Expected App Behaviors

Every new Xcode project comes configured to run right away in iOS Simulator or on a device. But simply being able to run on a device does not mean that your app is ready to ship on the App Store. Every app requires some amount of customization to ensure a good experience for the user. Customizations can range from providing an icon for your app to making architectural-level decisions about how your app presents and uses information. This chapter describes the behaviors that all apps are expected to handle and that you should consider early in the planning process.

Providing the Required Resources

Every app you create must have the following set of resources and metadata 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. For information about editing this file and for recommendations about what keys you should include, see The Information Property List File.
- **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. You can edit your app's list of requirements using the Required device capabilities entry in the Info tab of your project. For information on how to configure this key, see Declaring the Required Device Capabilities.
- **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. For information about how to specify app icons, see App Icons.
- **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. For information about creating your launch images, see App Launch (Default) Images.

These resources are required for all apps but are not the only ones you should include. There are many keys that Xcode does not include in your app's `Info.plist` file by default. Most of the additional keys are important only if you incorporate specific features into your app. For example, an app that uses the microphone should include the `NSMicrophoneUsageDescription` key and provide the user with information about how the app intends to use it.

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. Table 1-1 lists the contents of a typical iOS app bundle, which for demonstration purposes is called `MyApp`. This example is for illustrative purposes only. Some of the files listed in this table may not appear in your own app bundles.

Table 1-1 A typical app bundle

File	Example	Description
		The executable file contains your app's compiled code. The name of your app's

App executable	`MyApp`	executable file is the same as your app name minus the `.app` extension. This file is required.
The information property list file	`Info.plist`	The `Info.plist` file contains configuration data for the app. The system uses this data to determine how to interact with the app. This file is required and must be called `Info.plist`. For more information, see The Information Property List File.

App icons	`Icon.png`
	`Icon@2x.png`
	`Icon-Small.png`
	`Icon-Small@2x.png`
		Your app icon is used to represent your app on the device's Home screen. Other icons are used by the system in appropriate places. Icons with `@2x` in their filename are intended for devices with Retina displays.
		An app icon is required. For information about specifying icon image files, see App Icons.

Launch images	`Default.png`
	`Default-Portrait.png`
	`Default-Landscape.png`
		The system uses this file as a temporary background while your app is launching. It is removed as soon as your app is ready to display its user interface.
		At least one launch image is required. For information about specifying launch images, see App Launch (Default) Images.

Storyboard files (or nib files)	`MainBoard.storyboard`	Storyboards contain the views and view controllers that the app presents on screen. Views in a storyboard are organized according to the view controller that presents them. Storyboards also identify the transitions (called segues) that take the user from one set of views to another. The name of the main storyboard file is set by Xcode when you create your project. You can change the name by assigning a different value to the `UIMainStoryboardFile` key in the `Info.plist` file.) Apps that use nib files instead of storyboards can replace the `UIMainStoryboardFile` key with the `NSMainNibFile` key and use that key to specify their main nib file. The use of storyboards (or nib files) is optional but recommended.
		If you are distributing your app ad hoc, include a 512 x 512 pixel version of your app icon. This icon is normally provided by the App Store from the materials you submit to iTunes Connect. However, because apps distributed ad hoc do not go through the App Store, your icon must be present in your app bundle instead.

Ad hoc distribution icon	`iTunesArtwork`	iTunes uses this icon to represent your app. (The file you specify should be the same one you would have submitted to the App Store, if you were distributing your app that way.) The filename of this icon must be `iTunesArtwork` and must not include a filename extension. This file is required for ad hoc distribution but is optional otherwise.
Settings bundle	`Settings.bundle`	If you want to expose custom app preferences through the Settings app, you must include a settings bundle. This bundle contains the property list data and other resource files that define your app preferences. The Settings app uses the information in this bundle to assemble the interface elements required by your app. This bundle is optional. For more information about preferences and specifying a settings bundle, see *Preferences and Settings Programming Guide*.
Nonlocalized resource files	`sun.png` `mydata.plist`	Nonlocalized resources include things like images, sound files, movies, and custom data files that your app uses. All of these files should be placed at the top level of your app bundle.

Subdirectories for localized resources	`en.lproj`
	`fr.lproj`
	`es.lproj`
		Localized resources must be placed in language-specific project directories, the names for which consist of an ISO 639-1 language abbreviation plus the `.lproj` suffix. (For example, the `en.lproj`, `fr.lproj`, and `es.lproj` directories contain resources localized for English, French, and Spanish.)
		An iOS app should be internationalized and have a *language*`.lproj` directory for each language it supports. In addition to providing localized versions of your app's custom resources, you can also localize your app icon, launch images, and Settings icon by placing files with the same name in your language-specific project directories.
		For more information, see Internationalizing Your App.

