Information Property List Key Reference

Developer

Contents

About Info.plist Keys 10

At a Glance 10

- The Info.plist File Configures Your App 10
- Core Foundation Keys Describe Common Behavior 10
- Launch Services Keys Describe Launch-Time Behavior 11
- Cocoa Keys Describe Behavior for Cocoa and Cocoa Touch Apps 11
- OS X Keys Describe Behavior for OS X Apps 11
- iOS Keys Describe Behavior for iOS Apps 11
- System Extension Keys Describe Behavior for iOS and OS X Extensions 11

See Also 12

About Information Property List Files 13

Creating and Editing an Information Property List File 13

Adding Keys to an Information Property List File 15

Localizing Property List Values 15

Creating Device-Specific Keys 16

Custom Keys 17

Recommended Info.plist Keys 17

- Recommended Keys for iOS Apps 17
- Recommended Keys for Cocoa Apps 18
- Commonly Localized Keys 19

Core Foundation Keys 20

Key Summary 20

CFAppleHelpAnchor 23

CFBundleAllowMixedLocalizations 23

CFBundleDevelopmentRegion 23

CFBundleDisplayName 24

CFBundleDocumentTypes 24

- Document Roles 31
- Document Icons 31
- Recommended Keys 32

CFBundleExecutable 32

CFBundleHelpBookFolder 33

CFBundleHelpBookName 33
CFBundleIconFile 33
CFBundleIconFiles 34
CFBundleIcons 34
    Contents of the CFBundlePrimaryIcon Dictionary 35
    Contents of the UINewsstandIcon Dictionary 35
CFBundleIdentifier 37
CFBundleInfoDictionaryVersion 37
CFBundleLocalizations 37
CFBundleName 38
CFBundlePackageType 38
CFBundleShortVersionString 38
CFBundleSignature 38
CFBundleSpokenName 39
CFBundleURLTypes 39
CFBundleVersion 40
CFPlugInDynamicRegistration 40
CFPlugInDynamicRegisterFunction 40
CFPlugInFactories 41
CFPlugInTypes 41
CFPlugInUnloadFunction 41

Launch Services Keys 42
Key Summary 42
LSApplicationCategoryType 44
LSArchitecturePriority 46
LSBackgroundOnly 47
LSEnvironment 47
LSFileQuarantineEnabled 47
LSFileQuarantineExcludedPathPatterns 47
LSGetAppDiedEvents 48
LSMinimumSystemVersion 48
LSMinimumSystemVersionByArchitecture 48
LSMultipleInstancesProhibited 49
LSRequiresIPhoneOS 49
LSRequiresNativeExecution 49
LSUIElement 49
LSUIPresentationMode 50
LSVisibleInClassic 50

MinimumOSVersion 51

Cocoa Keys 52

Key Summary 52
GKGameCenterBadgingDisabled 55
GKShowChallengeBanners 56
NSAppleScriptEnabled 56
NSBluetoothPeripheralUsageDescription 56
NSCalendarsUsageDescription 56
NSCameraUsageDescription 56
NSContactsUsageDescription 57
NSDockTilePlugIn 57
NSHumanReadableCopyright 57
NSJavaNeeded 57
NSJavaPath 57
NSJavaRoot 58
NSLocationAlwaysUsageDescription 58
NSLocationUsageDescription 58
NSLocationWhenInUseUsageDescription 59
NSMainNibFile 59
NSMicrophoneUsageDescription 59
NSMotionUsageDescription 60
NSPersistentStoreTypeKey 60
NSPhotoLibraryUsageDescription 60
NSPrefPaneIconFile 60
NSPrefPaneIconLabel 60
NSPrincipalClass 61
NSRemindersUsageDescription 61
NSServices 61
NSSupportsAutomaticTermination 70
NSSupportsSuddenTermination 71
NSUbiquitousContainers 71
NSUbiquitousContainerIsDocumentScopePublic 71
NSUbiquitousContainerName 71
NSUbiquitousContainerSupportedFolderLevels 71
NSUbiquitousDisplaySet 72
NSUserActivityTypes 72
NSUserNotificationAlertStyle 72
UTExportedTypeDeclarations 73

