of Swift data you like, and CloudKit takes care of the rest.

The framework provides authentication, a private database, a public database and structured asset storage services allowing developers to focus on client-side development.[[2]](https://en.wikipedia.org/wiki/CloudKit#cite_note-:0-2) It is the foundation for both iCloud Storage and iCloud Photo Library.[[3]](https://en.wikipedia.org/wiki/CloudKit#cite_note-3)

CloudKit also offers several APIs to access the iCloud Storage, where a user can store data and files so that they can be easily accessible from other devices.[[4]](https://en.wikipedia.org/wiki/CloudKit#cite_note-4)

[Firebase says](https://firebase.google.com/products/realtime-database/):

*The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in realtime.*

[CloudKit promises to](https://developer.apple.com/icloud/cloudkit/):

*Store your app’s data in iCloud and keep everything up to date across apps and on the web with CloudKit JS*

CloudKit offers tons of APIs to access iCloud. You can create a user model inside your application linked to a user’s iCloud account. Meanwhile, you can have a public global database to store application-level data. You can also save large files and bulk data into iCloud Drive, so your users can use their data from their other devices. This works just like working on local files, but with all the operations sent to the cloud.

Overall, CloudKit is a framework that replaces back-end web services like old-school databases, file storage, and user authentication systems. With CloudKit’s help you don’t need to worry about any of these, so you can focus your energy on your application.

# iCloud storage plans and pricing

When you sign up for iCloud, you automatically get 5GB of free storage. If you need more space in iCloud, you can upgrade to a larger storage plan.

You can [upgrade your iCloud storage plan](https://support.apple.com/kb/HT201318) from your iPhone, iPad, iPod touch, Mac, or PC. Choose from a total of 50GB, 200GB, or 2TB. You can even share the 200GB and 2TB plans [with your family](https://support.apple.com/kb/HT201060).

India3 (INR)  
**50GB**: Rs 75  
**200GB**: Rs 219  
**2TB**: Rs 749

* Free: 5GB of storage
* $0.99/month: 50GB of storage
* $2.99/month: 200GB of storage
* $9.99/month: 2TB of storage

Using iCloud with a SQLite database is possible only if your app uses Core Data to manage that database. Accessing live database files in iCloud using the SQLite interfaces is not supported and will likely corrupt your database

The short answer is no, you would need to use Core Data as you suspected. Apple has stated that sqlite is unsupported.

Edit: Check out the section on [iCloud that's now in the iOS Application Programming Guide](http://developer.apple.com/library/ios/#DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/iCloud/iCloud.html) under Using iCloud in Conjunction with Databases

Using iCloud with a SQLite database is possible only if your app uses Core Data to manage that database. Accessing live database files in iCloud using the SQLite interfaces is not supported and will likely corrupt your database. However, you can create a Core Data store based on SQLite as long as you follow a few extra steps when setting up your Core Data structures. You can also continue to use other types of Core Data stores—that is, stores not based on SQLite—without any special modifications.

Use Core Data with CloudKit to give users seamless access to the data in your app across all their devices.

On its own, Core Data provides powerful object graph management features for developing an app with structured data. CloudKit lets users access their data across every device on their iCloud account, while serving as an always-available backup service. Core Data with CloudKit combines the benefits of local persistence with cloud backup and distribution.

An advance Emoticons & GIF keyboard.

<https://github.com/kevalpatel2106/EmoticonGIFKeyboard>

**EmoticonGIFKeyboard** is an easy to integrate, customizable and lightweight library to add support for emojis and GIFs.

<https://github.com/isaced/ISEmojiView>

Emoji Keyboard for iOS

<https://github.com/SamuelZhaoY/ios-emojikeyboard>

Emoji Keyboard for iOS (support sticker as well)

typedef enum {

UIKeyboardTypeDefault, // Default type for the current input method.

UIKeyboardTypeASCIICapable, // Displays a keyboard which can enter ASCII characters, non-ASCII keyboards remain active UIKeyboardTypeNumbersAndPunctuation, // Numbers and assorted punctuation.

UIKeyboardTypeURL, // A type optimized for URL entry (shows . / .com prominently).

UIKeyboardTypeNumberPad, // A number pad (0-9). Suitable for PIN entry.

UIKeyboardTypePhonePad, // A phone pad (1-9, \*, 0, #, with letters under the numbers).

UIKeyboardTypeNamePhonePad, // A type optimized for entering a person's name or phone number. UIKeyboardTypeEmailAddress, // A type optimized for multiple email address entry (shows space @ . prominently). UIKeyboardTypeDecimalPad, // A number pad including a decimal point

UIKeyboardTypeTwitter, // Optimized for entering Twitter messages (shows # and @)

UIKeyboardTypeWebSearch, // Optimized for URL and search term entry (shows space and .)

UIKeyboardTypeAlphabet = UIKeyboardTypeASCIICapable, // Deprecated } UIKeyboardType;

**Giphy** (styled **GIPHY**) is an online database and search engine that allows users to search for and share short looping videos with no sound, that resemble animated [GIF](https://en.m.wikipedia.org/wiki/GIF) files.

Giphy is a site that contains pretty much any gif you could ever need. Another thing that makes Giphy great is their API. It’s pretty simple to learn and implement and can be used any number of ways.

By integrating with GIPHY, the first and largest GIF search engine, you gain free access to our ever-growing content library of GIFs and Stickers, plus brand new features like animated Emoji and Text — featuring the latest in entertainment, sports, and breaking news from GIPHY's official content partners.

The fastest and easiest way to bring the full GIPHY experience directly to your app is with GIPHY SDK. Built with developers and product designers in mind, GIPHY SDK is a top-to-bottom solution for all things GIF in your app. This includes interfacing with GIPHY API, fetching and caching assets, and displaying GIFs and Stickers on screen in customizable UI templates.

With billions of requests a day, it’s safe to say GIPHY knows GIFs. We're excited to share our best-in-class tools with you so your users can have the best GIF experience possible, with all the same features they're already enjoying on Facebook, Slack, Instagram, and more – with just a few lines of code.

Once the app is created, you will be assigned your own API Key. It will be a 32 character code that contains both letters and numbers.

That's where [Apple's Health app](https://www.cnet.com/news/the-complete-guide-to-apples-health-app/) comes in. It serves as a picture of your health, with dashboards for activity, sleep, nutrition, mindfulness and more. The more devices you connect to Apple Health, the better that picture gets.

The Health app makes it easy to learn about your health and start reaching your goals. It consolidates health data from iPhone, Apple Watch and third-party apps you already use, so you can view all your progress in one convenient place. And it recommends other helpful apps to round out your collection — making it simpler than ever to move your health forward.

The Health app highlights four categories: Activity, Sleep, Mindfulness and Nutrition. Each plays an important role in your overall health — and in the app. Health suggests great apps from each category to get you going, and the Today view shows all your stats at a glance to help you stay on track. Once you start using the Health app, there’s no stopping you.

Once you’ve submitted your app for review, you can view its status in the My Apps section of App Store Connect or on the App Store Connect App for iPhone and iPad. Review times may vary by app. On average, 50% of apps are reviewed in 24 hours and over 90% are reviewed in 48 hours. If your submission is incomplete, review times may be further delayed or your app may be rejected. Once your app has been reviewed, its status will be updated and you will be notified.

## Avoiding Common App Rejections

We’ve highlighted some of the most common issues that cause apps to get rejected to help you better prepare your apps before submitting them for review.

#### Crashes and Bugs

You should submit your app for review only when it is complete and ready to be published. Make sure to thoroughly test your app on devices running the latest software and fix all bugs before submitting.

#### Broken Links

All links in your app must be functional. A link to user support with up-to-date contact information and a link to your privacy policy is required for all apps.

#### Placeholder Content

Finalize all images and text in your app before sending it in for review. Apps that are still in progress and contain placeholder content are not ready to be distributed and cannot be approved.

#### Requesting Permission

When requesting permission to access user or usage data, you should clearly and completely describe how your app will use the data. Including an example can help users understand why your app is requesting access to their personal information.

If your app’s code references one or more APIs that access sensitive user data, the app’s Info.plist file should contain a $!{infoPlistKey} key with a user-facing purpose string explaining clearly and completely why your app needs the data. Starting spring 2019, all apps submitted to the App Store that access user data will be required to include a purpose string.

[Learn about requesting permission](https://developer.apple.com/documentation/uikit/core_app/protecting_the_user_s_privacy)

#### Inaccurate Screenshots

App Store screenshots should accurately communicate your app’s value and functionality. Use text and overlay images to highlight your app’s user experience, not obscure it. Make sure app UI and product images match the corresponding device type in App Store Connect. This helps users understand your app and makes for a positive App Store experience.

#### Incomplete Information

Enter all of the details needed to review your app in the App Review Information section of App Store Connect. If some features require signing in, provide a valid demo account username and password. If there are special configurations to set, include the specifics. If features require an environment that is hard to replicate or require specific hardware, be prepared to provide a demo video or the hardware. Also, please make sure your contact information is complete and up-to-date.

#### Substandard user Interface

Apple places a high value on clean, refined, and user-friendly interfaces. Make sure your UI meets these requirements by planning your design carefully and following our design guides and UI Design Do’s and Don’ts.

#### Web clippings, content aggregators, or a collection of links

Your app should be engaging and useful, and make the most of the features unique to iOS. Websites served in an iOS app, web content that is not formatted for iOS, and limited web interactions do not make a quality app.

#### Repeated submission of similar apps

Submitting several apps that are essentially the same ties up the App Review process and risks the rejection of your apps. Improve your review experience — and the experience of your future users — by thoughtfully combining your apps into one.

#### Misleading Users

Your app must perform as advertised and should not give users the impression the app is something it is not. If your app appears to promise certain features and functionalities, it needs to deliver.

#### Not enough lasting value

If your app doesn’t offer much functionality or content, or only applies to a small niche market, it may not be approved. Before creating your app, take a look at the apps in your category on the App Store and consider how you can provide an even better user experience.

Test code

ViewController \*vc1 = [[ViewController alloc]init];

Vc1.arrayProp = vc.arrayDataPass;

Push vc1

[arraProp removeElementAtIndex:0];

### 1 - copy always creates an immutable copy means you can not modify the object. 2 - mutableCopy always creates a mutable copy means you can modify the object

Here onwards I try to explain copy and mutable copy with some examples:

### copy:

NSArray \*obj = [NSArray arrayWithObjects:@"1",@"2", nil];

NSArray \*copyObj = [obj copy];

NSArray \*mutableObj = [obj mutableCopy];

As the copyObj is assigned to of type NSArray so it will not reflect any method from NSMUtableArray class. So you will end up with compile time error.

[copyO bj removeObjectAtIndex:0] //error: No visible @interface for 'NSArray' declares the selector 'removeObjectAtIndex:'

As the mutableObj is assigned to of type NSArray so it will not reflect any method from NSMUtableArray class. So you will end up with compile time error.

[mutableObj removeObjectAtIndex:1] //error: No visible @interface for 'NSArray' declares the selector 'removeObjectAtIndex:'

### mutableCopy:

NSMutableArray \*obj = [NSMutableArray arrayWithObjects:@"1",@"2", nil];

NSMutableArray \*copyObjy = [obj copy];

NSMutableArray \*mutableObjy = [obj mutableCopy];

Now [copyObjy removeObjectAtIndex:0]; **will make crash as you are trying to modify immutable object (**\_\_NSArrayI**).**

Whereas [mutableObjy removeObjectAtIndex:1]; **will work as it return the mutable object. remove the object from mutable copy will not modify source object.**

NSArray \*arrayB = [NSArray arrayWithObject:@"a"]; // Can't modify this NOW

NSArray \*arrayA = [arrayB copy]; // Create another copy of Array B like XEROX. If you change in XEROX it does not reflect to original Copy NSArray \*arrayC = [arrayB mutableCopy]; // This is just Mutable copy so you can make changes in this and you have to change NSArray to NSMutableArray here as it the mutable copy.

NSArray \*arrayD = arrayB;// Any change made in arrayD reflect to ArrayB but ArrayD is NSArray (immutable) so you can not make any changes in ArrayD

# What is a protocol associated type?

Associated types are a powerful way of making protocols generic.

## Associated Types

When defining a protocol, it’s sometimes useful to declare one or more associated types as part of the protocol’s definition. An associated type gives a placeholder name to a type that is used as part of the protocol. The actual type to use for that associated type isn’t specified until the protocol is adopted. Associated types are specified with the associatedtype keyword.

<https://medium.com/@bobgodwinx/swift-associated-type-design-patterns-6c56c5b0a73a>

 Classes are reference type whereas Structures and Enumerations are of a value type in swift. What does that mean is that a class object shares a single instance of the object and passes the same reference if passed to any function or new object whereas the value type is the one which creates a copy of it and passes only the value.

