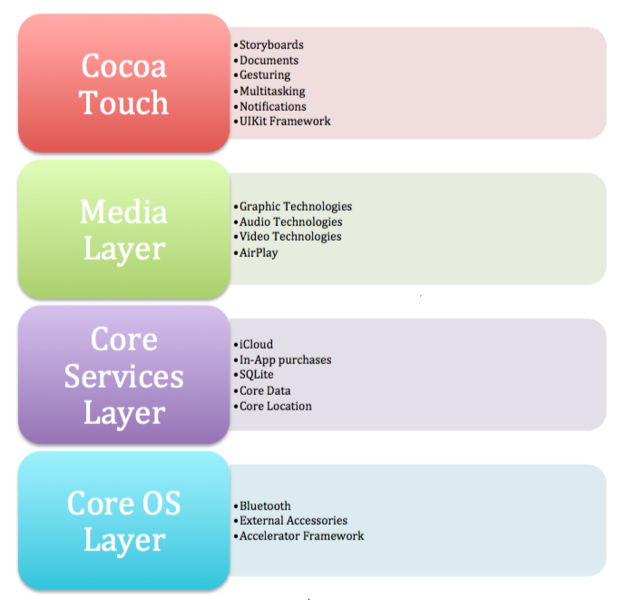
**iOS Technology Overview**

Whenever we develop an iPhone app apple does not directly allow us to access its hardware. All hardware interaction takes place through number of software layers that acts as intermediaries between the application code and device hardware.

These layers all together called as operating system. In case of iPhone, its operating system is known as iOS.

iOS consists of number of software layers, each of which provides framework for development of applications.

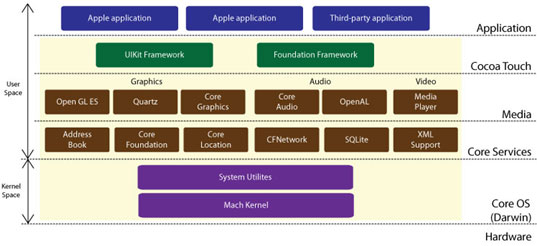
iOS is a layered technology. It consists of following layers.

****

Each operating system provides increasing level of abstraction away from complexity of working with the hardware.

In general, the higher level of layer you program to, the less effort and fewer lines of code you have write to achieve your objective. Less code you have to write the less opportunity you have to introduce bugs.

Now that we have identified various numbers of layers comprise in iOS, we can now look in more detail at the services provided by each layer and the corresponding frameworks that make those services provided to us as application developers.



**Cocoa Touch Layer**

Cocoa touch layer sits at the top of iOS stack and contains the frameworks that are most commonly used.

Cocoa touch framework is primarily written in the Objective-C, is based on the standard Mac OS X cocoa API (for apple desktops and laptop computers) and has been extended and modified to meet the needs of iPhone. The cocoa touch API provides following frameworks for iPhone app development.

### Address Book UI Framework

### EventKit UI Framework

### GameKit Framework

### iAd Framework

### MapKit Framework

### Message UI Framework

### Notification Center Framework

### PushKit Framework

### Twitter Framework

### UIKit Framework

The following section describes some of the key technologies available in cocoa touch layer.

App Extensions

### Handoff

### Document Picker

### AirDrop

TextKit

UIKit Dynamics

Multitasking

Auto Layout

Storyboards

UI State Preservation

Apple Push Notification Service

Local Notifications

Gesture Recognizers

**Media Layer**

Media layer provides the audio, video, animation and graphics to develop your iPhone app. The technologies present in the layer help you to build app that look and sound great.

**Media Layer Technologies**

Graphics technologies

Video technologies

**Airplay**

AirPlay lets your app to stream video to apple TV and audio to third party speaker and receivers.

AirPlay support is built in for numerous frameworks

* UIKit framework
* Media Player framework
* AV Foundation framework

You don’t need anything special to support it. Any content you play using these frameworks are automatically eligible for AirPlay distribution. When the user tries to play content using airplay, it is routed automatically by the system.

The playback classes of the Media Player framework automatically support AirPlay. You can also display Now Playing content on a connected Apple TV using AirPlay.

Use the [AVPlayer](https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVPlayer_Class/index.html" \l "//apple_ref/occ/cl/AVPlayer" \t "_self) class in AV Foundation to manage your app’s audio and video content. This class supports streaming its content via AirPlay when enabled by the user.

For web-based audio and video, you can allow that content to be played over AirPlay by including an embed tag with the airplay attribute. The [UIWebView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/index.html" \l "//apple_ref/occ/cl/UIWebView" \t "_self) class also supports media playback using AirPlay

**Media Layer Framework**

**Core Video Framework (CoreVideo.framework)**

It is a new framework introduced with iOS4, which provides buffering support for the core media framework. While this may be utilized by the application developers, it is typically not necessary to use this framework.

