UIApplicationDelegate

Language: Objective-C Swift Both On This Page Options

Inherits From
Not Applicable

Conforms To
`c:objc(pl)NSObject`

Import Statement
SWIFT
`import UIKit`

Availability
Available in iOS 2.0 and later

The `UIApplicationDelegate` protocol defines methods that are called by the singleton `UIApplication` object in response to important events in the lifetime of your app. The app delegate works alongside the app object to ensure your app interacts properly with the system and with other apps. Specifically, the methods of the app delegate give you a chance to respond to important changes. For example, you use the methods of the app delegate to respond to state transitions, such as when your app moves from foreground to background execution, and to respond to incoming notifications. In many cases, the methods of the app delegate are the only way to receive these important notifications.

Xcode provides an app delegate class for every new project, so you do not need to define one yourself initially. When your app launches, UIKit automatically creates an instance of the app delegate class provided by Xcode and uses it to execute the first bits of custom code in your app. All you have to do is take the class that Xcode provides and add your custom code.

The app delegate is effectively the root object of your app. Like the `UIApplication` object itself, the app delegate is a singleton object and is always present at runtime. Although the `UIApplication` object does most of the underlying work to manage the app, you decide your app's overall behavior by providing appropriate implementations of the app delegate's methods. Although most methods of this protocol are optional, you should implement most or all of them.

The app delegate performs several crucial roles:

- It contains your app's startup code.
- It responds to key changes in the state of your app. Specifically, it responds to both temporary interruptions and to changes in the execution state of your app, such as when your app transitions from the foreground to the background.
- It responds to notifications originating from outside the app, such as remote notifications (also known as push notifications), low-memory warnings, download completion notifications, and more.
- It determines whether state preservation and restoration should occur and assists in the preservation and restoration process as needed.
- It responds to events that target the app itself and are not specific to your app's views or view controllers.
- You can use it to store your app's central data objects or any content that does not have an owning view controller.

For more information about the role of the app delegate and how it fits into the app architecture, see *App Programming Guide for iOS*. For more information about the `UIApplication` singleton class, see *UIApplication Class Reference*.

Starting Up Your App

Launch time is an important point in an app's life cycle. At launch time, the app delegate is responsible for executing any custom code required to initialize your app. For example, the app delegate typically creates the app's initial data structures, registers for any required services, and tweaks the app's initial user interface based on any available data.

Some additional tasks that the app delegate performs at launch time include the following:

- **Look at the launch options dictionary to determine why your app was launched.** The `application:willFinishLaunchingWithOptions:` and `application:didFinishLaunchingWithOptions:` methods provide a dictionary with keys indicating the reason that your app was launched.
- **Determine whether state restoration should proceed.** If the app previously saved the state of its view controllers, restoration of those view controllers to their previous state proceeds only if the app delegate's `application:shouldRestoreApplicationState:` method returns `true`.
- **Register for any remote notifications your app supports.** In order to receive remote notifications (also known as push notifications), an app must register with the remote notification service by calling the `registerForRemoteNotificationTypes:` method of the `UIApplication` object. Usually, you perform this task at launch time.
- **Open a URL that was sent to your app.** If there is a URL to open, the system calls the `application:openURL:sourceApplication:annotation:` method of your app delegate. You also can tell if there is a URL to open by looking in the launch options dictionary for the `UIApplicationLaunchOptionsURLKey` key. You must declare the types of URLs your app supports by adding the `CFBundleURLTypes` key to your app's `Info.plist` file. For more information about registering and handling URLs, see *App Programming Guide for iOS*.
- **Provide the root window object for your app.** Technically, the app delegate provided by Xcode already implements the `window` property, so you do not have to do anything special here unless you want to customize your app's window.

The options dictionary passed to the `application:willFinishLaunchingWithOptions:` and `application:didFinishLaunchingWithOptions:` methods is an important source of launch-time information for your app. The keys in that dictionary tell you the reason your app was launched and give you a chance to adjust your launch procedures accordingly. For example, if your app was launched because of an incoming remote notification, you might want to reconfigure your user interface to display data related to that notification. For a list of possible reasons why your app might be launched, see Launch Options Keys.