As we all know, Classes are reference type whereas Structures and Enumerations are of a value type in swift. What does that mean is that a class object shares a single instance of the object and passes the same reference if passed to any function or new object whereas the value type is the one which creates a copy of it and passes only the value.

If we try to change any variable inside a class it’s straight forward.

class Employee {  
 var name : String  
 var teamName : Stringinit(name: String, teamName: String) {  
 self.name = name  
 self.teamName = teamName  
 }func changeTeam(newTeamName : String){  
 self.teamName = newTeamName  
 }  
}var emp1 = Employee(name : "Suneet", teamName:"Engineering")print(emp1.teamName) //Engineering  
emp1.changeTeam(newTeamName : "Product")  
print(emp1.teamName) //Product

https://miro.medium.com/freeze/max/60/1*ZtHLGKVPP8mf9XSdDqRc3Q.gif?q=20



Whereas if you try to do the same in any value type, it will show us a compilation error,

struct Employee {  
 var name : String  
 var teamName : Stringinit(name: String, teamName: String) {  
 self.name = name  
 self.teamName = teamName  
 }func changeTeam(newTeamName : String){  
 self.teamName = newTeamName  
 //cannot assign to property: 'self' is immutable }  
}

It will show us the below error

cannot assign to property: 'self' is immutable

As Structures are of value type we cannot modify the properties directly.

The compiler also helps us with the possible solution

note: mark method 'mutating' to make 'self' mutable

It clearly states that adding mutating keyword to any function in value type can enable them to modify the variable. Internally when we try to mutate the value type, it does not mutate its value but it mutates the variable holding that value.

struct Employee {  
 var name : String  
 var teamName : Stringinit(name: String, teamName: String) {  
 self.name = name  
 self.teamName = teamName  
 }**mutating** func changeTeam(newTeamName : String){  
 self.teamName = newTeamName  
 }  
   
}var emp1 = Employee(name : "Suneet", teamName:"Engineering")print(emp1.teamName) //Engineering  
emp1.changeTeam(newTeamName : "Product")  
print(emp1.teamName) //Product

https://miro.medium.com/freeze/max/60/1*Vg4DzfvF8AHMnArFqMiIKQ.gif?q=20



Not only enum or struct but there are other data types also which are of value type.

* enum
* struct
* Int
* Double
* String
* Array
* Dictionary
* Set
* Tuple

Whereas the below ones of reference type

* Functions
* Closures
* Class

In swift, classes are **reference type** whereas structures and enumerations are **value types**. The properties of value types cannot be modified within its instance methods by default. In order to modify the properties of a value type, you have to use the **mutating keyword** in the instance method. With this keyword, your method can then have the ability to mutate the values of the properties and write it back to the original structure when the method implementation ends.

Below is a simple implementation of Stack in Swift that illustrates the use of mutating functions.

|  |
| --- |
| struct Stack { |
|  | public private(set) var items = [Int]() // Empty items array |
|  |  |
|  | mutating func push(\_ item: Int) { |
|  | items.append(item) |
|  | } |
|  |  |
|  | mutating func pop() -> Int? { |
|  | if !items.isEmpty { |
|  | return items.removeLast() |
|  | } |
|  | return nil |
|  | } |
|  | } |
|  |  |
|  | var stack = Stack() |
|  | stack.push(4) |
|  | stack.push(78) |
|  | stack.items // [4, 78] |
|  | stack.pop() |
|  | stack.items // [4] |

I called Apple for a separate account question I had so I asked about this at the same time. I asked Brian (*Senior Advisor from Apple Developer Program Support*) to email it to me so it's *official*. There are no public facing documents for this and because of that Brian actually submitted this information to Apple as feedback so in the future it is something we would be able to reference online.

**Officially from Apple Developer Program**

**Conversation and official email**

A maximum of 3 app store and Ad Hoc production certificates can be active at the same time for the Apple Developer program. For Enterprise it’s 2 certificates.

If you want to verify the limits for certificates the best option would be to visit the Certificates, Identifies and Profiles Portal and create additional certificates. Once the option is grayed out, that will indicate that you have reached the limit of certificates that can be active at one time.

Two Enterprise Distribution certificates can be created at a time and a single Enterprise Distribution certificate can apply to several apps.

Each Enterprise license is completely distinct with independent distribution certificates. If a company registers in five Enterprise programs they will be able to create five independent distribution certificates.

# Single View Application

The next project template we’ll look at is the Single View Application. Its central difference compared to the Empty Application template is that this one includes a custom view controller, which is properly installed as the main window’s root view controller. When creating an app from this template, you also get the choice whether you want to “Use [Storyboards](https://developer.apple.com/library/ios/#documentation/general/conceptual/Devpedia-CocoaApp/Storyboard.html)” or not. Let’s go with storyboards first.

## With Storyboards

The Single View Application comes with three new files: the app’s storyboard in MainStoryboard.storyboard and a header and implementation file for the view controller class, inheriting from [UIViewController](https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIViewController_Class/). The storyboard contains a single scene representing the view controller and its empty view.

Pay attention to the implementation of the app delegate. Contrary to the Empty Application, the application:didFinishLaunchingWithOptions: method contains no significant code at all:

**-** (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions { // Override point for customization after application launch. **return** YES; }

Where does our app’s main window come from then? It turns out that, by specifying a storyboard in the app’s Info.plist file (under the [UIMainStoryboard](https://developer.apple.com/library/ios/documentation/general/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW9) key), the UIApplication object automatically creates a window and assigns it to our app delegate’s window property.

 The UIApplication instance also loads the storyboard, initializes the storyboard’s initial view controller and installs it as the window’s [rootViewController](https://developer.apple.com/library/ios/documentation/uikit/reference/UIWindow_Class/UIWindowClassReference/UIWindowClassReference.html#//apple_ref/doc/uid/TP40006817-CH3-SW33).

This way, control reaches your root view controller’s viewDidLoad method (where you usually place code to configure your app’s initial view) without a single line of code. We will see this pattern in all other templates that use storyboards, as well.

## Without Storyboards

If you opt not to use a storyboard when you create your app, Xcode will set you up with traditional NIB/XIB files instead. In the case of a Single View Application, you will get one .xib file that contains the view of your root view controller. Note that the file MainWindow.xib that used to be present in older project templates is no longer there. Your app’s main window is created in code, just like we saw in the Empty Application template. In addition, the view controller is created and assigned to the window:

**-** (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions { self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]; // Override point for customization after application launch. self.viewController = [[ViewController alloc] initWithNibName:@"ViewController" bundle:nil]; self.window.rootViewController = self.viewController; [self.window makeKeyAndVisible]; **return** YES; }

This also illustrates what changes you would have to make if you wanted to convert an existing app’s root view controller into one that is loaded from a storyboard: first, create the storyboard file and set up the root view controller scene inside. Then add the UIMainStoryboardFile key to your Info.plist (removing the NSMainNibFile key at the same time if present) and remove the now unnecessary code from application:didFinishLaunchingWithOptions:.

# Utility Application

The Utility Application template includes two view controllers. The root view controller (MainViewController) is set up exactly like the Single View Application. It contains a button that lets the user switch to another view controller (presented modally) and back.

# Tabbed Application

The Tabbed Application template sets up an app with a tab bar controller displaying two tabs, each represented by another content view controller. The setup of this app is trivial and just as straightforward as a Single View Application. With storyboards, the entire thing is set up inside the storyboard without a single line of code: a UITabBarController acts as the storyboards initial view controller and is connected via relationship segues to its two content view controllers.

Without storyboards, the creation of the tab bar controller and the two content view controllers takes place in code but is just as simple:

**-** (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions { self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]; // Override point for customization after application launch. UIViewController \*viewController1 = [[FirstViewController alloc] initWithNibName:@"FirstViewController" bundle:nil]; UIViewController \*viewController2 = [[SecondViewController alloc] initWithNibName:@"SecondViewController" bundle:nil]; self.tabBarController = [[UITabBarController alloc] init]; self.tabBarController.viewControllers = @[viewController1, viewController2]; self.window.rootViewController = self.tabBarController; [self.window makeKeyAndVisible]; **return** YES; }

If you have never worked with a tab bar controller before, you might be surprised to see that the captions and images on the tab bar are actually properties of the respective content view controllers and not of the tab bar itself, but it makes sense from an encapsulation perspective: each view controller should know itself how it wants to represented in a tab bar.

# Master-Detail Application

The Master-Detail Application template is interesting for a number of reasons. Firstly, it is the only template that differs significantly depending on the device family (iPhone or iPad) you choose to target. Secondly, the content view controllers actually do include some meaningful degree of functionality and are not just empty placeholders.

## iPhone vs. iPad vs. Universal

The iPhone version of this template starts out with a UINavigationController as the root view controller. The navigation controller contains a table view controller (the master view), and tapping on a row in the table transitions to a second view controller (the detail view). The app template for the iPad begins with a UISplitViewController, which in turn contains the master table view on the left and the detail view on the right. If you choose to create a Universal app (targeting both iPhone and iPad), it will contain both variants in separate storyboards. In all cases, the setup of the view controllers is again quite simple and should be familiar by now.

This app template is a good reference on how to set up a Universal application (or two separate apps for iPhone and iPad) from a single codebase, despite seemingly big platform-dependent differences in the app’s user interface. It is important to note how much code can be shared between both targets and how little code is device-family-dependent. For instance, the two content view controllers can be largely ignorant of the specific device they run on.

## Content View Controllers

The MasterViewController class (inheriting from UITableViewController) implements a fully functional table view controller that manages an array of simple data objects (instances of NSDate, in this case) and acts as the data source for its table view. If you have never worked with table views before, this is a good starting place for your first experiments. Once you are familiar with UITableView, it is trivial to recreate an interface like this from scratch.

## With Core Data

With the “Use Core Data” option checked, the timestamp objects this template uses even get inserted into the data store and are properly saved. The interplay between data store and table view is not done manually but handled by an [NSFetchedResultsController](https://developer.apple.com/library/ios/#DOCUMENTATION/CoreData/Reference/NSFetchedResultsController_Class/Reference/Reference.html), making this option worse if your goal is to learn how UITableView works. Access to the Core Data stack is not very well encapsulated (see below for more on the “Use Core Data” option).

# Page-Based Application

This is a relatively new template. It showcases the [UIPageViewController](https://developer.apple.com/library/ios/#documentation/uikit/reference/UIPageViewControllerClassReferenceClassRef/UIPageViewControllerClassReference.html) class that Apple introduced in iOS 5.0. Because this template was never available in the time before storyboards, you don’t even get the option whether you want to use them or not – which is ironic since page view controllers are data-source-based and cannot even be set up entirely within a storyboard file like the other container view controllers.

It is important to understand the structure of a page-based app: the RootViewController is a custom UIViewController subclass that acts as a container view controller for a UIPageViewController. The page view controller, in turn, asks its datasource to create and return the view controller(s) whose views it then displays as its pages. In our case, each page is represented by an instance of DataViewController. You will find the setup code in -[RootViewController viewDidLoad]:

**-** (void)viewDidLoad { [super viewDidLoad]; // Do any additional setup after loading the view, typically from a nib. // Configure the page view controller and add it as a child view controller. self.pageViewController = [[UIPageViewController alloc] initWithTransitionStyle:UIPageViewControllerTransitionStylePageCurl navigationOrientation:UIPageViewControllerNavigationOrientationHorizontal options:nil]; self.pageViewController.delegate = self; DataViewController \*startingViewController = [self.modelController viewControllerAtIndex:0 storyboard:self.storyboard]; NSArray \*viewControllers = @[startingViewController]; [self.pageViewController setViewControllers:viewControllers direction:UIPageViewControllerNavigationDirectionForward animated:NO completion:NULL]; self.pageViewController.dataSource = self.modelController; [self addChildViewController:self.pageViewController]; [self.view addSubview:self.pageViewController.view]; // Set the page view controller's bounds using an inset rect so that // self's view is visible around the edges of the pages. CGRect pageViewRect = self.view.bounds; self.pageViewController.view.frame = pageViewRect; [self.pageViewController didMoveToParentViewController:self]; // Add the page view controller's gesture recognizers to the book view // controller's view so that the gestures are started more easily. self.view.gestureRecognizers = self.pageViewController.gestureRecognizers; }

## Better Code Practices

Since this is a more modern template, it also contains better code practices than some of the others. For instance, Apple has created a separate ModelController class (inheriting from NSObject, not to be confused with a view controller) that acts as the page view controller’s data source rather than putting that code directly into the root view controller. Try to do the same in your own apps and not stuff everything into your view controllers.

The DataViewController is included in the storyboard, even though it is not connected to the root view controller via segues. Every time the page view controller asks its data source for a new page, the ModelController tells the storyboard to instantiate a new DataViewController instance and returns it.