For more information about the structure of an iOS app bundle, see *Bundle Programming Guide*. For information about how to load resource files from your bundle, see *Resource Programming Guide*.

The Information Property List File

Xcode uses information from the General, Capabilities, and Info tabs of your project to generate an information property list (`Info.plist`) file for your app at compile time. The `Info.plist` file is a

structured file that contains critical information about your app's configuration. It is used by the App Store and by iOS to determine your app's capabilities and to locate key resources. Every app must include this file.

Although the `Info.plist` file provided by Xcode includes default values for all of the required entries, most apps require some changes or additions. Whenever possible, use the General and Capabilities tabs to specify the configuration information for your app. Those tabs contain the most common configuration options available for apps. If you do not see a specific option on either of those tabs, use the Info tab.

For options where Xcode does not provide a custom configuration interface, you must provide appropriate keys and values. The Custom iOS Target Properties section of the Info tab contains a summary of the entries to be included in the `Info.plist` file. By default, Xcode displays human-readable descriptions of the intended feature but each feature actually corresponds to a unique key in the `Info.plist` file. Most keys are optional and used infrequently, but there are a handful of keys that you should consider when defining any new project:

- **Declare your app's required capabilities in the Info tab.** The Required device capabilities section contains information about the device-level features that your app requires to run. The App Store uses the information in this entry to determine the capabilities of your app and to prevent it from being installed on devices that do not support features your app requires. For more information, see Declaring the Required Device Capabilities.
- **Apps that require a persistent Wi-Fi connection must declare that fact.** If your app talks to a server across the network, you can add the Application uses Wi-Fi entry to the Info tab of your project. This entry corresponds to the `UIRequiresPersistentWiFi` key in the `Info.plist` file. Setting this key to `YES` prevents iOS from closing the active Wi-Fi connection when it has been inactive for an extended period of time. This key is recommended for all apps that use the network to communicate with a server.
- **Newsstand apps must declare themselves as such.** Include the `UINewsstandApp` key to indicate that your app presents content from the Newsstand app.
- **Apps that define custom document types must declare those types.** Use the Document Types section of the Info tab to specify icons and UTI information for the document formats that you support. The system uses this information to identify apps capable of handling specific file types. For more information about adding document support to your app, see *Document-Based App Programming Guide for iOS*.
- **Apps can declare any custom URL schemes they support.** Use the URL Types section of the Info tab to specify the custom URL schemes that your app handles. Apps can use custom URL schemes to communicate with each other. For more information about how to implement support for this feature, see Using URL Schemes to Communicate with Apps.
- **Apps should provide usage descriptions for certain app features.** Whenever there is a privacy concern about an app accessing a user's data or a device's capabilities, iOS frameworks prompt the user and request permission for your app to use the feature. Apps that use these features should provide privacy usage descriptions that explain what your app plans to do with the corresponding data. For information about the features that require user permission, see Table 1-2.

For detailed information about the keys and values you can include in the `Info.plist` file, see *Information Property List Key Reference*.

Declaring the Required Device Capabilities

All apps must declare the device-specific capabilities they need to run. Xcode includes a Required device capabilities entry in your project's Info tab and populates it with some minimum requirements. You can add values to this entry to specify additional requirements for your app. The Required device capabilities entry corresponds to the `UIRequiredDeviceCapabilities` key in your app's `Info.plist` file.

The value of the `UIRequiredDeviceCapabilities` key is either an array or dictionary that contains additional keys identifying features your app requires (or specifically prohibits). If you specify the value of the key using an array, the presence of a key indicates that the feature is required; the absence of a key indicates that the feature is not required and that the app can run without it. If you specify a dictionary instead, each key in the dictionary must have a Boolean value that indicates whether the feature is required or prohibited. A value of `true` indicates the feature is required and a value of `false` indicates that the feature must *not* be present on the device. If a given capability is

optional for your app, do not include the corresponding key in the dictionary.

For detailed information on the values you can include for the `UIRequiredDeviceCapabilities` key, see *Information Property List Key Reference*.

App Icons

Every app must provide an icon to be displayed on a device's Home screen and in the App Store. An app may actually specify several different icons for use in different situations. For example, an app can provide a small icon to use when displaying search results and can provide a high-resolution icon for devices with Retina displays.

New Xcode projects include image asset entries for your app's icon images. To add icons, assign the corresponding image files to the image assets of your project. At build time, Xcode adds the appropriate keys to your app's `Info.plist` file and places the images in your app bundle.