UTImportedTypeDeclarations 74

OS X Keys 76
Key Summary 76
APInstallerURL 77
APFiles 77
ATSApplicationFontsPath 77
CSResourcesFileMapped 78
NSMainStoryboardFile 78
QLSandboxUnsupported 78
QuartzGLEnable 78

iOS Keys 80
Key Summary 80
MKDirectionsApplicationSupportedModes 82
NSHealthShareUsageDescription 83
NSHealthUpdateUsageDescription 83
UIAppFonts 84
UIApplicationExitsOnSuspend 84
UIBackgroundModes 84
UIDeviceFamily 86
UIFileSharingEnabled 86
UIInterfaceOrientation 86
UILaunchImageFile 87
    Naming Your Launch Image Files 87
UILaunchImages 88
UIMainStoryboardFile 89
UINewsstandApp 89
UIPrerenderedIcon 90
UIRequiredDeviceCapabilities 90
UIRequiresPersistentWiFi 93
UIStatusBarHidden 94
UIStatusBarStyle 94
UISupportedExternalAccessoryProtocols 94
UISupportedInterfaceOrientations 95
UIViewControllerBasedStatusBarAppearance 95
UIViewEdgeAntialiasing 96
UIViewGroupOpacity 96

App Extension Keys 97

Key Summary 98
General App Extension Keys 100
- NSExtension 100
- NSExtensionAttributes 100
- NSExtensionMainStoryboard 100
- NSExtensionPointIdentifier 101
- NSExtensionPrincipalClass 101

Action Extension Keys 102
- NSExtensionActionWantsFullScreenPresentation 102
- NSExtensionActivationRule 102
- NSExtensionJavaScriptPreprocessingFile 103
- NSExtensionServiceAllowsToolbarItem 103
- NSExtensionServiceRoleType 103
- NSExtensionServiceToolbarIconFile 104
- NSExtensionServiceToolbarPaletteLabel 104

Custom Keyboard Extension Keys 104
- IsASCIICapable 104
- PrefersRightToLeft 105
- PrimaryLanguage 105
- RequestsOpenAccess 105

Document Picker and File Provider Extension Keys 105
- NSExtensionFileProviderDocumentGroup 105
- UIDocumentPickerModes 106
- UIDocumentPickerSupportedFileTypes 106

Photo Editing Extension Keys 106
- PHSupportedMediaTypes 106

Share Extension Keys 107
- NSExtensionActivationRule 107
- NSExtensionJavaScriptPreprocessingFile 107

Document Revision History 108

Swift 9

Figures and Tables

About Information Property List Files 13

Figure 1 Editing the information property list in Xcode 14

Core Foundation Keys 20

Table 1 Summary of Core Foundation keys 20
Table 2 Keys for type-definition dictionaries 25
Table 3 Document icon sizes for iOS 31
Table 4 Keys for the `CFBundlePrimaryIcon` dictionary 35
Table 5 Keys for the `UINewsstandIcon` dictionary 36
Table 6 Keys for CFBundleURLTypes dictionaries 39

Launch Services Keys 42

Table 1 Summary of Launch Services keys 42
Table 2 UTIs for app categories 44
Table 3 UTIs for game-specific categories 45
Table 4 Execution architecture identifiers 46

Cocoa Keys 52

Table 1 Summary of Cocoa keys 52
Table 2 Keys for NSServices dictionaries 61
Table 3 Contents of the `NSRequiredContext` dictionary 69
Table 4 UTI property list keys 73

OS X Keys 76

Table 1 Summary of OS X keys 76
Table 2 Keys for APFiles dictionary 77

iOS Keys 80

Table 1 Summary of UIKit keys 80
Table 2 Values for the `UIBackgroundModes` array 84
Table 3 Values for the `UIDeviceFamily` key 86
Table 4 Keys for launch image dictionaries 88
Table 5 Dictionary keys for the `UIRequiredDeviceCapabilities` key 91
Table 6 Supported orientations 95

App Extension Keys 97

Table 1 Summary of app extension keys 98

Table 2 Values for the `NSExtensionPointIdentifier` key 101

Table 3 Keys for the `NSExtensionActivationRule` dictionary 102

Table 4 Values for the `NSExtensionServiceRoleType` key 104

Table 5 Values for the `UIDocumentPickerModes` key 106

SwiftObjective-C

About Info.plist Keys

To provide a better experience for users, iOS and OS X rely on the presence of special meta information in each app or bundle. This meta information is used in many different ways. Some of it is displayed to the user, some of it is used internally by the system to identify your app and the document types it supports, and some of it is used by the system frameworks to facilitate the launch of apps. The way an app provides its meta information to the system is through the use of a special file called an information property list file.