# OpenGL Game

The OpenGL Game template is the odd one out. It too uses UIKit like the other app templates, but only to set up an OpenGL ES 2.0 stack and give you a starting point for writing OpenGL code. It has been updated fairly recently and takes full advantage of the [GLKit framework](https://developer.apple.com/library/ios/#documentation/GLkit/Reference/GLKit_Collection/Introduction/Introduction.html%23//apple_ref/doc/uid/TP40010915-CH1-SW1) introduced in iOS 5.0.

When you create an OpenGL-based app, you will notice two new files in the project: Shader.vsh and Shader.fsh, a basic vertex and fragment shader. You will also notice that your app’s view controller inherits from [GLKViewController](https://developer.apple.com/library/ios/documentation/GLkit/Reference/GLKViewController_ClassRef/Reference/Reference.html#//apple_ref/doc/uid/TP40010925) and its view is a [GLKView](https://developer.apple.com/library/ios/documentation/GLkit/Reference/GLKView_ClassReference/Reference/Reference.html#//apple_ref/doc/uid/TP40010923).

The view controller contains almost 400 lines of code for setting up the OpenGL ES 2.0 stack

 and for a small sample scene. Surprisingly (and unlike the other templates), it even includes a useful implementation for the didReceiveMemoryWarning method.

If you want to write an OpenGL-based app, this template is a very good starting point. Setting up the OpenGL stack takes a lot of code that is not particularly well documented in one place anywhere else, and since the setup is platform-dependent, even OpenGL experts will benefit from using the template.

# Project Creation Options

Finally, let’s take a look at some of the options Xcode offers for the project templates. As you already know, not every option is available for every template, and some of them I have already discussed above. Here is what I have to say about the others.

## Targeted Device Family: iPhone vs. iPad vs. Universal

With the exception of the Master-Detail Application template, choosing between iPhone and iPad does not affect the projects in any substantial way. Storyboards and NIB files are generated in the appropriate size for the targeted device. For universal apps, Xcode creates separate storyboard or NIB files per target platform. The generated code is mostly identical with very few exceptions, like this example:

**-** (void)awakeFromNib { **if** ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPad) { self.clearsSelectionOnViewWillAppear = NO; self.contentSizeForViewInPopover = CGSizeMake(320.0, 600.0); } [super awakeFromNib]; }

## Use Automatic Reference Counting

I cannot imagine a reason why you should not enable ARC for new projects, other than wanting to learn about manual retain and release. Automatic Reference Counting makes your life as a Cocoa developer much easier.

## Use Core Data

The Core Data option is only available for an Empty Application, a Utility Application or a Master-Detail Application. Enabling it tells Xcode to create an empty .xcdatamodeld file for designing your app’s data model. You can easily create it manually if you decide to incorporate Core Data at a later time, though.

In addition, Xcode creates a bunch of code (about 100 lines) in your app delegate for setting up and accessing your Core Data stack, which mainly consists of three objects: a managedObjectContext, a mangedObjectModel and a persistentStoreCoordinator. Reading this code is certainly instructive (the comments contain lots of good information) and writing it from scratch can be difficult if you have never worked with Core Data before. However, there are so many deficiencies in Apple’s code that I only recommend it as a starting point. You should rewrite it significantly for your own needs.

Problems with Apple’s sample code include:

* Error handling is missing (though this is clearly documented in the comments).
* The -saveContext method is called from applicationWillTerminate: although it is clearly documented that this method is never called in the application lifecycle on modern multitasking-capable iOS devices. The correct place for saving is applicationDidEnterBackground:, which lacks a call to saveContext. This suggests that Apple has not touched this part of the sample code for years.
* The app delegate is arguably not the right place for managing your Core Data stack. You should create a separate class for it.
* The sample code creates the app’s data store file in the application documents directory, which is arguably not the right place for it. In most cases, the Library directory is probably a better choice because it does not give users access to the file (e.g. via iTunes file sharing).
* The code doesn’t use [Core Data’s new concurrency model](http://floriankugler.com/blog/2013/4/2/the-concurrent-core-data-stack), with separate managed objects contexts for the main queue and a background queue, even though Apple recommends to use it.

## Include Unit Tests

Checking this option tells Xcode to create a separate build target for running your unit tests. It is a good idea to set this up from the start as it does not interfere with the rest of your app in any way if you don’t use it. If you want to create a unit test target for an existing app at a later time, Apple provides [a step-by-step guide](https://developer.apple.com/library/mac/#documentation/developertools/Conceptual/UnitTesting/02-Setting_Up_Unit_Tests_in_a_Project/setting_up.html). Make sure to set up a target for what Apple calls “Application Tests” (as opposed to “Logic Tests”).

# Conclusion

Apple’s iOS app project templates are good starting points if you want to learn how certain standard app patterns are architected. It is essential to understand that the available project templates do not vary much from each other (with the exception of the OpenGL template). It does not really matter from which template you start your project as you can easily switch to (or incorporate) another at any time.

Also remember that – like all of Apple’s sample code – the app templates do not necessarily represent best practices and some parts of them have not been updated in years. Use them for getting ideas but don’t be afraid to do stuff differently. And right after creating a new app, [enable more compiler warnings](https://oleb.net/blog/2013/04/compiler-warnings-for-objective-c-developers/).

Personally, I start nearly all my projects from the Single View Application template and build from there.

RichEditorView is a simple, modular, drop-in UIView subclass for Rich Text Editing.

When you enroll in the Apple Developer Program, Apple Developer Enterprise Program, or iOS Developer University Program, you automatically become the Account Holder for your membership. If you’re enrolled as an organization, you have the option of adding additional members to your team. The role you assign them controls access to the development and distribution tools included with your membership.

## Roles for the Apple Developer Program

Each organization has one team with one set of roles across the Apple Developer website and App Store Connect. Management of users and roles will be done in App Store Connect.

In addition, the Team Agent role and Legal role are now called "Account Holder".

### Permissions on the Apple Developer Website

|  | **Account Holder** | **Admin** | **App Manager** | **Developer** | **Finance** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| Eligible for events and additional benefits | ● | ● | ● | ● | ● | ● | ● | ● |
| Purchase and submit Technical Support Incidents | ● | ● | ● | ● | ● | ● | ● | ● |
| Download beta software | ● | ● | ● | ● | ● | ● | ● | ● |
| Post in Developer Forums | ● | ● | ● | ● | ● | ● | ● | ● |
| Accept legal agreements | ● |  |  |  |  |  |  |  |
| Renew membership | ● |  |  |  |  |  |  |  |
| Submit certificate signing requests | ● | ○ | ○ | ○ |  |  |  |  |
| Create and revoke development certificates | ● | ○ | ○ | ○ |  |  |  |  |
| Create and revoke distribution certificates | ● | ○ | ○ |  |  |  |  |  |
| Create Developer ID certificates | ● |  |  |  |  |  |  |  |
| Create other certificate types | ● | ○ | ○ |  |  |  |  |  |
| Create development provisioning profiles | ● | ○ | ○ | **+** |  |  |  |  |
| Delete development provisioning profiles | ● | ○ | ○ |  |  |  |  |  |
| Create and delete distribution provisioning profiles | ● | ○ | ○ |  |  |  |  |  |
| Download provisioning profiles | ● | ○ | ○ | ○ |  |  |  |  |
| Register and configure App IDs | ● | ○ | ○ | **+** |  |  |  |  |
| Delete App IDs | ● | ○ | ○ |  |  |  |  |  |
| Add UDIDs | ● | ○ | ○ | **+** |  |  |  |  |
| Disable UDIDs | ● | ○ | ○ |  |  |  |  |  |
| Create keys | ● | ○ | ○ |  |  |  |  |  |
| Create and revoke Safari Extension Certificates | ● | ○ | ○ | ○ |  |  |  |  |
| Notarize software | ● | ● | ● | ● |  |  |  |  |

○ Requires access to Certificates, Identifiers & Profiles, which can be provided in App Store Connect. This includes access to CloudKit Dashboard.  
**+** Can be done in Xcode 7 or later with access to Certificates, Identifiers & Profiles.

### Permissions in App Store Connect

Note: If you’re enrolled as an individual and add users in App Store Connect, users receive access only to your content in App Store Connect and are not considered part of your team in the Apple Developer Program.

| Agreements, Tax, and Banking | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Account Holder** | **Admin** | **App Manager** | **Developer** | **Finance** | **Marketing** | **Sales** | **Customer Support** |
|  | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Manage agreements, tax, and banking | ● | ● |  |  | ● |  |  |  |
| Sign Agreements | ● |  |  |  |  |  |  |  |

| Users and Access | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Account Holder** | **Admin** | **App Manager** | **Developer** | **Finance** | **Marketing** | **Sales** | **Customer Support** |
|  | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Manage users and roles | ● | ● | ● |  |  |  |  |  |
| Manage app access | ● | ● | ● |  |  |  |  |  |
| Manage sandbox testers | ● | ● | ● |  |  |  |  |  |
| Generate API keys | ● | ● |  |  |  |  |  |  |

| My Apps | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Account Holder** | **Admin** | **App Manager** | **Developer** | **Finance** | **Marketing** | **Sales** | **Customer Support** |
|  | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| Create apps and submit versions | ● | ● | ● |  |  |  |  |  |
| Edit App Store details | ● | ● | ● |  |  | ● |  |  |
| Edit app pricing and availability | ● | ● | ● |  |  | ○ |  |  |
| Transfer and delete apps | ● |  |  |  |  |  |  |  |
| Reset summary rating | ● | ● | ● |  |  |  |  |  |
| Create in-app purchases | ● | ● | ● | ● |  | ● |  |  |
| Submit in-app purchases | ● | ● | ● |  |  |  |  |  |
| Edit in-app purchases | ● | ● | ● | ○ |  | ● |  |  |
| Manage Game Center | ● | ● | ● | ● |  | ● |  |  |
| Manage leaderboard scores | ● | ● | ● |  |  |  |  |  |
| Manage promo codes and promo art | ● | ● | ● |  |  | ● |  |  |
| Manage TestFlight builds | ● | ● | ● | ○ |  | ○ |  |  |
| Manage TestFlight testers | ● | ● | ● | **+** |  | **+** |  |  |
| Upload builds | ● | ● | ● | ● |  |  |  |  |
| View Ratings and Reviews | ● | ● | ● | ● |  | ● |  | ● |
| Respond to customer reviews | ● | ● |  |  |  |  |  | ● |

○ Read Only  
**+** Internal Testers Only

| Reporting and Analytics | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Account Holder** | **Admin** | **App Manager** | **Developer** | **Finance** | **Marketing** | **Sales** | **Customer Support** |
|  | | | | | | | | |
|  |  |  |  |  |  |  |  |  |
| View app analytics | ● | ● |  |  | ● |  | ● |  |
| View sales and trends | ● | ● | ● |  |  |  | ● |  |
| View sales and trends reports | ● | ● |  |  | ● |  | ○ |  |
| View payments and financial reports | ● | ● |  |  | ● |  |  |  |

○ Requires access to reports, which can be provided in App Store Connect.

## Roles for the Apple Developer Enterprise Program

Note: Since this program does not include App Store distribution, membership benefits do not include access to App Store Connect. User management is done in the People section of your account.

### Permissions on the Apple Developer Website

|  | **Account Holder** | **Admin** | **Developer** |
| --- | --- | --- | --- |
|  |  |  |  |
| Accept Legal Agreements | ● |  |  |
| Renew Membership | ● |  |  |
| Create Developer ID Certificates | ● |  |  |
| Invite Members and Assign Roles | ● | ● |  |
| Create Provisioning Profiles | ● | ● | ○ |
| Approve Certificate Signing Requests | ● | ● |  |
| Add and Disable UDIDs | ● | ● | ○ |
| Register and Configure App IDs | ● | ● | ○ |
| Delete App IDs | ● | ● |  |
| Create an iOS Distribution Certificate and Distribution Provisioning Profiles | ● | ● |  |
| Create Certificates for Apple Push Notification service and Pass Type IDs | ● | ● |  |
| Create and Revoke Development Certificates | ● | ● | ● |
| Purchase and Submit Technical Support Incidents | ● | ● | ● |
| Post in Developer Forums | ● | ● | ● |
| Download Beta Software | ● | ● | ● |
| Download Provisioning Profiles | ● | ● | ● |
| Submit Certificate Signing Requests | ● | ● | ● |
| Notarize software | ● | ● | ● |

○ Access requires Xcode 7 or later.

## Roles in the iOS Developer University Program

Note: Since this program does not include App Store distribution, membership benefits do not include access to App Store Connect. User management is done in the People section of your account.