Managing State Transitions

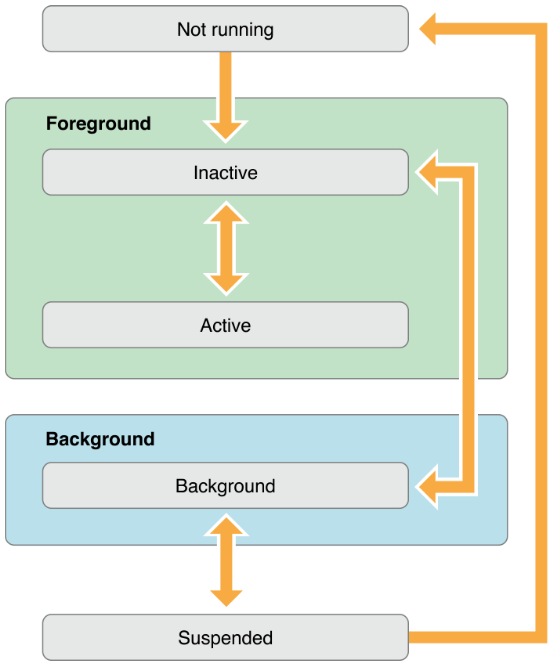
One of the main jobs of the app delegate is to respond to state transitions reported by the system. For every state change that occurs, the system calls the appropriate methods of the app delegate. Each state has different rules that govern how the app should is expected to behave and the app delegate methods must adjust the behavior of the app accordingly. Table 1 lists the possible states of an app and the high-level expectations. Figure 1 shows how an app moves from one state to another.

Table 1 App states

State	Description
Not running	The app has not been launched or was terminated, either by the user or the system.
Inactive	The app is running in the foreground but is 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. Upon entering this state, the app should put itself into a quiescent state with the expectation of moving to the background or active state shortly.
Active	The app is running in the foreground and receiving events. This is the normal mode for foreground apps. An app in the active state has no special restrictions placed on it. It is the

	An app in the active state has no special restrictions placed on it. It is the foreground app and should be responsive to the user.
Background	The app is executing code but is not visible onscreen. When the user quits an app, the system moves the app to the background state briefly before suspending it. At other times, the system may launch the app into the background (or wake up a suspended app) and give it time to handle specific tasks. For example, the system may wake up an app so that it can process background downloads, certain types of location events, remote notifications, and other types of events. An app in the background state should do as little work as possible. Apps that request time to process specific types of events should process those events and return control back to the system as quickly as possible.
Suspended	The app is in memory but is not executing code. The system suspends apps that are in the background and do not have any pending tasks to complete. The system may purge suspended apps at any time without waking them up to make room for other apps.

Figure 1 State changes in an iOS app



The following methods are called when state transitions occur. For information about what to do in response to each transition, see the method descriptions:

- Launch time:
    - `application:willFinishLaunchingWithOptions:`
    - `application:didFinishLaunchingWithOptions:`
- Transitioning to the foreground—
    - `applicationDidBecomeActive:`
- Transitioning to the background:
    - `applicationDidEnterBackground:`

- Transitioning to the inactive state
  - `applicationWillResignActive:` (Called when leaving the foreground state.)
  - `applicationWillEnterForeground:` (Called when transitioning out of the background state.)
- Termination:
  - `applicationWillTerminate:` (Called only when the app is running. This method is not called if the app is suspended.)

The specific tasks you perform during a given state transition are dependent upon your app and its capabilities. For tips and guidance about what to do during state transitions, see *App Programming Guide for iOS*.

Responding to Notifications and Events

The system sends many different notifications and events to the app delegate, letting the app delegate decide how best to respond to the incoming information and update the app. Most of these notifications correspond to app-level behaviors that might require you to update your app's data or user interface or respond to a changed condition of the system. The way you handle these notifications depends on the architecture of your app. In many cases, the app delegate might not do anything more than notify other objects (such as your app's view controllers) that they need to update themselves, but in some cases the app delegate might do the work itself.