For information about designing your app icons, including the sizes of those icons, see *iOS Human Interface Guidelines*.

App Launch (Default) Images

When the system launches an app for the first time on a device, it temporarily displays a static launch image on the screen. This image is your app's launch image and it is a resource that you specify in your Xcode project. Launch images provide the user with immediate feedback that your app has launched while giving your app time to prepare its initial user interface. When your app's window is configured and ready to be displayed, the system swaps out the launch image for that window.

When a recent snapshot of your app's user interface is available, the system prefers the use of that image over the use of your app's launch images. The system takes a snapshot of your app's user interface when your app transitions from the foreground to the background. When your app returns to the foreground, it uses that image instead of a launch image whenever possible. In cases where the user has killed your app or your app has not run for a long time, the system discards the snapshot and relies once again on your launch images.

New Xcode projects include image asset entries for your app's launch images. To add launch images, add the corresponding image files to the image assets of your project. At build time, Xcode adds the appropriate keys to your app's `Info.plist` file and places the images in your app bundle.

For information about designing your app's launch images, including the sizes of those images, see *iOS Human Interface Guidelines*.

Supporting User Privacy

Maintaining user privacy should be an important consideration when designing your app. Most iOS devices contain user and device data that users might not want to expose to apps or external entities. Remember that the user might delete your app if it uses data in an inappropriate way.

Access user or device data only with the user's informed consent obtained in accordance with applicable law. In addition, take appropriate steps to protect user and device data and be transparent about how you use it. Here are some best practices that you can take:

- Review guidelines from government or industry sources, including the following documents:
  - The Federal Trade Commission's report on mobile privacy: Mobile Privacy Disclosures: Building Trust Through Transparency.
  - The EU Data Protection Commissioners' Opinion on data protection for Mobile Apps: http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2013/wp202_en.pdf
  - The Japanese Ministry of Internal Affairs and Communications' Smartphone Privacy Initiatives:
    - Smartphone Privacy Initiative (2012):

English:
http://www.soumu.go.jp/main_sosiki/joho_tsusin/eng/presentation/pdf/Initiative.pdf

Japanese: http://www.soumu.go.jp/main_content/000171225.pdf

- Smartphone Privacy Initiative II (2013):

  English:
  http://www.soumu.go.jp/main_sosiki/joho_tsusin/eng/presentation/pdf/Summary_II.pdf

  Japanese: http://www.soumu.go.jp/main_content/000247654.pdf

- The California State Attorney General's recommendations for mobile privacy: Privacy on the Go: Recommendations for the Mobile Ecosystem

These reports provide helpful recommendations for protecting user privacy. You should also review these documents with your company's legal counsel.

- Request access to user or device data that is protected by the iOS system authorization settings at the time the data is needed. Consider supplying a usage description string in your app's `Info.plist` file explaining why your app needs that data. Data protected by iOS system authorization settings includes location data, contacts, calendar events, reminders, photos, and media; see Table 1-2. Provide reasonable fallback behavior in situations where the user does not grant access to the requested data.
- Be transparent with users about how their data is going to be used. For example, you should specify a URL for your privacy policy or statement with your iTunes Connect metadata when you submit your app, and you might also want to summarize that policy in your app description.

  For more information about providing your app's privacy policy in iTunes Connect, see Adding an App in iTunes Connect.
- Give the user control over their user or device data. Provide settings so that the user can disable access to certain types of sensitive information as needed.
- Request and use the minimum amount of user or device data needed to accomplish a given task. Do not seek access to or collect data for non obvious reasons, for unnecessary reasons, or because you think it might be useful later.
- Take reasonable steps to protect the user and device data that you collect in your apps. When storing such information locally, try to use the iOS data protection feature (described in Protecting Data Using On-Disk Encryption) to store it in an encrypted format. And try to use HTTPS when sending user or device data over the network.
- If your app uses the `ASIdentifierManager` class, you must respect the value of its `advertisingTrackingEnabled` property. And if that property is set to `NO` by the user, then use the `ASIdentifierManager` class only for Limited Advertising Purposes. "Limited Advertising Purposes" means frequency capping, attribution, conversion events, estimating the number of unique users, advertising fraud detection, debugging for advertising purposes only, and other uses for advertising that may be permitted by Apple in Documentation for the Ad Support APIs.
- If you have not already done so, stop using the unique device identifier (UDID) provided by the `uniqueIdentifier` property of the `UIDevice` class. That property was deprecated in iOS 5.0, and the App Store does not accept new apps or app updates that use that identifier. Instead, apps should use the `identifierForVendor` property of the `UIDevice` class or the `advertisingIdentifier` property of the `ASIdentifierManager` class, as appropriate.
- If your app supports audio input, configure your audio session for recording only at the point where you actually plan to begin recording. Do not configure your audio session for recording at launch time if you do not plan to record right away. In iOS 7, the system alerts users when apps configure their audio session for recording and gives the user the option to disable recording for your app.