### Permissions on the Apple Developer Website

|  | **Account Holder** | **Admin** | **Developer** |
| --- | --- | --- | --- |
|  |  |  |  |
| Accept Legal Agreements | ● |  |  |
| Renew Membership | ● |  |  |
| Create Developer ID Certificates | ● |  |  |
| Invite Members and Assign Roles | ● | ● |  |
| Create Provisioning Profiles | ● | ● |  |
| Approve Certificate Signing Requests | ● | ● |  |
| Add and Disable UDIDs | ● | ● |  |
| Register and Configure App IDs | ● | ● | ○ |
| Delete App IDs | ● | ● |  |
| Create an iOS Distribution Certificate and Distribution Provisioning Profiles | ● | ● |  |
| Create Certificates for Apple Push Notification service and Pass Type IDs | ● | ● |  |
| Create and Revoke Development Certificates | ● | ● | ● |
| Post in Developer Forums | ● | ● | ● |
| Download Provisioning Profiles | ● | ● | ● |
| Submit Certificate Signing Requests | ● | ● | ● |

○ Access requires Xcode 7 or later.

# Unified Development Teams for Organizations

Starting February 12, 2019 development teams for organizations in the Apple Developer Program will be unified across the Apple Developer website and App Store Connect, with one team and one set of roles.

## Overview

Currently, each Apple Developer Program organization membership includes a development team with access to resources on the Apple Developer website and separate access to App Store Connect. Each site has its own set of members and roles — Apple Developer Program roles determine access to membership benefits related to app development and distribution, such as Certificates, Identifiers & Profiles, while App Store Connect roles determine access to app management, financial information, analytics, and more for apps on the App Store.

Starting February 12, 2019, these teams will be unified for organization memberships. Each organization will have one team with one set of roles across the Apple Developer website and App Store Connect. Management of users and roles will be done in App Store Connect.

These changes will not affect individual memberships or Apple Developer Enterprise Program memberships.

## How Roles will be Unified

### Members of a Team on the Apple Developer Website and App Store Connect

If you're part of an organization's team on the Apple Developer website and in App Store Connect, you will keep your current App Store Connect role. However, permissions for your current App Store Connect role may change, you may be assigned additional App Store Connect roles, or you may have modified access to Certificates, Identifiers & Profiles on the Apple Developer website.

### Members of a Team on the Apple Developer Website Only

If you’re part of an organization's team on the Apple Developer website but not part of its team in App Store Connect, you'll gain access to all apps associated with the organization's team in App Store Connect and be assigned the Developer role by default. Please note that the Developer role does not include the ability to create distribution and service certificates, approve certificate signing requests, or delete app IDs.

### Members of a Team in App Store Connect Only

If you’re part of an organization's team in App Store Connect but not part of its team on the Apple Developer website, your new role may include access to additional membership benefits or developer tools, including Certificates, Identifiers & Profiles and CloudKit dashboard.

## Editing Roles

Changes to roles and permissions can be made in the [Users and Roles](https://appstoreconnect.apple.com/access/users) section of App Store Connect, accessible by Account Holders and admins they've added to App Store Connect. Contact your Account Holder or an admin if you require changes.

## Unified Role Permissions

These upcoming role updates include permissions across the Apple Developer website and App Store Connect. Please note that the “Account Holder” role on the Apple Developer website and the “Legal” role in App Store Connect will be called “Account Holder”.

### Membership Benefits on the Apple Developer Website

|  | **Account Holder** | **Admin** | **Finance** | **App Manager** | **Developer** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Eligible for events and additional benefits | ● | ● | ● | ● | ● | ● | ● | ● |
| Purchase and submit Technical Support Incidents | ● | ● | ● | ● | ● | ● | ● | ● |
| Download beta software | ● | ● | ● | ● | ● | ● | ● | ● |
| Post in Developer Forums | ● | ● | ● | ● | ● | ● | ● | ● |
| Accept legal agreements | ● |  |  |  |  |  |  |  |
| Renew membership | ● |  |  |  |  |  |  |  |
| Submit certificate signing requests | ● | ○ |  | ○ | ○ |  |  |  |
| Create and revoke development certificates | ● | ○ |  | ○ | ○ |  |  |  |
| Create and revoke distribution certificates | ● | ○ |  | ○ |  |  |  |  |
| Create Developer ID certificates | ● |  |  |  |  |  |  |  |
| Create other certificate types | ● | ○ |  | ○ |  |  |  |  |
| Create development provisioning profiles | ● | ○ |  | ○ | **+** |  |  |  |
| Delete development provisioning profiles | ● | ○ |  | ○ |  |  |  |  |
| Create and delete distribution provisioning profiles | ● | ○ |  | ○ |  |  |  |  |
| Download provisioning profiles | ● | ○ |  | ○ | ○ |  |  |  |
| Register and configure App IDs | ● | ○ |  | ○ | **+** |  |  |  |
| Delete App IDs | ● | ○ |  | ○ |  |  |  |  |
| Add UDIDs | ● | ○ |  | ○ | **+** |  |  |  |
| Disable UDIDs | ● | ○ |  | ○ |  |  |  |  |
| Create keys | ● | ○ |  | ○ |  |  |  |  |
| Create and revoke Safari Extension Certificates | ● | ○ |  | ○ | ○ |  |  |  |

○ Requires access to Certificates, Identifiers & Profiles, which can be provided in App Store Connect. This includes access to CloudKit Dashboard.  
**+** Can be done in Xcode 7 or later with access to Certificates, Identifiers & Profiles.

### App Store Connect

#### Agreements, Tax, and Banking

|  | **Account Holder** | **Admin** | **Finance** | **App Manager** | **Developer** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Manage agreements, tax, and banking | ● | ● | ● |  |  |  |  |  |
| Sign Agreements | ● |  |  |  |  |  |  |  |

#### Users and Access

|  | **Account Holder** | **Admin** | **Finance** | **App Manager** | **Developer** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Manage users and roles | ● | ● |  | ● |  |  |  |  |
| Manage app access | ● | ● |  | ● |  |  |  |  |
| Manage sandbox testers | ● | ● |  | ● |  |  |  |  |
| Generate API keys | ● | ● |  |  |  |  |  |  |

#### My Apps

|  | **Account Holder** | **Admin** | **Finance** | **App Manager** | **Developer** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Create apps and submit versions | ● | ● |  | ● |  |  |  |  |
| Edit App Store details | ● | ● |  | ● |  | ● |  |  |
| Edit app pricing and availability | ● | ● |  | ● |  | ○ |  |  |
| Transfer and delete apps | ● |  |  |  |  |  |  |  |
| Reset summary rating | ● | ● |  | ● |  |  |  |  |
| Create in-app purchases | ● | ● |  | ● | ● | ● |  |  |
| Submit in-app purchases | ● | ● |  | ● |  |  |  |  |
| Edit in-app purchases | ● | ● |  | ● | ○ | ● |  |  |
| Manage Game Center | ● | ● |  | ● | ● | ● |  |  |
| Manage leaderboard scores | ● | ● |  | ● |  |  |  |  |
| Manage promo codes and promo art | ● | ● |  | ● |  | ● |  |  |
| Manage TestFlight builds | ● | ● |  | ● | ○ | ○ |  |  |
| Manage TestFlight testers | ● | ● |  | ● | **+** | **+** |  |  |
| Upload builds | ● | ● |  | ● | ● |  |  |  |
| View Ratings and Reviews | ● | ● |  | ● | ● | ● |  | ● |
| Respond to customer reviews | ● | ● |  |  |  |  |  | ● |

○ Read Only  
**+** Internal Testers Only

#### Reporting and Analytics

|  | **Account Holder** | **Admin** | **Finance** | **App Manager** | **Developer** | **Marketing** | **Sales** | **Customer Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| View app analytics | ● | ● | ● |  |  |  | ● |  |
| View sales and trends | ● | ● | ● |  |  |  | ● |  |
| View sales and trends reports | ● | ● | ● |  |  |  | ○ |  |
| View payments and financial reports | ● | ● | ● |  |  |  |  |  |

○ Requires access to reports, which can be provided in App Store Connect.

Type

# UserRole

Strings representing user roles in App Store Connect.

## Declaration

string UserRole

## Possible Values

**ADMIN**

Serves as a secondary contact for teams and has many of the same responsibilities as the Account Holder role. Admins have access to all apps.

**FINANCE**

Manages financial information, including reports and tax forms. A user assigned this role can view all apps in Payments and Financial Reports, Sales and Trends, and App Analytics.

**TECHNICAL**

The Technical role is no longer assignable to new users in App Store Connect. Existing users with the Technical role can manage all the aspects of an app, such as pricing, App Store information, and app development and delivery. Techncial users have access to all apps.

**SALES**

Analyzes sales, downloads, and other analytics for the app.

**MARKETING**

Manages marketing materials and promotional artwork. A user assigned this role will be contacted by Apple if the app is in consideration to be featured on the App Store.

**DEVELOPER**

Manages development and delivery of an app.

**ACCOUNT\_HOLDER**

Responsible for entering into legal agreements with Apple. The person who completes program enrollment is assigned the Account Holder role in both the Apple Developer account and App Store Connect.

**READ\_ONLY**

**APP\_MANAGER**

Manages all aspects of an app, such as pricing, App Store information, and app development and delivery.

**ACCESS\_TO\_REPORTS**

Downloads reports associated with a role. The Access To Reports role is an additional permission for users with the App Manager, Developer, Marketing, or Sales role. If this permission is added, the user has access to all of your apps.

**CUSTOMER\_SUPPORT**

Analyzes and responds to customer reviews on the App Store. If a user has only the Customer Support role, they'll go straight to the Ratings and Reviews section when they click on an app in My Apps.

<https://developer.apple.com/documentation/appstoreconnectapi/userrole>

<https://developer.apple.com/support/teams/>

<https://developer.apple.com/support/roles/>

|  |
| --- |
| **Internet protocol suite** |
| [**Application layer**](https://en.wikipedia.org/wiki/Application_layer) |
| * [BGP](https://en.wikipedia.org/wiki/Border_Gateway_Protocol) * [DHCP](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol) * [DNS](https://en.wikipedia.org/wiki/Domain_Name_System) * [FTP](https://en.wikipedia.org/wiki/File_Transfer_Protocol) * [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) * [HTTPS](https://en.wikipedia.org/wiki/HTTPS) * [IMAP](https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) * [LDAP](https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol) * [MGCP](https://en.wikipedia.org/wiki/Media_Gateway_Control_Protocol) * [MQTT](https://en.wikipedia.org/wiki/MQTT) * [NNTP](https://en.wikipedia.org/wiki/Network_News_Transfer_Protocol) * [NTP](https://en.wikipedia.org/wiki/Network_Time_Protocol) * [POP](https://en.wikipedia.org/wiki/Post_Office_Protocol) * [ONC/RPC](https://en.wikipedia.org/wiki/Open_Network_Computing_Remote_Procedure_Call) * [RTP](https://en.wikipedia.org/wiki/Real-time_Transport_Protocol) * [RTSP](https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol) * [RIP](https://en.wikipedia.org/wiki/Routing_Information_Protocol) * [SIP](https://en.wikipedia.org/wiki/Session_Initiation_Protocol) * [SMTP](https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol) * [SNMP](https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol) * [SSH](https://en.wikipedia.org/wiki/Secure_Shell) * [Telnet](https://en.wikipedia.org/wiki/Telnet) * [TLS/SSL](https://en.wikipedia.org/wiki/Transport_Layer_Security) * [XMPP](https://en.wikipedia.org/wiki/XMPP) * [*more...*](https://en.wikipedia.org/wiki/Category:Application_layer_protocols) |

Its very simple:

**For ObjC:**

NSString \*string1 = @"This is"; NSString \*string2 = @"Swift Language";

**ForSwift:**

let string1 = "This is" let string2 = "Swift Language"

**For ObjC AppendString:**

NSString \*appendString = [NSString stringWithFormat:@"%@ %@",string1,string2];

**For Swift AppendString:**

var appendString1 = "\(string1) \(string2)"

var appendString2 = string1+string2

**Result:**

print("APPEND STRING 1:\(appendString1)")

print("APPEND STRING 2:\(appendString2)")

**Complete Code In Swift:**

let string1 = "This is" let string2 = "Swift Language"

var appendString = "\(string1) \(string2)"

var appendString1 = string1+string2

print("APPEND STRING1:\(appendString1)")

print("APPEND STRING2:\(appendString2)")

However, using Bluetooth Low Energy, you don’t necessarily have to pair with a device in order to perform useful work. A number of health-related medical devices, such as the Heart Rate Monitor and Blood Pressure Monitor, allow your mobile device to connect to them and receive vital information about your health. One very cool use of Bluetooth LE is Apple’s iBeacon. Using an iBeacon, a device can detect the presence of an iBeacon and use that information to provide locale-specific functions, such as merchandise promotion, payment information, etc.