Property lists are a way of structuring arbitrary data and accessing it at runtime. An information property list is a specialized type of property list that contains configuration data for a bundle. The keys and values in the file describe the various behaviors and configuration options you want applied to your bundle. Xcode typically creates an information property list file for any bundle-based projects automatically and configures an initial set of keys and values with appropriate default values. You can edit the file, however, to add any keys and values that are appropriate for your project or change the default values of existing keys.

At a Glance

This document describes the keys (and corresponding values) that you can include in an information property list file. This document also includes an overview of information property list files to help you understand their importance and to provide tips on how to configure them.

The Info.plist File Configures Your App

Every app and plug-in uses an `Info.plist` file to store configuration data in a place where the system can easily access it. OS X and iOS use `Info.plist` files to determine what icon to display for a bundle, what document types an app supports, and many other behaviors that have an impact outside the bundle itself.

Relevant chapter: About Information Property List Files (page 13)

Core Foundation Keys Describe Common Behavior

There are many keys that you always specify, regardless of the type of bundle you are creating. Those keys start with a CF prefix and are known as the Core Foundation keys. Xcode includes the most important keys in your `Info.plist` automatically but there are others you must add manually.

Relevant chapter: Core Foundation Keys (page 20)

Launch Services Keys Describe Launch-Time Behavior

Launch Services provides support for launching apps. To do this, though, it needs to know information about how your app wants to be launched. The Launch Services keys describe the way your app prefers to be launched.

Relevant chapter: Launch Services Keys (page 42)

Cocoa Keys Describe Behavior for Cocoa and Cocoa Touch Apps

The Cocoa and Cocoa Touch frameworks use keys to identify high-level information such as your app's main nib file and principal class. The Cocoa keys describe those and other keys that affect how the Cocoa and Cocoa Touch frameworks initialize and run your app.

Relevant chapter: Cocoa Keys (page 52)

OS X Keys Describe Behavior for OS X Apps

Some OS X frameworks use keys to modify their basic behavior. Developers of Mac apps might include these keys during testing or to modify certain aspects of your app's behavior.

Relevant chapter: OS X Keys (page 76)

iOS Keys Describe Behavior for iOS Apps

An iOS app communicates a lot of information to the system using `Info.plist` keys. Xcode supplies a standard `Info.plist` with the most important keys but most apps need to augment the standard file with additional keys describing everything from the app's initial orientation to whether it supports file sharing.

Relevant chapter: iOS Keys (page 80)

System Extension Keys Describe Behavior for iOS and OS X Extensions

System extensions are a way to incorporate your own custom behavior into existing system facilities such as notification center. . Xcode supplies a standard `Info.plist` with the most important keys but you can specify additional keys that describe custom behavior for your extensions.

Relevant chapter: System Extension Keys (page 97)

See Also

For more information about generic property lists, including how they are structured and how you use them, see *Property List Programming Guide* .

Some information property list keys use Uniform Type Identifiers (UTIs) to refer to data of different types. For an introduction to UTIs and how they are specified, see *Uniform Type Identifiers Overview* .

About Information Property List Files

An information property list file is a structured text file that contains essential configuration information for a bundled executable. The file itself is typically encoded using the Unicode UTF-8 encoding and the contents are structured using XML. The root XML node is a dictionary, whose contents are a set of keys and values describing different aspects of the bundle. The system uses these keys and values to obtain information about your app and how it is configured. As a result, all bundled executables (plug-ins, frameworks, and apps) are expected to have an information property list file.

By convention, the name of an information property list file is `Info.plist`. This name of this file is case sensitive and must have an initial capital letter `I`. In iOS apps, this file resides in the top-level of the bundle directory. In OS X bundles, this file resides in the bundle's `Contents` directory. Xcode typically creates this file for you automatically when you create a project of an appropriate type.