**Core Text Framework (CoreText.framework)**

The iOS Core Text is a C-based API designed to handling of advanced text layouts and fonts rendering the requirements.

**Image I/O Framework (ImageIO.framework)**

Image I/O Framework provides interface for importing and exporting of image data and image metadata. This framework makes use of the Core Graphics data types and functions and supports all of the standard image types available in iOS.

**Assets Library Framework (AssetsLibrary.framework)**

Asset Library framework provides a mechanism for locating and retrieving video and photo files located on iPhone devices. In addition to accessing existing photo and videos, It also provides functionality to save new video and photos to the standard photo device album.

**Media** **Player Framework (MediaPlayer.framework)**

Media player framework provides high level support for playing audio and videos on your device. You can use this framework for the following purpose.

* Play video to a user’s screen or to another device over AirPlay. You can play this video full screen or in a resizable view.
* Access the user’s iTunes music library. You can play music tracks and playlists, search for songs, and present a media picker interface to the user.
* Configure and manage movie playback.
* Display Now Playing information in the lock screen and App Switcher. You can also display this information on an Apple TV when content is delivered via AirPlay.
* Detect when video is being streamed over AirPlay.

**Audio Technology**

**AV Foundation framework (AVFoundation.framework)**

It is an Objective C based framework designed to allow the play back, recording management of audio content.

**Media player framework (MediaPlayer.framework)**

Media player framework provides easy access to the user’s iTunes library, support for playing tracks and playlist. This framework is also used to integrate audio into your app quickly and when you don’t need to control the play back behavior.

iOS supports many industry standard and apple specific formats, including following formats

* AAC
* Apple Lossless (ALAC)
* A-law
* IMA/ADPCM (IMA4)
* Linear PCM
* µ-law
* DVI/Intel IMA ADPCM
* Microsoft GSM 6.10
* AES3-2003

**The App Life Cycle**

Applications play a sophisticated interplay between the system framework & custom code. The System Framework it provides the basic infrastructure that the apps need to run. It also provides the code that is required to customize the infrastructure. Finally, it provides the look & feel to the app that exactly you want to.

The iOS Framework relies on the design patterns as in MVC (Mobile View Controller) & delegation in their implementation. In order to create the successful app, it is pivotal to understand the design patterns.

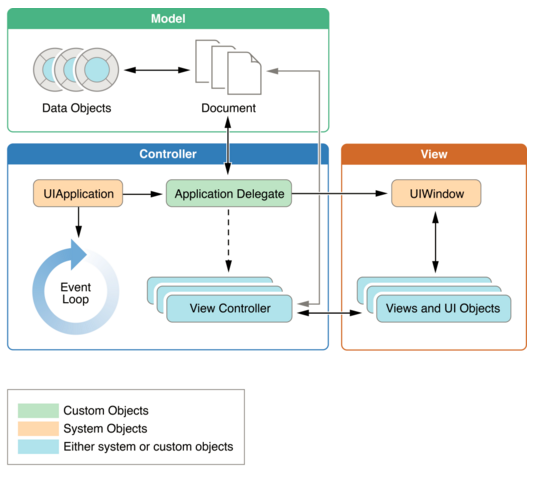
**The Structure of an App**

The functions **UIApplicationMain** handles the procedure by creating the core objects of the app and for startup, this function sets up so many key objects.

The heart of every iOS app is the **UIApplication** object that works to facilitate the interaction between the system & other objects in the app.

The iOS apps use **MVC (Modelviewcontroller)** architecture. This kind of pattern separates the app's data & business logic from the visual presentation of the data. The architecture is pivotal in order to create an app that can run on 'diverse devices' with 'different screen sizes'.

**Below diagram shows object commonly found in most apps**

****

**The Role of objects in iOS App**

**UIApplication Object**

It manages the event loop & high level app behaviors. Also, it reports key app transitions & some special events. Use the UIApplication object as is that is, without sub classing.

**App Delegate Object**

This is the heart of the custom code. The object tasks with UIApplication object in order to handle app initialization, state transitions & many high level app events. App Delegate object uses to set up the application's initial data structures.

**Documents & Data Model Objects**

The Data Model Objects store the content of the app. Apps can also be used documents objects in order to manage few or all data model objects. The documents objects are not needed but offering the convenient method to group data, which belongs to the single file or file package.

**View Controller Objects**

It manages the presentation of the content of your app. A view controller manages a single view & the collection of sub views. For all view controller objects, the UIViewController class is the base class, providing default functionality for loading views, presenting & rotating them in response to device rotations.