[iBeacon is an Apple concept](https://developer.apple.com/ibeacon/) that allows Bluetooth devices to send and capture information within short distances.

There will be a beacon device and a receiver (smartphone app) that can send hyper-contextual text on a micro-local scale.

When you posses a mobile device with this technology activated in it, you are continuously sending out information regarding your location and ID.

These beacons act like links to the outside world. An app uses these beacons to trigger events or call actions, like for example, allow the user to open locked doors, activate automation systems, receive notifications and so on.

The purpose of the broadcaster (a battery powered device) is to just send information, and the receiver will be an iOS app on a laptop or any mobile device and the information will be sent through a USB Bluetooth dongle and an [Arduino kit.](https://www.arduino.cc/)

The technology used by Beacons is known as [BLE or Bluetooth Low Energy](https://en.wikipedia.org/wiki/Bluetooth_low_energy) and is actually a wireless personal area network that transmits data over short distances.

Now, that the introduction is given, let’s get into the use cases of iBeacon technology:

#### ****1. Korean Railways****

Look at the innovative use of iBeacon on Korean trains.

As soon as a pregnant lady gets into the train, she gets a seat. The campaign was actually started [under the name “Pink Light”](http://www.reuters.com/article/us-south-korea-pregnant-beacon-idUSKCN0ZN1ZV), owning to the low birthrate of women in Korea. So to ensure that the women stay safe, the railways decided to make their journey comfortable and safe by providing them with a seat as soon as they enter.

**How do they do it?**  
By providing pregnant women with small beacons that they can wear on their handbags. This beacon would then transmit a flashlight signaling that the lady is on a train.

The beacon will also alert other people on the train that a pregnant lady is on board and that she needs a vacant seat. Passengers seated on the courtesy seat would have to give up their seats. The beacon would keep flashing until the lady is seated.

#### ****2. Allrecipes****

Allrecipes is a digital food brand, and they have an app that gives you exciting food tips and cooking ideas.

Through the [Allrecipes Dinner Spinner app](http://dish.allrecipes.com/mobile-apps/), you can come up with customized recipes if you ever run out of recipes. They aim to take the burden off your shoulders, giving you ideas on how to limit your grocery bill as well.

**How do they do it?**All you need to do is download the Spinner app, and as soon as you enter the store, the beacon within the app will be activated through the Bluetooth in your device. It would then give recipe suggestions based on the grocery available at a particular store.

#### ****3. Philips Museum, Eindhoven, The Netherlands, National Slate Museum, Wales, Brooklyn Museum, New York****

iBeacons play a major role in every industry. We are not discussing retail here because that’s an obvious one. Instead, let’s look at how museums utilize iBeacon technology in a big way. After all, they need to do a serious take on how to bring in more visitors and keep them engaged.

**How do they do it?**  
Museums can send interactive content on the artwork visitors are interested in. They do this by identifying the time a visitor dwells on a particular object and send him/her more information on it.[Museums use iBeacon technology](https://www.brooklynmuseum.org/community/blogosphere/2014/10/14/positioning-visitors-with-ibeacons/)to give self-guided tours to visitors.

They can learn about the layout of the museum, the goods on display and a lot more through this technology. And they even go a step further, by issuing entry tickets to visitors so they don’t actually have to stand in queue to get one.

#### ****4. Hotels like Casino, Marriot & IHG****

Hotels aim to deliver a tech-savvy experience to their customers. They transform guest experience by providing a number of excellent features to enable personalized services. If hotels launched apps to help guests reserve their rooms in the past, they now use iBeacons to act as their personal concierge.

**How do they do it?**  
The moment a guest approaches the front desk, the clerk can tap into the micro- location capabilities through the iBeacons and gather pertinent information regarding the new entrant. The guest is saved the task of giving all his details manually.

iBeacons play a major role in keyless entry into rooms, provide guided tours to the various amenities within the facility and send specifically targeted messages and alerts by watching their digital footprint.

#### ****5. Huge****

Huge is a Digital Advertising company, headquartered in Brooklyn, that installed iBeacons to keep track of their interns’ projects. Many companies in the retail sector are already doing this. Apart from ethical employee tracking, you can also monitor payroll accuracy, labor code compliance, workplace issues and even budgeting.

**How do they do it?**Huge has a huge office and apart from monitoring the work of their interns through iBeacons, guidance is also provided to them regarding the physical layout of the workplace. This is all done through the iBeacons installed in their mobile devices.

BOOL b; NSLog(@"Bool value: %d",b);

BOOL flag = YES; NSLog(flag ? @"Yes" : @"No");

NFC is a lot like RFID, only it's a more up-close-and-personal type of wireless. Whereas RFID can be used from a distance, NFC readers work at a maximum range of about 4 inches (10 centimeters). NFC readers aren't suitable for RFID-style inventory tracking; their range is too short. So NFC tags will appear in a flood of products and promotional items where bits of digitized information might come in handy.

Unlike RFID versions, NFC readers aren't always specialized devices. As a matter of fact, NFC chips will be incorporated right into your smartphone's circuitry. About 20 percent of phones worldwide might have NFC capabilities by 2014 [source: [Juniper](http://www.juniperresearch.com/viewpressrelease.php?id=308&pr=239)]. With the widespread reach of NFC phones, NFC tags could one day become as commonplace as bar codes.

You'll have instant access to information about products, services, landmarks and even people, all thanks to **near field communication** (**NFC**) technologyand the **smart tags** that work with [NFC](https://electronics.howstuffworks.com/near-field-communication.htm).

NFC isn't a fundamentally groundbreaking technology. Like [Bluetooth](https://electronics.howstuffworks.com/bluetooth.htm) and [WiFi](https://computer.howstuffworks.com/wireless-network.htm), it's a wireless radio communications standard. In the wireless world, NFC's closest relative is actually **RFID (radio frequency identification)**. Retailers and parcel shipping companies in particular love RFID as a way to keep tabs on inventory supplies and shipments. You can read all about it in [How RFID Works](https://electronics.howstuffworks.com/gadgets/high-tech-gadgets/rfid3.htm).

**Near-field communication** (**NFC**) is a set of [communication protocols](https://en.wikipedia.org/wiki/Communications_protocol) that enable two electronic devices, one of which is usually a portable device such as a [smartphone](https://en.wikipedia.org/wiki/Smartphone), to establish [communication](https://en.wikipedia.org/wiki/Data_communication) by bringing them within 4 cm (1​1⁄2 in) of each other.[[1]](https://en.wikipedia.org/wiki/Near-field_communication#cite_note-1)

NFC devices are used in [contactless payment](https://en.wikipedia.org/wiki/Contactless_payment) systems, similar to those used in credit cards and [electronic ticket](https://en.wikipedia.org/wiki/Electronic_ticket) smart cards and allow [mobile payment](https://en.wikipedia.org/wiki/Mobile_payment) to replace or supplement these systems. This is sometimes referred to as NFC/CTLS (contactless) or CTLS NFC. NFC is used for [social networking](https://en.wikipedia.org/wiki/Social_networking_service), for sharing contacts, photos, videos or files.[[2]](https://en.wikipedia.org/wiki/Near-field_communication#cite_note-Pelly2011-2) NFC-enabled devices can act as electronic [identity documents](https://en.wikipedia.org/wiki/Identity_document) and [keycards](https://en.wikipedia.org/wiki/Keycard_lock).[[3]](https://en.wikipedia.org/wiki/Near-field_communication#cite_note-TechEnabler-3) NFC offers a low-speed connection with simple setup that can be used to [bootstrap](https://en.wikipedia.org/wiki/Bootstrapping) more capable wireless connections.[[3]](https://en.wikipedia.org/wiki/Near-field_communication#cite_note-TechEnabler-3)

NFC-enabled portable devices can be provided with [application software](https://en.wikipedia.org/wiki/Application_software), for example, to read electronic tags or make payments when connected to an NFC-compliant apparatus. Earlier close-range communication used technology that was proprietary to the manufacturer for applications such as stock tickets, access control and payment readers.

**Radio-frequency identification** (**RFID**) uses [electromagnetic fields](https://en.wikipedia.org/wiki/Electromagnetic_field) to automatically identify and track tags attached to objects. The tags contain electronically stored information. Passive tags collect energy from a nearby RFID reader's interrogating [radio waves](https://en.wikipedia.org/wiki/Radio_wave). Active tags have a local power source (such as a battery) and may operate hundreds of meters from the RFID reader. Unlike a [barcode](https://en.wikipedia.org/wiki/Barcode), the tags don't need to be within the line of sight of the reader, so it may be embedded in the tracked object. RFID is one method of [automatic identification and data capture](https://en.wikipedia.org/wiki/Automatic_identification_and_data_capture) (AIDC).[[1]](https://en.wikipedia.org/wiki/Radio-frequency_identification#cite_note-1)

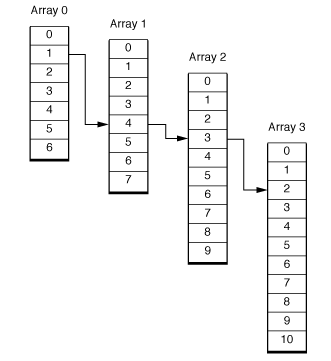
RFID tags are used in many industries. For example, an RFID tag attached to an automobile during production can be used to track its progress through the assembly line; RFID-tagged pharmaceuticals can be tracked through warehouses; and [implanting RFID microchips](https://en.wikipedia.org/wiki/Microchip_implant_(animal)) in livestock and pets enables positive identification of animals.

Since RFID tags can be attached to cash, clothing, and possessions, or implanted in animals and people, the possibility of reading personally-linked information without consent has raised serious privacy concerns.[[2]](https://en.wikipedia.org/wiki/Radio-frequency_identification#cite_note-Angell-2) These concerns resulted in standard specifications development addressing privacy and security issues. [ISO/IEC 18000](https://en.wikipedia.org/wiki/ISO/IEC_18000) and ISO/IEC 29167 use on-chip [cryptography](https://en.wikipedia.org/wiki/Cryptography) methods for untraceability, tag and reader [authentication](https://en.wikipedia.org/wiki/Authentication), and over-the-air privacy. [ISO/IEC 20248](https://en.wikipedia.org/wiki/ISO/IEC_20248) specifies a [digital signature](https://en.wikipedia.org/wiki/Digital_signature) data structure for RFID and [barcodes](https://en.wikipedia.org/wiki/Barcodes) providing data, source and read method authenticity. This work is done within [ISO/IEC JTC 1/SC 31 Automatic identification and data capture techniques](https://en.wikipedia.org/wiki/ISO/IEC_JTC_1/SC_31_Automatic_identification_and_data_capture_techniques). Tags can also be used in shops to expedite checkout, and to prevent theft by customers and employees.

**First info about NSIndexPath**

The NSIndexPath class represents the path to a specific node in a tree of nested array collections. This path is known as an index path.

Each index in an indexPath represents the index into an array of children from one node in the tree to another, deeper node.

**For example**, the indexPath **1.4.3.2** specifies the path shown in Figure 

Here in your case indexPath.row returns the index of the row at the specific indexPath.

**Differences between** indexPath.row and indexPath.item

Generally indexPath has **two properties**

1 - **row**

2 - **item**

**row -** property use with UITableView for get specific **row** base on indexPath. it is also read only property

Available in iOS 2.0 and later.

**item -** properly use with UICollectionView for get **item** in section. It is a read-only property. To use this property you need to declare it in  
**UICollectionView.h**

> Available in iOS 6.0 and later.

With Swift 5, according to your needs, you may choose one of the **six following ways** to concatenate/merge two arrays.