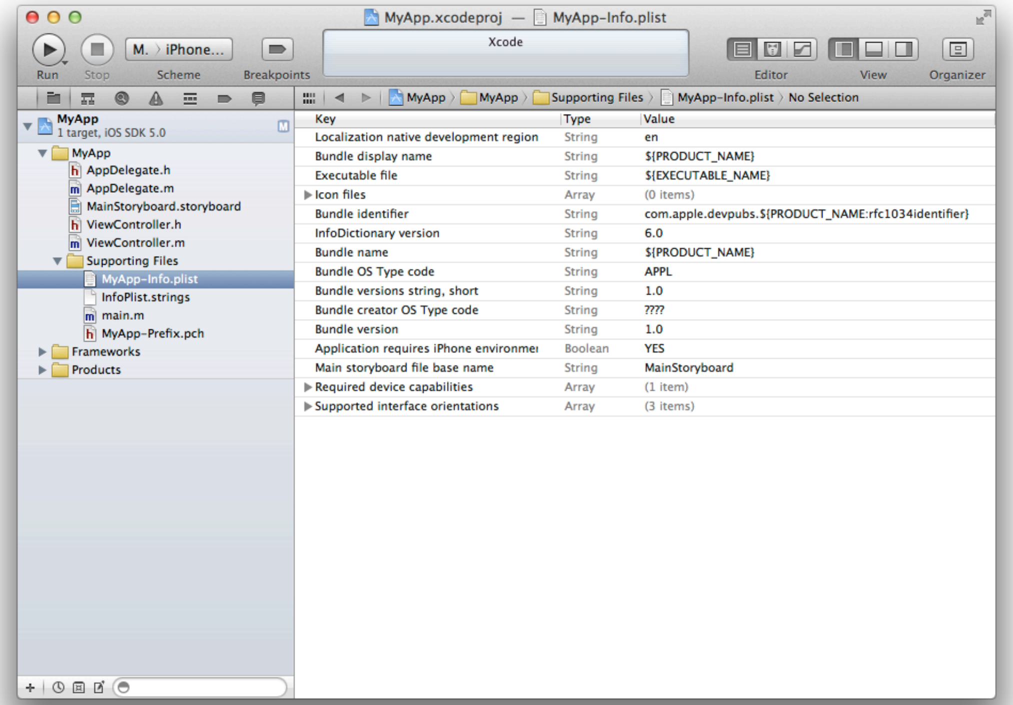
Important: In the sections that follow, pay attention to the capitalization of files and directories that reside inside a bundle. The `NSBundle` class and Core Foundation bundle functions consider case when searching for resources inside a bundle directory. Case mismatches could prevent you from finding your resources at runtime.

Creating and Editing an Information Property List File

The simplest way to create an information property list file is to let Xcode create it for you. Each new bundle-based project that you create in Xcode comes with a file named *<project>*`-Info.plist`, where *<project>* is the name of the project. At build time, this file is used to generate the `Info.plist` file that is then included in the resulting bundle.

To edit the contents of your information property list file, select the <*project*>`-Info.plist` file in your Xcode project to display the property list editor. Figure 1 shows the editor for the information property list file of a new Cocoa app project. The file created by Xcode comes preconfigured with keys that every information property list should have.

Figure 1 Editing the information property list in Xcode



Key	Type	Value
Localization native development region	String	en
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
▶ Icon files	Array	(0 items)
Bundle identifier	String	com.apple.devpubs.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone environme	Boolean	YES
Main storyboard file base name	String	MainStoryboard
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)

To edit the value for a specify key, double-click the value in the Xcode property list editor to select it, then type a new value. Most values are specified as strings but Xcode also supports several other scalar types. You can also specify complex types such as an array or dictionary. The property list editor displays an appropriate interface for editing each type. To change the type of a given value, make sure the value is not selected and Control-click it to display its contextual menu. From the Value Type submenu, select the type you want to use for the value.

Because information property lists are usually just text files, you can also edit them using any text editor that supports the UTF-8 file encoding. Because they are XML files, however, editing property list files manually is generally discouraged.

Adding Keys to an Information Property List File

Although the `Info.plist` file provided by Xcode contains the most critical keys required by the system, most apps should typically specify several additional keys. Many subsystems and system apps use the `Info.plist` file to gather information about your app. For example, when the user chooses File > Get Info for your app, the Finder displays information from many of these keys in the resulting information window.