Table 1-2 lists the types of data authorizations supported by iOS. Using the services listed in this table causes an alert to be displayed to the user requesting permission to do so. You can determine if the user authorized your app for a service using the API listed for each item. You should view this table as a starting point for your app's own privacy behaviors and not as a finite checklist. The contents of this table may evolve over time.

Table 1-2 Data protected by system authorization settings

Data	System authorization support
Location	The current authorization status for location data is available from the `authorizationStatus` class method of `CLLocationManager`. In requesting authorization in iOS 8 and later, you must use the `requestWhenInUseAuthorization` or `requestAlwaysAuthorization` method and include the `NSLocationWhenInUseUsageDescription` or `NSLocationAlwaysUsageDescription` key in your `Info.plist` file to indicate the level of authorization you require.
Photos	The authorization status for photo data is available from the `authorizationStatus` method of the `PHPhotoLibrary` class. To inform the user about how you intend to use this information, include the NSPhotoLibraryUsageDescription key in your `Info.plist` file.
Music, video, and other media assets	The authorization status for media assets is available from the `authorizationStatus` method of `ALAssetsLibrary`.
Contacts	The authorization status for contact data is available from the `ABAddressBookGetAuthorizationStatus` function. To inform the user about how you intend to use this information, include the `NSContactsUsageDescription` key in your `Info.plist` file.
Calendar data	The authorization status for calendar data is available from the `authorizationStatusForEntityType:` method of `EKEventStore`. To inform the user about how you intend to use this information, include the `NSCalendarsUsageDescription` key in your `Info.plist` file.
Reminders	The authorization status for reminder data is available from the `authorizationStatusForEntityType:` method of `EKEventStore`. To inform the user about how you intend to use this information, include the `NSRemindersUsageDescription` key in your `Info.plist` file.
Bluetooth peripherals	The authorization status for Bluetooth peripherals is available from the `state` property of `CBCentralManager`. To inform the user about how you intend to use Bluetooth, include the `NSBluetoothPeripheralUsageDescription` key in your `Info.plist` file.
Microphone	In iOS 7 and later, the authorization status for the microphone is available from the `requestRecordPermission:` method of `AVAudioSession`. To inform the user about how you intend to use the microphone, include the `NSMicrophoneUsageDescription` key in your `Info.plist` file.
Camera	In iOS 7 and later, the authorization status for the camera is available in `deviceInputWithDevice:error:` method of `AVCaptureDeviceInput`. To inform the user about how you intend to use the camera, include the `NSCameraUsageDescription` key in your `Info.plist` file.

Internationalizing Your App

Because iOS apps are distributed in many countries, localizing your app's content can help you reach many more customers. Users are much more likely to use an app when it is localized for their native language. When you factor your user-facing content into resource files, localizing that content is a relatively simple process.

Before you can localize your content, you must internationalize your app in order to facilitate the localization process. Internationalizing your app involves factoring out any user-facing content into localizable resource files and providing language-specific project (`.lproj`) directories for storing that content. It also means using appropriate technologies (such as date and number formatters) when working with language-specific and locale-specific content.

For a fully internationalized app, the localization process creates new sets of language-specific resource files for you to add to your project. A typical iOS app requires localized versions of the following types of resource files:

- Storyboard files (or nib files)—Storyboards can contain text labels and other content that need to be localized. You might also want to adjust the position of interface items to accommodate changes in text length. (Similarly, nib files can contain text that needs to be localized or layout that needs to be updated.)
- Strings files—Strings files (so named because of their `.strings` filename extension) contain localized versions of the static text that your app displays.
- Image files—You should avoid localizing images unless the images contain culture-specific content. Whenever possible, you should avoid storing text directly in your image files. For images that you load and use from within your app, store text in a strings file and composite that text with your image-based content at runtime.
- Video and audio files—You should avoid localizing multimedia files unless they contain language-specific or culture-specific content. For example, you would want to localize a video file that contained a voice-over track.

For information about the internationalization and localization process, see *Internationalization and Localization Guide*. For information about the proper way to use resource files in your app, see *Resource Programming Guide*.

Copyright © 2015 Apple Inc. All Rights Reserved. Terms of Use | Privacy Policy | Updated: 2015-09-16