## #1. Merge two arrays into a new array with Array's +(\_:\_:) generic operator

let array1 = [1, 2, 3]

let array2 = [4, 5, 6]

let flattenArray = array1 + array2

print(flattenArray) // prints [1, 2, 3, 4, 5, 6]

## #2. Append the elements of an array into an existing array with Array's +=(\_:\_:) generic operator

var array1 = [1, 2, 3]

let array2 = [4, 5, 6]

array1 += array2

print(array1) // prints [1, 2, 3, 4, 5, 6]

## #3. Append an array to another array with Array's append(contentsOf:) method

var array1 = [1, 2, 3]

let array2 = [4, 5, 6]

array1.append(contentsOf: array2)

print(array1) // prints [1, 2, 3, 4, 5, 6]

## #4. Merge two arrays into a new array with Sequence's flatMap(\_:) method

let array1 = [1, 2, 3]

let array2 = [4, 5, 6]

let flattenArray = [array1, array2].flatMap({ (element: [Int]) -> [Int] in return element })

print(flattenArray) // prints [1, 2, 3, 4, 5, 6]

## #5. Merge two arrays into a new array with Sequence's joined() method and Array's init(\_:) initializer

let array1 = [1, 2, 3]

let array2 = [4, 5, 6]

let flattenCollection = [array1, array2].joined() // type: FlattenBidirectionalCollection<[Array<Int>]>

let flattenArray = Array(flattenCollection)

print(flattenArray) // prints [1, 2, 3, 4, 5, 6]

## #6. Merge two arrays into a new array with Array's reduce(\_:\_:) method

let array1 = [1, 2, 3]

let array2 = [4, 5, 6]

let flattenArray = [array1, array2].reduce([], { (result: [Int], element: [Int]) -> [Int] in return result + element })

print(flattenArray) // prints [1, 2, 3, 4, 5, 6]

The compare methods in Cocoa and Cocoa Touch return how the compared objects should be ordered, instead of just returning a boolean that tells whether the values are the same or not. There are three values:

* NSOrderedAscending: The left operand is smaller than the right operand.
* NSOrderedSame: The two operands are equal.
* NSOrderedDescending: The left operand is greater than the right operand.

So your code simply checks whether the string pointed to by result is equal to the string *"ERROR"*, ignoring differences in case (that is, *"error"*, *"eRRoR"* etc. are all considered to be equal to *"ERROR"*).

# NSComparisonResult

Constants that indicate sort order.

These constants are used to indicate how items in a request are ordered, from the first one given in a method invocation or function call to the last (that is, left to right in code).

[NSOrderedAscending](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsorderedascending?language=objc)

The left operand is smaller than the right operand.

[NSOrderedSame](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsorderedsame?language=objc)

The two operands are equal.

[NSOrderedDescending](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsordereddescending?language=objc)

The left operand is greater than the right operand.

Get Dates:

// Get current date let dateA = NSDate() // Get a later date (after a couple of milliseconds)

let dateB = NSDate()

Using SWITCH Statement

// Compare them

switch dateA.compare(dateB) {

case .OrderedAscending : print("Date A is earlier than date B")

case .OrderedDescending : print("Date A is later than date B")

case .OrderedSame : print("The two dates are the same")

}

using IF Statement

if dateA.compare(dateB) == .orderedAscending {

datePickerTo.date = datePicker.date

}

//OR if case .orderedAcending = dateA.compare(dateB) { }

# compare:

Indicates the temporal ordering of the receiver and another given date.

## Parameters

**anotherDate**

The date with which to compare the receiver.

This value must not be nil. If the value is nil, the behavior is undefined and may change in future versions of macOS.

## Return Value

If:

* The receiver and anotherDate are exactly equal to each other, [NSOrderedSame](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsorderedsame?language=objc)
* The receiver is later in time than anotherDate, [NSOrderedDescending](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsordereddescending?language=objc)
* The receiver is earlier in time than anotherDate, [NSOrderedAscending](https://developer.apple.com/documentation/foundation/nscomparisonresult/nsorderedascending?language=objc).

<https://stackoverflow.com/questions/tagged/swift>

let currentVersion = "3.0.1"

let storeVersion = "3.0.2"

if storeVersion.compare(currentVersion, options: .numeric) == .orderedDescending {

print("store version is newer")

}

let version = "1.0.0"

let targetVersion = "0.5.0" version.compare(targetVersion, options: .numeric) == .orderedSame // false version.compare(targetVersion, options: .numeric) == .orderedAscending // false

version.compare(targetVersion, options: .numeric) == .orderedDescending // true

Objective-C was created by Brad Cox and Tom Love in 1984 as an extension of C. It added SmallTalk style messaging and Object Orientation to the C language.

# Comparative advantage of Objective-C:

1. Interoperability with C++ and Objective C++
2. Dynamic features like method swizzling
3. Better support for writing Binary Frameworks.

# Disadvantages of Objective-C:

1. Since Objective-C is built on top of C, it lacks **namespacing**. All classes in an Objective-C application should be globally unique. So to avoid collision there is a convention of prefixing the names of classes. This is the reason we have the **‘NS’** prefix for the class in the **Foundation** Framework and the **‘UI’** prefix for the classes in **UIKit**.
2. Explicit pointers.
3. The ability to send a message on a nil object without crashing and the lack of strict typing lead to bugs that are hard to trace and fix.
4. The language is syntactically verbose and complex, but this is expected given that it is a fairly old language.

Swift is a young language released in 2014. It is designed to be safe and performant with modern syntax and features. Swift was Open-Sourced in December 2015.

# Comparative Advantages of Swift:

1. Swift is safer due to static typing and the use of optionals and optional chaining.
2. Support for namespaces, a clear mutability syntax, functional patterns and concise syntax.
3. Interactive development using Playgrounds.
4. Swift is easier to learn for new programmers, [The official language guide](https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html) by Apple is a great resource.
5. Swift is performant and is finding its place in the server side applications. The advantages of using Swift on the server side are explained in this [talk](https://academy.realm.io/posts/tryswift-chris-robert-end-to-end-application-development-swift-backend/) on [Realm Academy](https://academy.realm.io/) by **Chris Bailey**, where he points out the advantage that Swift has over other frameworks on the server and the cloud. According to him Swift is highly performant and has a low memory footprint which makes it an ideal choice for Server side development.
6. Swift is now stable with its ABI locked down.
7. The Swift standard library code contains around 42.5% of its code in Swift. The split up of the different languages used in the standard library is as shown in this picture below. This Swift code is probably the best Swift code which developers can refer to improve their Swift coding. It is covered very well in this [talk](https://www.youtube.com/watch?v=wB-bi8_rSLs).

8. SwiftUI is a Swift-only declarative framework for making UI for multiple platforms with built-in support for dark mode and other accessibility features. SwiftUI is fully compatible with UIKit, and a SwiftUI View can embed a UIView/NSView, which can also embed a SwiftUI view.

9. The preview of the SwiftUI is available right with Xcode without running the project on the simulator (giving instant visual feedback). Also, the preview device can be switched on the fly by adding a modifier for a preview device, without the hassle of building and running the project on a different simulator.

# Comparative Disadvantages of Swift:

1. Higher compile time.
2. No direct way of using C++ libraries.
3. Module format stability is still not achieved and is required for developers who want to share their code as a binary framework.

Custom String Convertible is a protocol that you can implement on a type such as a class or struct to have a custom representation of the properties that you have declared. By conforming to this protocol, the output of your code instantly becomes more readable, but more importantly, it outputs the information that you desire. Swift has a default representation of the properties, it will list all the properties and values that you have defined. Take at this code example below:

struct Car {

var name: String

var year: Int

var color: String

var new: Bool

}

let dreamCar = Car(name: "Mustang", year: 1995, color: "blue", new: false)

print (dreamCar)//Car(name: "Mustang", year: 1995, color: "blue", new: false)

### What is Deep and Shallow copy?

An instance, whether it’s a value type or a reference type, can be copied in one of the following ways:

#### ****Deep copy —****Duplicates everything

* With a deep copy, any object pointed to by the source is copied and the copy is pointed to by the destination. So two completely separate objects will be created.
* **Collections** — A deep copy of a collection is two collections with all of the elements in the original collection duplicated.
* **Less prone to race conditions** and performs well in a multithreaded environment — changes in one object will have no effect on another object.
* **Value types** are copied deeply.

#### ****Shallow copy —****Duplicates as little as possible

* With a shallow copy, any object pointed to by the source is also pointed to by the destination. So only one object will be created in the memory.
* **Collections** — A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.
* **Faster** — only the reference is copied.
* Copying **reference types** creates a shallow copy.

newArary = oldArray isn't a copy at all. You end up with two pointers pointing to the exact same memory location.

newArray = [NSMutableArray arrayWithArray:oldArray]; is a shallow copy. You end up with two distinct arrays, so if you were to remove or add items from one array, it wouldn't affect the other array. However, the **items** in the two arrays are identical. If the first element of oldArray were an NSMutableDictionary and you added a key to it, you'd see that change on the first element of newArray as well (since those two objects are the same).

To do a deep copy, you would have to make a new array, and each element of the new array would be a deep copy of the corresponding element of the old array. (Yes, that definition is recursive).

First of all, NSArray's don't have a deep copy function. However, you can make a deep copy function by doing the following:

@interface NSArray(deepCopy)

-(NSArray \*) deepCopy;

@end

@implementation

-(NSArray \*) deepCopy {

NSMutableArray \*ret = [NSMutableArray array];

for (id val in self) {

if ([val conformsToProtocol:@protocol(NSCopying)]) {

[ret addObject:[val copy]];

} else {

[ret addObject:val]; }

} return ret; }

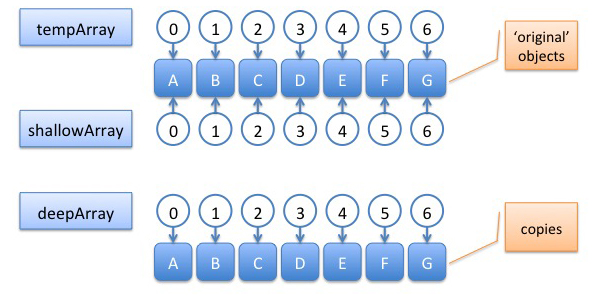
@end

Second of all, newArray = oldArray **does not copy the array**. It simply makes newArray point to the array that oldArray points to.

Third, +arrayWithArray: does a shallow copy of the array, meaning the individual objects are NOT copied.

So what you're basically doing here, is creating three arrays with tempArray and shallowArray pointing to the same objects and deepArray to copies.

Initially it's like this, after instantiating the arrays:

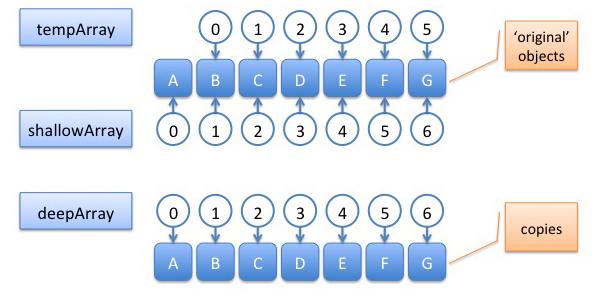


both tempArray and shallowArray point to the same objects. But the arrays themselves are different objects!

And after

[tempArray removeObjectAtIndex:0]

this is the

situation: 

The behavior that you seem to suspect, would be achieved by having shallowArray point to the same object as tempArray:

shallowArray = tempArray;

<https://medium.com/@maddy.lucky4u/copying-in-swift-5b02e9bad00d>

We will cover-

1. ***Copy***
2. ***Deep Copy***
3. ***Shallow Copy***
4. ***Deep Copy Of Reference Types***
5. ***Copy On Write (COW)***

I am going to talk about copying objects in swift. Copying can be two ways — **copy content** and **copy reference**.

Copy Content is also known as **Deep Copy**. **Deep Copy** works with value types. For example Structure, Array, Dictionary, Set, String etc.

Copy Reference is also known as **shallow Copy. shallow Copy**works withreference types. For example Classes.

Property of app delegate if we remove the application from background.

Pass data using segue.

[self performSegueWithIdentifier:@"conduitMulti" sender:sender];

-(void)prepareForSegue:(UIStoryboardSegue \*)segue sender:(id)sender

{

if([segue.identifier isEqualToString:@"conduitMulti"])

{

PT\_MultiListViewController \*multiListViewController = segue.destinationViewController;

multiListViewController.delegate = self;

//populate conduit information in target controller

multiListViewController.ConduitHeader = \_cdtHeader;

multiListViewController.ConduitConfig = \_cdtConfig;

multiListViewController.ConduitData = \_cdtData;

multiListViewController.ConduitButton = \_cdtButton;

//reset selection array

[[[PT\_Singleton sharedStore] getConduitSelections] removeAllObjects];

}

}

NSKeyedArchiver, a concrete subclass of [NSCoder](https://developer.apple.com/documentation/foundation/nscoder?language=objc), provides a way to encode objects (and scalar values) into an architecture-independent suitable for storage in a file. When you archive a set of objects, the archiver writes the class information and instance variables for each object to the archive. The companion class [NSKeyedUnarchiver](https://developer.apple.com/documentation/foundation/nskeyedunarchiver?language=objc) decodes the data in an archive and creates a set of objects equivalent to the original set.

A keyed archive differs from a non-keyed archive in that all the objects and values encoded into the archive have names, or keys. When decoding a non-keyed archive, the decoder must decode values in the same order the original encoder used. When decoding a keyed archive, the decoder requests values by name, meaning it can decode values out of sequence or not at all. Keyed archives, therefore, provide better support for forward and backward compatibility.