You add keys to your app's Info.plist using the Xcode property list editor. For information about how to use this editor, see *Property List Editor Help* .

Important: The property list editor in Xcode displays human-readable strings (instead of the actual key name) for many keys by default. To display the actual key names as they appear in the `Info.plist` file, Control-click any of the keys in the editor window and enable the Show Raw Keys/Values item in the contextual menu.

For a list of the recommended keys you should include in a typical app, see Recommended Info.plist Keys (page 17).

Localizing Property List Values

The values for many keys in an information property list file are human-readable strings that are displayed to the user by the Finder or your own app. When you localize your app, you should be sure to localize the values for these strings in addition to the rest of your app's content.

Localized values are not stored in the `Info.plist` file itself. Instead, you store the values for a particular localization in a strings file with the name `InfoPlist.strings`. You place this file in the same language-specific project directory that you use to store other resources for the same localization. The contents of the `InfoPlist.strings` file are the individual keys you want localized and the appropriately translated value. The routines that look up key values in the `Info.plist` file take the user's language preferences into account and return the localized version of the key (from the appropriate `InfoPlist.strings` file) when one exists. If a localized version of a key does not exist, the routines return the value stored in the `Info.plist` file.

For example, the TextEdit app has several keys that are displayed in the Finder and thus should be localized. Suppose your information property list file defines the following keys:

```
<key>CFBundleDisplayName</key>
<string>TextEdit</string>
<key>NSHumanReadableCopyright</key>
<string>Copyright ¬© 1995-2009, Apple Inc.,All Rights Reserved.
</string>
```

The French localization for TextEdit then includes the following strings in the `InfoPlist.strings` file of its `Contents/Resources/French.lproj` directory:

```
CFBundleDisplayName = "TextEdit";
NSHumanReadableCopyright = "Copyright © 1995-2009 Apple Inc.\nTous droits réservés.";
```

For more information about the placement of `InfoPlist.strings` files in your bundle, see *Bundle Programming Guide* . For information about creating strings files, see *Resource Programming Guide* . For additional information about the localization process, see *Internationalization and Localization Guide* .

Creating Device-Specific Keys

In iOS 3.2 and later, apps can designate keys in the `Info.plist` file as being applicable only to specific types of devices. To create a device-specific key, you combine the key name with some special qualifiers using the following pattern:

key_root-<*platform*>~<*device*>

In this pattern, the *key_root* portion represents the original name of the key. The <*platform*> and <*device*> portions are both optional endings that you can use to apply keys to specific platforms or devices. For the platform key, you can specify a value of `iphoneos` or `macos` depending on the platform you are targeting.

To apply a key to a specific device, you can use one of the following values:

- `iphone` - The key applies to iPhone devices only.
- `ipod` - The key applies to iPod touch devices only.
- `ipad` - The key applies to iPad devices only.

When specifying keys, it is recommended that you always include a key without any device modifiers so that you have a reasonable default value. You can then add versions of the key with device-specific overrides, as needed, to customize the value for a specific device. When searching for a key in your app's `Info.plist` file, the system chooses the key that is most specific to the current device first. If it does not find a device-specific key, it looks for one without any device modifiers. Thus, having an unmodified key means that there is always a default value for the given key. For example, to specify a portrait launch orientation for iPhone and iPod touch devices and a landscape-right launch orientation for iPad, you should specify the corresponding keys in your `Info.plist` file:

```
<key>UIInterfaceOrientation</key>
```

```
<string>UIInterfaceOrientationPortrait</string>
<key>UIInterfaceOrientation~ipad</key>
<string>UIInterfaceOrientationLandscapeRight</string>
```

Custom Keys

OS X and iOS ignore any custom keys you include in an Info.plist file. If you want to include app-specific configuration information in your `Info.plist` file, you may do so freely as long as your key names do not conflict with the ones Apple uses. When defining custom key names, it is advised that you prefix them with a unique prefix such as your app's bundle ID or your company's domain name.

Recommended Info.plist Keys

When creating an information property list file, there are several keys that you should always include. These keys are almost always accessed by the system and providing them ensures that the system has the information it needs to work with your app effectively.

