# Expected app behaviors

* Every app must have the following set of resources and metadata about your app so that it can be displayed properly on iOS devices:
* **An information property-list file:** The Info.plist file contains metadata about your app, which the system uses to interact with your app. Xcode creates this file for you automatically based on your project’s configuration and settings. If you want to view or modify the contents of this file directly, you can do so from the Info tab of your project.
* **A declaration of the app’s required capabilities:** Every app must declare the hardware capabilities or features that it requires to run. The App Store uses this information to determine whether or not a user can run your app on a specific device. he Required device capabilities entry corresponds to the UIRequiredDeviceCapabilities key in your app’s Info.plist file.
* **One or more icons:** The system displays your app icon on the home screen of a user’s device. The system may also use other versions of your icon in the Settings app or when displaying the results of a search.
* **One or more launch images:** When an app is launched, the system displays a temporary image until the app is able to present its user interface. This temporary image is your app’s launch image and it provides the user with immediate feedback that your app is launching and will be ready soon. You must provide at least one launch image for your app and you may provide additional launch images to address specific scenarios.

## The App Bundle

* When you build your iOS app, Xcode packages it as a bundle. A bundle is a directory in the file system that groups related resources together in one place. An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.
* App executable: The executable file contains your app’s compiled code. The name of your app’s executable file is the same as your app name minus the .app extension.
* Info.plist
* App icons
* Launch Images
* Storyboard Files
* Settings bundle

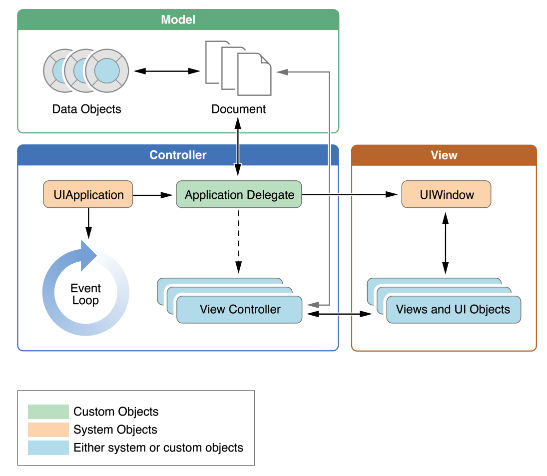
# The App Life Cycle

## The Main Function

* The entry point for every C-based app is the main function and iOS apps are no different. What is different is that for iOS apps you do not write the main function yourself. Instead, Xcode creates this function as part of your basic project.
* The only thing to mention about the main function is that its job is to hand control off to the UIKit framework. The [UIApplicationMain](https://developer.apple.com/documentation/uikit/1622933-uiapplicationmain" \t "_self) function handles this process by creating the core objects of your app, loading your app’s user interface from the available storyboard files, calling your custom code so that you have a chance to do some initial setup, and putting the app’s run loop in motion.
* **UIApplicationMain -** Creates the application object and the application delegate and sets up the event cycle.

## The Structure of an App

* During startup, the UIApplicationMain function sets up several key objects and starts the app running. At the heart of every iOS app is the UIApplication object, whose job is to facilitate the interactions between the system and other objects in the app. Figure 2-1 shows the objects commonly found in most apps, while Table 2-1 lists the roles each of those objects plays. The first thing to notice is that iOS apps use a model-view-controller architecture. This pattern separates the app’s data and business logic from the visual presentation of that data. This architecture is crucial to creating apps that can run on different devices with different screen sizes.
* **UIApplication -** The centralized point of control and coordination for apps running in iOS. Every iOS app has exactly one instance of UIApplication (or, very rarely, a subclass of UIApplication). When an app is launched, the system calls the [UIApplicationMain](https://developer.apple.com/documentation/uikit/1622933-uiapplicationmain?language=objc)function; among its other tasks, this function creates a [Singleton](https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html#//apple_ref/doc/uid/TP40008195-CH49) UIApplication object.