The keys given to encoded values must be unique only within the scope of the currently-encoding object. A keyed archive is hierarchical, so the keys used by object A to encode its instance variables don’t conflict with the keys used by object B. This is true even if A and B are instances of the same class. Within a single object, however, the keys used by a subclass can conflict with keys used in its superclasses.

An [NSArchiver](https://developer.apple.com/documentation/foundation/nsarchiver?language=objc) object can write the archive data to a file or to a mutable-data object (an instance of [NSMutableData](https://developer.apple.com/documentation/foundation/nsmutabledata?language=objc)) that you provide.

During debugging time also locked of device.

**What is epoch time?**

The **Unix epoch** (or **Unix time** or **POSIX time** or **Unix timestamp**) is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT), not counting leap seconds (in ISO 8601: 1970-01-01T00:00:00Z). Literally speaking the epoch is Unix time 0 (midnight 1/1/1970), but 'epoch' is often used as a synonym for Unix time. Some systems store epoch dates as a signed 32-bit integer, which might cause problems on January 19, 2038 (known as the Year 2038 problem or Y2038). The converter on this page converts timestamps in seconds (10-digit), milliseconds (13-digit) and microseconds (16-digit) to readable dates.

Keeping a codebase consistent and maintainable in a project with a team of devs sometimes may be very hard, different conventions and styles, plus different levels of experience with the language across devs may result in most of the times in an app very difficult to debug and mostly very hard to understand for new developer joining the team.

While investigating for a CI for one project that will help my team and I avoid bugs and keep our code clean I found [Swiftlint](https://github.com/realm/SwiftLint), A tool to enforce Swift style and conventions, its great it follows Swift guidelines, its maintained by the Realm team and its open source which means its get updated constantly by the Swift community.

<https://medium.com/cocoaacademymag/swiftlint-introduction-tutorial-cd692d41dda3>

<https://github.com/realm/SwiftLint>

A tool to enforce Swift style and conventions.

# [Defer usage in Swift](https://www.avanderlee.com/swift/defer-usage-swift/)

Although the defer keyword was already introduced in Swift 2.0, it’s still quite uncommon to use it in projects. Its usage can be hard to understand, but using it can improve your code a lot in some places.

## How does it work?

A defer statement is used for executing code just before transferring program control outside of the scope that the statement appears in.

func updateImage() {

defer { print("Did update image")

}

print("Will update image")

imageView.image = updatedImage

} // Will update Image // Did update image

## Order of execution with multiple defer statements

If multiple statements appear in the same scope, the order they appear is the reverse of the order they are executed. The last defined statement is the first to be executed which is demonstrated by the following example by printing numbers in logical order.

func printStringNumbers() {

defer { print("1") }

defer { print("2") }

defer { print("3") }

print("4")

} /// Prints 4, 3, 2, 1

## A common use case

The most common use case seen around is opening and closing a context within a scope, for example when handling access to files. A FileHandle requires to be closed once the access has been finished. You can benefit from the defer statement to ensure you don’t forget to do this.

func writeFile() {

let file: FileHandle? = FileHandle(forReadingAtPath: filepath)

defer { file?.closeFile()

} // Write changes to the file }

## Ensuring results

A more advanced usage of the statement is by ensuring a result value to be returned in a completion callback. This can be very handy as it’s easy to forget to trigger this callback.

func getData(completion: (\_ result: Result<String>) -> Void) {

var result: Result<String>?

defer { guard let result = result else { fatalError("We should always end with a result") } completion(result) } // Generate the result.. }

The statement makes sure to execute the completion handler at all times and verifies the result value. Whenever the result value is nil, the fatalError is thrown and the application fails.

let str = "Hello, world!"

let reversed = String(str.reversed())

print(reversed)

# What is ABI?

ABI stands for Application Binary Interface. At runtime, Swift program binaries interact with other libraries through an ABI. It defines many low level details for binary entities like how to call functions, how their data is represented in memory, where metadata is and how to access it.

# What is ABI Stability?

*ABI stability means locking down the ABI to the point that future compiler versions can produce binaries conforming to the stable ABI. It enables binary compatibility between applications and libraries compiled with different Swift versions.*

Earlier, Swift was not ABI stable, so each binary was bundling its own version of the Swift Dynamic Library. If you open an .ipa you can find swift standard libraries (.dylib) in SwiftSupport or Frameworks.

Suppose Apple music is using Swift 3.2, so it bundles Swift 3.1 Dynamic Library inside but the Notes app is using Swift 4.0, so it bundles Swift 4.0 and it’s 4.0 ABI.

Now Swift 5 is ABI Stable, Swift will be embeded within the iOS Operating System and it’s ABI will be compatible with every version of Swift. i.e Apple Music is using Swift 5.0, but Notes app is using Swift 5.2, and both are consuming Swift ABI embedded in the Operating System.

# Why does ABI Stability matter?

* **Reduced Bundle size:** Size of your application will decrease because you will no longer have to include the Swift standard library in your Frameworks folder.
* **Source compatibility:**Newer compilers can compile code written in an older version of Swift. This aims to reduce the migration pain that Swift developers face when migrating to a newer Swift version.
* **Less frequent language changes:** Less changes leads to less efforts in migration.
* **Binary framework & runtime compatibility:**It enables the distribution of frameworks in a binary form that works across multiple Swift versions. Binary framework include both swift module file and shared library. Module format stability stabilizes the module file, which is the compiler’s representation of the public interfaces of a framework. Module format stability will take time to exist (May not make it in time). Developers can create Pre-compiled Frameworks in Swift because they don’t need to bundle the Swift standard library into their framework.

Swift is a fast, safe and a fun language to code in with full stack potential and a great community support. It is about **2.6** times faster than Objective-C according to [Apple](https://www.apple.com/swift/), however, some studies indicate that the difference is not that [dramatic](https://www.thedroidsonroids.com/blog/ios/swift-vs-obj-c-performance-comparision). Swift code is easier to maintain as there are no separate interface and implementation files, the syntax is shorter and the language supports [dynamic frameworks](https://www.ca.com/en/blog-developers/dynamic-versus-static-framework-in-ios.html).

The language has grown significantly and has been adopted by a large number of developers. It is the **6th most loved language** according to **StackOverflow** [Developer Survey 2018](https://insights.stackoverflow.com/survey/2018/#most-loved-dreaded-and-wanted). For a language released in just 2014, the adoption rate is phenomenal.

The ABI Stability [Manifesto](https://github.com/apple/swift/blob/master/docs/ABIStabilityManifesto.md) states they have the goal of accomplishing:

1. **Source Compatibility,**which means that newer compilers can compile code written in an older version of Swift. This removes the version-lock currently in Swift.
2. **Binary framework & runtime compatibility,**which enables the distribution of frameworks in a binary form that works across multiple Swift versions. Binary framework compatibility will be achieved by **module format stability** which stabilizes the module file, which is the compiler’s representation of the public interfaces of a framework and **ABI stability** enables binary compatibility between applications and libraries compiled with different Swift versions.

*At runtime Swift program binaries interact with other libraries and components.****Application Binary Interface****is the specification to which independently compiled binary entities must conform to be linked together and executed. These binary entities must agree on many low-level details like how to call functions, data representation in memory, and even where their metadata is and how to access it.*

# What is ABI Stability?

**ABI stability** means locking down the ABI to the point that future compiler versions can produce binaries conforming to the stable **ABI**. Once an **ABI** is stable, it tends to persist for the rest of the platform’s lifetime.

ABI stability only affects invariants of externally visible public interfaces and symbols. For example, future compilers are free to change the calling conventions for internal function calls so long as the public interfaces are preserved.

# Prerequisites of ABI Stability

1. Types, such as structs and classes, must have a defined in-memory layout for instances of that type and share the same layout conventions.
2. Type metadata is used extensively by Swift programs. This metadata must either have a defined memory layout or have a set of defined APIs for querying the metadata of a type.
3. Every exported or external symbol in a library needs a unique name upon which binary entities can agree. Swift provides function overloading and contextual namespaces (such as modules and types), which means that any name in source code might not be globally unique. A unique name is produced through a technique called name mangling.
4. Functions must adhere to calling conventions, which entails such things as the layout of the call stack, what registers are preserved, and ownership conventions.
5. Swift ships with a runtime library which handles things like dynamic casting, reference counting, reflection, etc. Compiled Swift programs make external calls out to this runtime. Thus, Swift runtime API is Swift ABI.
6. Swift ships with a standard library that defines many common types, structures, and operations on these. For a shipped standard library to work with applications written in different versions of Swift, it must expose a stable API. Thus, Swift Standard Library API is Swift ABI, as well as the layout of many of the types it defines.

*All these tasks have been accomplished by the Swift core team but they have not yet been released on GitHub. Looking at the*[*status*](https://swift.org/abi-stability/)*of the tasks we can safely expect the next major release of Swift to be ABI stable.*

# Advantages of ABI Stability

1. So once Swift is declared to be ABI stable, the code written from this point on would be compatible with the newer versions of the language and the developer won’t have to update all the external dependencies of the project while migrating to a new version of Swift.
2. The library author can supply his framework as a binary framework once module format stability is achieved.
3. The application bundle size would decrease as the stable Swift runtime could then be incorporated within the OS.
4. The language would keep evolving but the changes to the **ABI**from that point on would be additive. The **ABI-additive** changes may be taken advantage of when the minimum targeted Swift version supports them, since ABI stability locks only the externally visible public interfaces and symbols. The newer compilers can make internal changes that may improve efficiency.

Apple made dynamic framework to reduce the launch time of apps and their memory footprints. Dynamic framework also accepts assets inside the framework file, so you don’t have to create a separate bundle file to host files like images or screens (.nib).

Here is how Apple describes their dynamic framework:

“Two important factors that determine the performance of apps are their launch times and their memory footprints. Reducing the size of an app’s executable file and minimizing its use of memory once it’s launched make the app launch faster and use less memory once it’s launched. Using dynamic libraries instead of static libraries reduces the executable file size of an app. They also allow apps to delay loading libraries with special functionality only when they’re needed instead of at launch time. This feature contributes further to reduced launch times and efficient memory use.”

**Static framework**

The following graphic shows how static libraries are all loaded into the address space for later use. As such, they become part of the executable, and are statically linked to client apps. This architecture makes apps slower to load and run.

**Dynamic framework**

With dynamic libraries, the app loads code into its address space when it’s actually needed, either at launch time or at runtime. The libraries are not part of the executable file. This decreases the memory footprint for your app.

***More benefits***

* Dynamic framework is exposed in the Xcode project build settings.  
  The only code change needed is the location of any assets that will reside inside the framework file. Your app is improved, even without recompiling.
* All system libraries in OS X are dynamic libraries, so app written for OS X automatically benefit. (This is how apps that use Carbon or Cocoa technologies benefit from improvements to OS X).
* Unlike static libraries, dynamic libraries are initialized when they are loaded; so they can perform clean-up tasks when client apps terminate normally. For details, see [Module Initializers and Finalizers](https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/DynamicLibraries/100-Articles/DynamicLibraryDesignGuidelines.html#//apple_ref/doc/uid/TP40002013-SW17).”

#### WHEN

Whenever we make changes to the SDK architecture, we always consider:

* Customer experience
* Technology
* Backwards compatibility
* Supporting the latest OS releases

Clearly, the dynamic frameworks had some advantages over static libraries. When the dynamic framework was announced by Apple, we could have moved to dynamic it right away. But we believed that would limit our audience because dynamic framework is only available for iOS 8 and above. So, to maintain backward compatibility, we migrated from the old .a library to the new .framework model, but kept it as static.

We then focused on enhancing the customer experience by creating Xcode templates with predefined code for starting the SDK, folder structure, and project build settings to make getting started with a new project easier.

Another important improvement was to make our iOS Mobile SDK available in CocoaPods ([https://cocoapods.org](https://cocoapods.org/)). (In the following section, you will learn how important having CocoaPods in place is for dynamic frameworks.)

Timing is everything. Recently, our customers using iOS 7 updated to iOS 9 or 10. Because iOS Mobile SDK always supports the latest two versions of the OS (9 and 10 as of this blog), now is the perfect time to move to dynamic framework.

#### HOW

There is one side effect of dynamic framework that makes the developer experience less than optimal.

Xcode produces two files whenever the framework is built:

* Simulator architecture file
* Device file

Apple suggests that developers deliver the two files to customers. But that means you need to add extra settings, or switch files whenever you want to test on the device or simulator.

To make a better experience, we built **one .framework file** with both simulator and device files. So you don’t waste time adding extra settings to the project, or have to switch files whenever you test the app in a different platform.

**So what do you need to do to use our dynamic frameworks?** Add them to your project via the Embedded Binaries’ section (project general tab) and before deploying your app to the Apple store, just add a custom script (copy and paste from the documentation in our website) to your app’s Build Phases, build it, and the simulator architecture will be removed from the framework as Apple requires.