Recommended Keys for iOS Apps

It is recommended that an iOS app include the following keys in its information property list file. Most are set by Xcode automatically when you create your project.

- CFBundleDevelopmentRegion (page 23)
- CFBundleDisplayName (page 24)
- CFBundleExecutable (page 32)
- CFBundleIconFiles (page 34)
- CFBundleIdentifier (page 37)
- CFBundleInfoDictionaryVersion (page 37)
- CFBundlePackageType (page 38)
- CFBundleVersion (page 40)
- LSRequiresIPhoneOS (page 49)
- UIMainStoryboardFile (page 89)

In addition to these keys, there are several that are commonly included:

- UIRequiredDeviceCapabilities (page 90) (required)
- UIStatusBarStyle (page 94)
- UIInterfaceOrientation (page 86)
- UIRequiresPersistentWiFi (page 93)

For descriptions of these keys, see the other chapters of this book.

Recommended Keys for Cocoa Apps

It is recommended that a Cocoa app include the following keys in its information property list file. Most are set by Xcode automatically when you create your project but some may need to be added.

- CFBundleDevelopmentRegion (page 23)
- CFBundleDisplayName (page 24)
- CFBundleExecutable (page 32)
- CFBundleIconFile (page 33)
- CFBundleIdentifier (page 37)
- CFBundleInfoDictionaryVersion (page 37)
- CFBundleName (page 38)
- CFBundlePackageType (page 38)
- CFBundleShortVersionString (page 38)
- CFBundleSignature (page 38)
- CFBundleVersion (page 40)
- NSHumanReadableCopyright (page 57)

These keys identify your app to the system and provide some basic information about the services it provides. Cocoa apps should also include the following keys to identify key resources in the bundle:

- NSMainNibFile (page 59)
- NSPrincipalClass (page 61)

Note: If you are building a Cocoa app using an Xcode template, the NSMainNibFile (page 59) and NSPrincipalClass (page 61) keys are typically already set in the template project.

For descriptions of these keys, see the other chapters of this book.

Commonly Localized Keys

In addition to the recommended keys, there are several keys that should be localized and placed in your language-specific `InfoPlist.strings` files:

- CFBundleDisplayName (page 24)
- CFBundleName (page 38)
- CFBundleShortVersionString (page 38)
- NSHumanReadableCopyright (page 57)

For more information about localizing information property list keys, see Localizing Property List Values (page 15).

Core Foundation Keys

SwiftObjective-C

The Core Foundation framework provides the underlying infrastructure for bundles, including the code used at runtime to load bundles and parse their structure. As a result, many of the keys recognized by this framework are fundamental to the definition of bundles themselves and are instrumental in determining the contents of a bundle.

Core Foundation keys use the prefix `CF` to distinguish them from other keys. For more information about Core Foundation, see *Core Foundation Framework Reference* .

Key Summary

Table 1 contains an alphabetical listing of Core Foundation keys, the corresponding name for that key in the Xcode property list editor, a high-level description of each key, and the platforms on which you use it. Detailed information about each key is available in later sections.

Table 1 Summary of Core Foundation keys

Key	Xcode name	Summary	Platforms
CFAppleHelpAnchor	“Help file”	The bundle’s initial HTML help file. See CFAppleHelpAnchor (page 23) for details.	OS X
CFBundleAllowMixed-Localizations	“Localized resources can be mixed”	Used by Foundation tools to retrieve localized resources from frameworks. See CFBundleAllowMixedLocalizations (page 23) for details.	iOS, OS X
CFBundleDevelopment-Region	“Localization native development region”	(Recommended) The default language and region for the bundle, as a language ID. See CFBundleDevelopmentRegion (page 23) for details.	iOS, OS X
CFBundleDisplayName	“Bundle display name”	(Recommended, Localizable) The actual name of the bundle. See CFBundleDisplayName (page 24) for details.	iOS, OS X