- When a remote notification arrives, the system calls the `application:didReceiveRemoteNotification:fetchCompletionHandler:` method. Notifications usually signal the availability of new information. In your app delegate method, you might begin downloading new data from a server so that you can update your app's data structures. You might also use the notification to update your user interface.
- When a local notification fires, the system calls the `application:didReceiveLocalNotification:` method. The app must be running (or recently launched) to receive this event.
- When the user taps a custom action in the alert for a remote or local notification's, the system calls the `application:handleActionWithIdentifier:forRemoteNotification:completionHandler:` or `application:handleActionWithIdentifier:forLocalNotification:completionHandler:` method in the background so that your app can perform the associated action.
- For apps that want to initiate background downloads, the system calls the `application:performFetchWithCompletionHandler:` method when the time is right for you to start those downloads.
- For apps that use the `NSURLSession` class to perform background downloads, the system calls the `application:handleEventsForBackgroundURLSession:completionHandler:` method when those downloads finished while the app was not running. You can use this method to process the downloaded files and update the affected view controllers.
- When a low-memory condition occurs, the system notifies the app delegate by calling its `applicationDidReceiveMemoryWarning:` method. The app notifies its view controllers separately so the app delegate should use this notification to remove references to objects and data not managed directly by a view controller.
- When a significant change in time occurs, the system notifies the app delegate by calling its `applicationSignificantTimeChange:` method. If your app is sensitive to changes in time, you can use this method to update your app's data and user interface.
- When the user locks the device, the system calls the app delegate's `applicationProtectedDataWillBecomeUnavailable:` method. Data protection prevents unauthorized access to files while the device is locked. If your app references a protected file, you must remove that file reference and release any objects associated with the file when this method is called. When the user subsequently unlocks the device, you can reestablish your references to the data in the app delegate's

subsequently unlocks the device, you can reestablish your references to the data in the app delegate's `applicationProtectedDataDidBecomeAvailable:` method.

Monitoring App State Changes

`application(_:willFinishLaunchingWithOptions:)`

`application(_:didFinishLaunchingWithOptions:)`

`applicationDidBecomeActive(_:)`

`applicationWillResignActive(_:)`

`applicationDidEnterBackground(_:)`

`applicationWillEnterForeground(_:)`

`applicationWillTerminate(_:)`

Providing a Window for Storyboarding

`window` *Property*

Downloading Data in the Background

`application(_:performFetchWithCompletionHandler:)`

`application(_:handleEventsForBackgroundURLSession:completionHandler:)`

Handling Remote Notifications

`application(_:didRegisterForRemoteNotificationsWithDeviceToken:)`

`application(_:didFailToRegisterForRemoteNotificationsWithError:)`

`application(_:didReceiveRemoteNotification:fetchCompletionHandler:)`

`application(_:handleActionWithIdentifier:forRemoteNotification:completionHandler:)`

`application(_:didReceiveRemoteNotification:)`

Handling Local Notifications

`application(_:didReceiveLocalNotification:)`

`application(_:handleActionWithIdentifier:forLocalNotification:completionHandler:)`

Processing the User Notification Settings

`application(_:didRegisterUserNotificationSettings:)`

Responding to System Notifications

`applicationDidReceiveMemoryWarning(_:)`

`applicationSignificantTimeChange(_:)`

Responding to WatchKit Requests

`application(_:handleWatchKitExtensionRequest:reply:)`

Managing App State Restoration

`application(_:shouldSaveApplicationState:)`

`application(_:shouldRestoreApplicationState:)`

`application(_:viewControllerWithRestorationIdentifierPath:coder:)`

`application(_:willEncodeRestorableStateWithCoder:)`

`application(_:didDecodeRestorableStateWithCoder:)`

Opening a URL Resource

`application(_:openURL:sourceApplication:annotation:)`

Continuing User Activities

`application(_:willContinueUserActivityWithType:)`

`application(_:continueUserActivity:restorationHandler:)`

`application(_:didUpdateUserActivity:)` *Required*

`application(_:didFailToContinueUserActivityWithType:error:)`

Disallowing Use of Specified App Extension Types

`application(_:shouldAllowExtensionPointIdentifier:)` *Required*

Managing Status Bar Changes

`application(_:willChangeStatusBarOrientation:duration:)`

`application(_:didChangeStatusBarOrientation:)`

`application(_:willChangeStatusBarFrame:)`

`application(_:didChangeStatusBarFrame:)`

Responding to Data Protection Changes

`applicationProtectedDataWillBecomeUnavailable(_:)`

`applicationProtectedDataDidBecomeAvailable(_:)`

Managing the Default Interface Orientations

`application(_:supportedInterfaceOrientationsForWindow:)`

Deprecated Methods

`applicationDidFinishLaunching(_:)`

`application(_:handleOpenURL:)`

Constants

`Launch Options Keys`