**For the most hands off experience**, we highly recommend that you use the Mobile SDK with CocoaPods. All of the build settings and script is automatically handled by CocoaPods, *so you don’t need to do anything at all*.

# SQL Data Types: BINARY LARGE OBJECT

Technically a **Binary Large Object**, a BLOB is an **object data type**, meaning it refers to an object. Unlike a character or integer data type, the object data type only contains a pointer or reference to the value of the object. A BLOB can hold a very large block of data, anything from documents to images to videos. You could store your great American novel in a BLOB if you really wanted to (as a file).

A BLOB is really the object's agent, or handler: The database manager shouldn't need to know what's in the file or or to work it it, but it can still be a part of the database.

## MySQL

MySQL supports four BLOB types:

1. TINYBLOB
2. BLOB
3. MEDIUMBLOB
4. LONGBLOB

These are all BLOBs, but they differ in how large they can be. TINYBLOB is only about 256 bytes, and LONGBLOB is 4 gigabytes! Why would we even create a TINYBLOB, since at 256 bytes, it can hardly be considered a large object? For MySQL, the focus is on the object: you could still store small text files in the database, as opposed to having to copy/paste the data from the text into another field. If you want to use a BLOB in MySQL, use the **LONGBLOB** option, as it supports a larger file size.

In order to save some memory and processing overhead, MySQL stores the BLOB information in a separate memory area than the normal table-processing memory.

You can access the delegate like this:

MainClass \*appDelegate = (MainClass \*)[[UIApplication sharedApplication] delegate];

Replace MainClass with the name of your application class.

Then, provided you have a property for the other view controller, you can call something like:

[appDelegate.viewController someMethod];

New add-ons are now available: **increase the maximum upload size** of your account by **500 MB** or **1 GB** when the current maximum of your Starter or Premium plan is not enough.

SQLite data type is an attribute that specifies the type of data of any object. Each column, variable and expression has related data type in SQLite.

You would use these data types while creating your tables. SQLite uses a more general dynamic type system. In SQLite, the datatype of a value is associated with the value itself, not with its container.

## SQLite Storage Classes

Each value stored in an SQLite database has one of the following storage classes −

|  |  |
| --- | --- |
| **Sr.No.** | **Storage Class & Description** |
| 1 | **NULL**  The value is a NULL value. |
| 2 | **INTEGER**  The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value. |
| 3 | **REAL**  The value is a floating point value, stored as an 8-byte IEEE floating point number. |
| 4 | **TEXT**  The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE) |
| 5 | **BLOB**  The value is a blob of data, stored exactly as it was input. |

SQLite storage class is slightly more general than a datatype. The INTEGER storage class, for example, includes 6 different integer datatypes of different lengths.

## SQLite Affinity Type

SQLite supports the concept of **type affinity** on columns. Any column can still store any type of data but the preferred storage class for a column is called its **affinity**. Each table column in an SQLite3 database is assigned one of the following type affinities −

|  |  |
| --- | --- |
| **Sr.No.** | **Affinity & Description** |
| 1 | **TEXT**  This column stores all data using storage classes NULL, TEXT or BLOB. |
| 2 | **NUMERIC**  This column may contain values using all five storage classes. |
| 3 | **INTEGER**  Behaves the same as a column with NUMERIC affinity, with an exception in a CAST expression. |
| 4 | **REAL**  Behaves like a column with NUMERIC affinity except that it forces integer values into floating point representation. |
| 5 | **NONE**  A column with affinity NONE does not prefer one storage class over another and no attempt is made to coerce data from one storage class into another. |

## SQLite Affinity and Type Names

Following table lists down various data type names which can be used while creating SQLite3 tables with the corresponding applied affinity.

|  |  |
| --- | --- |
| **Data Type** | **Affinity** |
| * INT * INTEGER * TINYINT * SMALLINT * MEDIUMINT * BIGINT * UNSIGNED BIG INT * INT2 * INT8 | INTEGER |
| * CHARACTER(20) * VARCHAR(255) * VARYING CHARACTER(255) * NCHAR(55) * NATIVE CHARACTER(70) * NVARCHAR(100) * TEXT * CLOB | TEXT |
| * BLOB * no datatype specified | NONE |
| * REAL * DOUBLE * DOUBLE PRECISION * FLOAT | REAL |
| * NUMERIC * DECIMAL(10,5) * BOOLEAN * DATE * DATETIME | NUMERIC |

## Boolean Datatype

SQLite does not have a separate Boolean storage class. Instead, Boolean values are stored as integers 0 (false) and 1 (true).

## Date and Time Datatype

SQLite does not have a separate storage class for storing dates and/or times, but SQLite is capable of storing dates and times as TEXT, REAL or INTEGER values.

|  |  |
| --- | --- |
| **Sr.No.** | **Storage Class & Date Formate** |
| 1 | **TEXT**  A date in a format like "YYYY-MM-DD HH:MM:SS.SSS" |
| 2 | **REAL**  The number of days since noon in Greenwich on November 24, 4714 B.C. |
| 3 | **INTEGER**  The number of seconds since 1970-01-01 00:00:00 UTC |

You can choose to store dates and times in any of these formats and freely convert between formats using the built-in date and time functions.

Go to Settings > General > About. To the right of Model, you'll see the part number. To see the model number, tap the part number.

<https://www.raywenderlich.com/960290-ios-unit-testing-and-ui-testing-tutorial>

VMware, Inc. is a publicly traded software company listed on the NYSE under stock ticker VMW. Dell Technologies is a majority share holder. VMware provides cloud computing and virtualization software and services. It was one of the first commercially successful companies to virtualize the x86 architecture.

 iOS will automatically remove objects from the cache if the device is running low on memory.

Helpfully, if the system does encounter memory pressure NSCache will automatically start to remove items without you knowing about it, which means you won't get a memory warning unless even more RAM needs to be cleared. It will also remove items intelligently, trying to keep as much cached as possible.

When you use a compass app on your smartphone, it somehow knows which direction the phone is pointing. With stargazing apps, it somehow knows where in the sky you’re looking to properly display constellations. Smartphones and other mobile technology identify their orientation through the use of an accelerator, a small device made up of axis-based motion sensing.

The motion sensors in accelerometers can even be used to [detect earthquakes](https://www.livescience.com/30656-detect-earthquake-with-home-computer.html), and may by used in [medical devices](https://www.livescience.com/39552-mems-phone-sensors-human-body.html) such as bionic limbs and other artificial body parts. Several devices, part of the [quantified self movement](https://www.livescience.com/39185-quantified-self-movement.html), use accelerometers.

An accelerometer is an electromechanical device used to measure acceleration forces. Such forces may be static, like the continuous force of gravity or, as is the case with many mobile devices, dynamic to sense movement or vibrations.

Acceleration is the measurement of the change in velocity, or speed divided by time. For example, a car accelerating from a standstill to 60 mph in six seconds is determined to have an acceleration of 10 mph per second (60 divided by 6).

## The purpose of the accelerometer

The application of accelerometers extends to multiple disciplines, both academic and consumer-driven. For example, accelerometers in laptops protect hard drives from damage. If the laptop were to suddenly drop while in use, the accelerometer would detect the sudden free fall and immediately turn off the hard drive to avoid hitting the reading heads into the hard drive platter. Without this, the two would strike and cause scratches to the platter for extensive file and reading damage. Accelerometers are likewise used in cars as the industry method way of detecting car crashes and deploying airbags almost instantaneously.

In another example, a dynamic accelerometer measures gravitational pull to determine the angle at which a device is tilted with respect to the Earth. By sensing the amount of acceleration, users analyze how the device is moving.

Accelerometers allow the user to understand the surroundings of an item better. With this small device, you can determine if an object is moving uphill, whether it will fall over if it tilts any more, or whether it’s flying horizontally or angling downward. For example, smartphones rotate their display between portrait and landscape mode depending on how you tilt the phone.

## How they work

An accelerator looks like a simple circuit for some larger electronic device. Despite its humble appearance, the accelerometer consists of many different parts and works in many ways, two of which are the piezoelectric effect and the capacitance sensor. The piezoelectric effect is the most common form of accelerometer and uses microscopic crystal structures that become stressed due to accelerative forces. These crystals create a voltage from the stress, and the accelerometer interprets the voltage to determine velocity and orientation.

The capacitance accelerometer senses changes in capacitance between microstructures located next to the device. If an accelerative force moves one of these structures, the capacitance will change and the accelerometer will translate that capacitance to voltage for interpretation.

Accelerometers are made up of many different components, and can be purchased as a separate device. Analog and digital displays are available, though for most technology devices, these components are integrated into the main technology and accessed using the governing software or operating system.

Typical accelerometers are made up of multiple axes, two to determine most two-dimensional movement with the option of a third for 3D positioning. Most smartphones typically make use of three-axis models, whereas cars simply use only a two-axis to determine the moment of impact. The sensitivity of these devices is quite high as they’re intended to measure even very minute shifts in acceleration. The more sensitive the accelerometer, the more easily it can measure acceleration.

Accelerometers, while actively used in many electronics in today’s world, are also available for use in custom projects. Whether you’re an engineer or tech geek, the accelerometer plays a very active role in a wide range of functionalities. In many cases you may not notice the presence of this simple sensor, but odds are you may already be using a device with it.

## What is a gyroscope?

A gyroscope is a device that uses Earth's gravity to help determine orientation. Its design consists of a freely-rotating disk called a rotor, mounted onto a spinning axis in the center of a larger and more stable wheel. As the axis turns, the rotor remains stationary to indicate the central gravitational pull, and thus which way is "down."

"One typical type of gyroscope is made by suspending a relatively massive rotor inside three rings called gimbals," according to a study guide by [Georgia State University](http://hyperphysics.phy-astr.gsu.edu/hbase/gyr.html). "Mounting each of these rotors on high quality bearing surfaces insures that very little torque can be exerted on the inside rotor."

# Persistent and ephemeral storage

For each node in the cluster, you can configure both a persistent storage and an ephemeral storage path.

* Persistent storage is mandatory; it is used by the cluster to store information that needs to persist even if a shard or a node fails, e.g server logs, configurations, files. For example, if you choose any type of persistence for a database, then the persistence information is stored in this location.
* Ephemeral storage is optional. If defined, it is used by the cluster to store information that does not need to persist. This aids in optimization and helps to reduce the load on the persistent storage.

# GameplayKit

Architect and organize your game logic. Incorporate common gameplay behaviors such as random number generation, artificial intelligence, pathfinding, and agent behavior.

GameplayKit is an object-oriented framework that provides foundational tools and technologies for building games. GameplayKit includes tools for designing games with functional, reusable architecture, as well as technologies for building and enhancing gameplay features such as character movement and opponent behavior.

# SpriteKit

Add high-performance 2D content with smooth animations to your app, or create a game with a high-level set of 2D game-based tools.

SpriteKit is a general-purpose framework for drawing shapes, particles, text, images, and video in two dimensions. It leverages Metal to achieve high-performance rendering, while offering a simple programming interface to make it easy to create games and other graphics-intensive apps. Using a rich set of animations and physics behaviors, you can quickly add life to your visual elements and gracefully transition between screens.

SpriteKit is supported on iOS, macOS, tvOS, and watchOS, and it integrates well with frameworks such as GameplayKit and SceneKit.

ObjectMapper is a framework written in Swift that makes it easy for you to convert your model objects (classes and structs) to and from JSON.

<https://github.com/tristanhimmelman/ObjectMapper>

<https://github.com/tristanhimmelman/ObjectMapper#objectmapper--realm>

<https://github.com/John-Lluch/SWRevealViewController>

# SWRevealViewController

A UIViewController subclass for revealing a rear (left and/or right) view controller behind a front controller, inspired by the Facebook app, done right!

Apple Configurator is a free Mac [OS X](https://whatis.techtarget.com/definition/OS-X) tool for configuring and deploying [iOS](https://searchmobilecomputing.techtarget.com/definition/iOS) devices in the enterprise by using a physical [USB](https://whatis.techtarget.com/definition/USB-Universal-Serial-Bus-USB-30-SuperSpeed-USB) connection.

The main purpose of the Apple Configurator [app](https://searchmobilecomputing.techtarget.com/definition/app) is to allow IT administrators configure settings on iPhones and iPads before rolling them out to end users. Administrators can create configuration profiles, import existing profiles from the [iPhone Configuration Utility](https://searchmobilecomputing.techtarget.com/definition/iPhone-Configuration-Utility), install specific operating system versions and enforce mobile device security policies.

The Apple Configurator tool also provides some basic mobile device management ([MDM](https://searchmobilecomputing.techtarget.com/definition/mobile-device-management)) capabilities, including device [wipe](https://whatis.techtarget.com/definition/wipe).