Key	Xcode name	Summary	Platforms
CFBundleDocumentTypes	"Document types"	An array of dictionaries describing the document types supported by the bundle. See CFBundleDocumentTypes (page 24) for details.	iOS, OS X
CFBundleExecutable	"Executable file"	(Recommended) Name of the bundle's executable file. See CFBundleExecutable (page 32) for details.	iOS, OS X
CFBundleHelpBookFolder	"Help Book directory name"	The name of the folder containing the bundle's help files. See CFBundleHelpBookFolder (page 33) for details.	OS X
CFBundleHelpBookName	"Help Book identifier"	The name of the help file to display when Help Viewer is launched for the bundle. See CFBundleHelpBookName (page 33) for details.	OS X
CFBundleIconFile	"Icon file"	A legacy way to specify the app's icon. Use the CFBundleIcons (page 34) or CFBundleIconFiles (page 34) keys instead. See CFBundleIconFile (page 33) for details.	iOS, OS X
CFBundleIconFiles	"Icon files"	A top-level key for specifying the file names of the bundle's icon image files. See CFBundleIconFiles (page 34) for details. See also CFBundleIcons (page 34) as an alternative to this key.	iOS 3.2 and later
CFBundleIcons	None	File names of the bundle's icon image files. See CFBundleIconFiles (page 34) for details.	iOS 5.0 and later
CFBundleIdentifier	"Bundle identifier"	(Recommended) An identifier string that specifies the app type of the bundle. The string should be in reverse DNS format using only the Roman alphabet in upper and lower case (A–Z, a–z), the dot ("."), and the hyphen ("-"). See CFBundleIdentifier (page 37) for details.	iOS, OS X

Key	Xcode name	Summary	Platforms
CFBundleInfo-DictionaryVersion	“InfoDictionary version”	(Recommended) Version information for the `Info.plist` format. See CFBundleInfoDictionaryVersion (page 37) for details.	iOS, OS X
CFBundleLocalizations	“Localizations”	Contains localization information for an app that handles its own localized resources. See CFBundleLocalizations (page 37) for details.	iOS, OS X
CFBundleName	“Bundle name”	(Recommended, Localizable) The short display name of the bundle. See CFBundleName (page 38) for details.	iOS, OS X
CFBundlePackageType	“Bundle OS Type code”	The four-letter code identifying the bundle type. See CFBundlePackageType (page 38) for details.	iOS, OS X
CFBundleShortVersion-String	“Bundle versions string, short”	(Localizable) The release-version-number string for the bundle. See CFBundleShortVersionString (page 38) for details.	iOS, OS X
CFBundleSignature	“Bundle creator OS Type code”	The four-letter code identifying the bundle creator. See CFBundleSignature (page 38) for details.	iOS, OS X
CFBundleSpokenName	"Accessibility bundle name”	The spoken name of the app. See CFBundleSpokenName (page 39) for details.	iOS, OS X
CFBundleURLTypes	“URL types”	An array of dictionaries describing the URL schemes supported by the bundle. See CFBundleURLTypes (page 39) for details.	iOS, OS X
CFBundleVersion	“Bundle version”	(Recommended) The build-version-number string for the bundle. See CFBundleVersion (page 40) for details.	iOS, OS X
CFPlugInDynamic-Registration	“Plug-in should be registered dynamically”	If YES, register the plug-in dynamically; otherwise, register it statically. See CFPlugInDynamicRegistration (page 40) for details.	OS X

Key	Xcode name	Summary	Platforms
CFPlugInDynamic-RegistrationFunction	Plug-in dynamic registration function name"	The name of the custom, dynamic registration function. See CFPlugInDynamicRegisterFunction (page 40) for details.	OS X
CFPlugInFactories	"Plug-in factory interfaces"	For static registration, this dictionary contains a list of UUIDs with matching function names. See CFPlugInFactories (page 41) for details.	OS X
CFPlugInTypes	"Plug-in types"	For static registration, the list of UUIDs CFPlugInTypes (page 41) for details.	OS X
CFPlugInUnloadFunction	"Plug-in unload function name"	The name of the custom function to call when it's time to unload the plug-in code from memory. See CFPlugInUnloadFunction (page 41) for details.	OS X

CFAppleHelpAnchor

`CFAppleHelpAnchor` (`String` - OS X) identifies the name of the bundle's initial HTML help file, minus the `.html` or `.htm` extension. This file must be located in the bundle's localized resource directories or, if the help is not localized, directly under the `Resources` directory.