**UIWindow Object**

It coordinates the presentation of one or more views on screen. Usually, most apps have only one window that represents content on the main screen. But, apps may have an additional window for content that is displayed on an external display.

**View, Control & Layer Objects**

Views & controls render the visual presentation of app's content. A view is an object that draws the content in a designed rectangular area. The Control objects are specialized type of view responsible to events within the area.

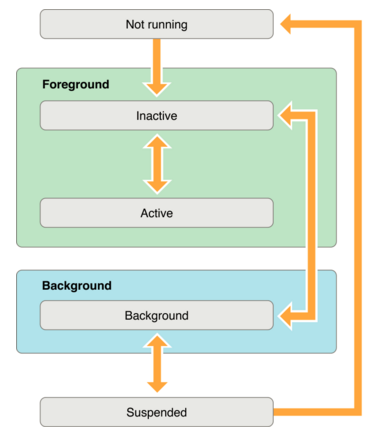
**Execution States for Apps**

At any given moment, your app is in one of the states listed. The system moves your app from state to state in response to actions happening throughout the system. For example, when the user presses the Home button, a phone call comes in, or any of several other interruptions occurs, the currently running apps change state in response.

* Not running
* Inactive
* Active
* Background
* suspended

|  |  |
| --- | --- |
| Not running | The app has not been launched or was running but was terminated by the system. |
| Inactive | The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state. |
| Active | The app is running in the foreground and is receiving events. This is the normal mode for foreground apps. |
| Background | The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state. For information about how to execute code while in the background, see [Background Execution](https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html#//apple_ref/doc/uid/TP40007072-CH4-SW1). |
| Suspended | The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code.  When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app. |

**State changes in an iOS app**



State transitions are takes place by corresponding call to the method of your app delegate object. These methods are responsible for state change in an app in appropriate way, methods are listed below

**application:willFinishLaunchingWithOptions**

This is the app’s first method executes at launch time.

Syntax

- (BOOL)application:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

willFinishLaunchingWithOptions:([NSDictionary](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/index.html#//apple_ref/doc/c_ref/NSDictionary) \*)launchOptions

Use this method to initialize your app and prepare it to run. This method is called after your app has been launched and its main storyboard or nib file has been loaded, but before your app’s state has been restored. At the time this method is called.

**application:didFinishLaunchingWithOptions**

This method allows you to perform any final initialization before your app is displayed to the user.

Syntax

-(BOOL)application:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application  
didFinishLaunchingWithOptions:([NSDictionary](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSDictionary_Class/index.html#//apple_ref/doc/c_ref/NSDictionary) \*)launchOptions

Use this method to complete your app’s initialization and make any final tweaks. This method is called after state restoration has occurred but before your app’s window and other UI have been presented. At some point after this method returns, the system calls another of your app delegate’s methods to move the app to the active (foreground) state or the background state.

**applicationDidBecomeActive**

Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.

Syntax

- (void)applicationDidBecomeActive:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

This method is called to let your app know that it moved from the inactive to active state. This can occur because your app was launched by the user or the system. Apps can also return to the active state if the user chooses to ignore an interruption that sent the app temporarily to the inactive state.

**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.

Syntax

- (void)applicationWillResignActive:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

This method is called to let your app know that it is about to move from the active to inactive state. This can occur for certain types of temporary interruptions or when the user quits the app and it begins the transition to the background state. An app in the inactive state continues to run but does not dispatch incoming events to responders

**applicationDidEnterBackground**

Lets you know that your app is now running in the background and may be suspended at any time.

Syntax

- (void)applicationDidEnterBackground:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

Use this method to release shared resources, invalidate timers, and store enough app state information to restore your app to its current state in case it is terminated later. You should also disable updates to your app’s user interface and avoid using some types of shared system resources. It is also imperative that you avoid using OpenGL ES in the background.

**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

Syntax

- (void)applicationWillEnterForeground:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

this method is called as part of the transition from the background to the active state. You can use this method to undo many of the changes you made to your app upon entering the background. The call to this method is invariably followed by a call to the [applicationDidBecomeActive:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html" \l "//apple_ref/occ/intfm/UIApplicationDelegate/applicationDidBecomeActive:)method, which then moves the app from the inactive to the active state.

**applicationWillTerminate**

Lets you know that your app is being terminated. This method is not called if your app is suspended.

Syntax

- (void)applicationWillTerminate:([UIApplication](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplication_Class/index.html#//apple_ref/doc/c_ref/UIApplication) \*)application

This method lets your app know that it is about to be terminated and purged from memory entirely. You should use this method to perform any final clean-up tasks for your app, such as freeing shared resources, saving user data, and invalidating timers. Your implementation of this method has approximately five seconds to perform any tasks and return. If the method does not return before time expires, the system may kill the process altogether.