* **The Model-View-Controller (MVC)** design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. The collection of objects of a certain MVC type in an application is sometimes referred to as a layer—for example, model layer.
  + **Model objects** encapsulate the data specific to an application and define the logic and computation that manipulate and process that data. For example, a model object might represent a character in a game or a contact in an address book. A model object can have to-one and to-many relationships with other model objects, and so sometimes the model layer of an application effectively is one or more object graphs.
  + **A view object** is an object in an application that users can see. A view object knows how to draw itself and can respond to user actions. A major purpose of view objects is to display data from the application’s model objects and to enable the editing of that data. Despite this, view objects are typically decoupled from model objects in an MVC application.
  + **A controller object** acts as an intermediary between one or more of an application’s view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.
* The role of objects in an iOS app are as follows:
  + **UIApplication object -** The UIApplication object manages the event loop and other high-level app behaviors. It also reports key app transitions and some special events (such as incoming push notifications) to its delegate, which is a custom object you define. Use the UIApplication object as is—that is, without subclassing.
  + **App delegate object -** The app delegate is the heart of your custom code. This object works in tandem with the UIApplicationobject to handle app initialization, state transitions, and many high-level app events. This object is also the only one guaranteed to be present in every app, so it is often used to set up the app’s initial data structures.
  + **Documents and data model objects -** Data model objects store your app’s content and are specific to your app. For example, a banking app might store a database containing financial transactions, whereas a painting app might store an image object or even the sequence of drawing commands that led to the creation of that image.
  + **View Controller objects -** View controller objects manage the presentation of your app’s content on screen. A view controller manages a single view and its collection of subviews. When presented, the view controller makes its views visible by installing them in the app’s window.
  + **UIWindow object -** A [UIWindow](https://developer.apple.com/documentation/uikit/uiwindow" \t "_self) object coordinates the presentation of one or more views on a screen. Most apps have only one window, which presents content on the main screen, but apps may have an additional window for content displayed on an external display.
  + **View objects and control objects -** Views and controls provide the visual representation of your app’s content. A view is an object that draws content in a designated rectangular area and responds to events within that area. Controls are a specialized type of view responsible for implementing familiar interface objects such as buttons, text fields, and toggle switches.

## The Main Run Loop

* An app’s main run loop processes all user-related events. The [UIApplication](https://developer.apple.com/documentation/uikit/uiapplication" \t "_self) object sets up the main run loop at launch time and uses it to process events and handle updates to view-based interfaces. As the name suggests, the main run loop executes on the app’s main thread. This behavior ensures that user-related events are processed serially in the order in which they were received.
* Figure 2-2 shows the architecture of the main run loop and how user events result in actions taken by your app. As the user interacts with a device, events related to those interactions are generated by the system and delivered to the app via a special port set up by UIKit. Events are queued internally by the app and dispatched one-by-one to the main run loop for execution. The UIApplication object is the first object to receive the event and make the decision about what needs to be done. A touch event is usually dispatched to the main window object, which in turn dispatches it to the view in which the touch occurred. Other events might take slightly different paths through various app objects.

# Strategies for Handling App State Transitions

## What to do at Launch Time

* When your app is launched (either into the foreground or background), use your app delegate’s [application:willFinishLaunchingWithOptions:](https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1623032-application) and [application:didFinishLaunchingWithOptions:](https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1622921-application) methods to do the following:
  + Check the contents of the launch options dictionary for information about why the app was launched, and respond appropriately.
  + Initialize your app’s critical data structures.
  + Prepare your app’s window and views for display:
    - Apps that use OpenGL ES for drawing must not use these methods to prepare their drawing environment. Instead, defer any OpenGL ES drawing calls to the [applicationDidBecomeActive:](https://developer.apple.com/documentation/uikit/uiapplicationdelegate/1622956-applicationdidbecomeactive" \t "_self) method.
    - Show your app window from your application:willFinishLaunchingWithOptions:method. UIKit delays making the window visible until after the application:didFinishLaunchingWithOptions: method returns.
* At launch time, the system automatically loads your app’s main storyboard file and loads the initial view controller.
* Your application:willFinishLaunchingWithOptions: and application:didFinishLaunchingWithOptions: methods should always be as lightweight as possible to reduce your app’s launch time. Apps are expected to launch, initialize themselves, and start handling events in less than 5 seconds. If an app does not finish its launch cycle in a timely manner, the system kills it for being unresponsive. Thus, any tasks that might slow down your launch (such as accessing the network) should be scheduled performed on a secondary thread.
* application:willFinishLaunchingWithOptions: Tells the delegate that the launch process has begun but that state restoration has not yet occurred.
* application:didFinishLaunchingWithOptions: Tells the delegate that the launch process is almost done and the app is almost ready to run.

## 