CFBundleAllowMixedLocalizations

`CFBundleAllowMixedLocalizations` (`Boolean` - iOS, OS X) specifies whether the bundle supports the retrieval of localized strings from frameworks. This key is used primarily by Foundation tools that link to other system frameworks and want to retrieve localized resources from those frameworks.

CFBundleDevelopmentRegion

`CFBundleDevelopmentRegion` (`String` - iOS, OS X) specifies the default language and region for the bundle, as a language ID. For example, English for the United Kingdom has the language ID `en–UK`. The system uses this value if it cannot locate a resource for the user's preferred language.

For more information, see Language IDs in *Internationalization and Localization Guide* . For details on how a bundle finds localized resources, see The Bundle Search Pattern in *Bundle Programming Guide* .

CFBundleDisplayName

`CFBundleDisplayName` (`String` - iOS, OS X) specifies the display name of the bundle. If you support localized names for your bundle, include this key in both your information property list file and in the `InfoPlist.strings` files of your language subdirectories. If you localize this key, you should also include a localized version of the `CFBundleName` key.

If you do not intend to localize your bundle, do not include this key in your `Info.plist` file. Inclusion of this key does not affect the display of the bundle name but does incur a performance penalty to search for localized versions of this key.

Before displaying a localized name for your bundle, the Finder compares the value of this key against the actual name of your bundle in the file system. If the two names match, the Finder proceeds to display the localized name from the appropriate `InfoPlist.strings` file of your bundle. If the names do not match, the Finder displays the file-system name.

For more information about display names in OS X, see *File System Programming Guide* .

CFBundleDocumentTypes

`CFBundleDocumentTypes` (`Array` - iOS, OS X) contains an array of dictionaries that associate one or more document types with your app. Each dictionary is called a type-definition dictionary and contains keys used to define the document type. Table 2 lists the keys that are supported in these dictionaries.

Table 2 Keys for type-definition dictionaries

Key	Xcode name	Type	Description	Platfor
CFBundleTypeExtensions	"Document Extensions"	`Array`	This key contains an array of strings. Each string contains a filename extension (minus the leading period) to map to this document type. To open documents with any extension, specify an extension with a single asterisk "`*`". (In OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in OS X v10.5.	OS X
CFBundleTypeIconFile	"Icon File Name"	`String`	This key contains a string with the name of the icon file (`.icns`) to associate with this OS X document type. For more information about specifying document icons, see Document Icons (page 31).	OS X
CFBundleTypeIconFiles	None	`Array`	An array of strings containing the names of the image files to use for the document icon in iOS. For more information about specifying document icons, see Document Icons (page 31).	iOS

Key	Xcode name	Type	Description	Platfor
CFBundleTypeMIMETypes	“Document MIME types”	`Array`	Contains an array of strings. Each string contains the MIME type name you want to map to this document type. (In OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in OS X v10.5.	OS X
CFBundleTypeName	“Document Type Name”	`String`	This key contains the abstract name for the document type and is used to refer to the type. This key is required and can be localized by including it in an `InfoPlist.strings` files. This value is the main way to refer to a document type. If you are concerned about this key being unique, you should consider using a uniform type identifier (UTI) for this string instead. If the type is a common Clipboard type supported by the system, you can use one of the standard types listed in the `NSPasteboard` class description.	iOS, OS

Key	Xcode name	Type	Description	Platfor
CFBundleTypeOSTypes	"Document OS Types"	`Array`	This key contains an array of strings. Each string contains a four-letter type code that maps to this document type. To open documents of any type, include four asterisk characters (`****`) as the type code. These codes are equivalent to the legacy type codes used by Mac OS 9. (In OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in OS X v10.5.	OS X
CFBundleTypeRole	"Role"	`String`	This key specifies the app's role with respect to the type. The value can be `Editor`, `Viewer`, `Shell`, or `None`. This key is required.	OS X

Key	Xcode name	Type	Description	Platfor
LSItemContentTypes	“Document Content Type UTIs”	`Array`	This key contains an array of strings. Each string contains a UTI defining a supported file type. The UTI string must be spelled out explicitly, as opposed to using one of the constants defined by Launch Services. For example, to support PNG files, you would include the string “`public.png`“ in the array. When using this key, also add the `NSExportableTypes` key with the appropriate entries. In OS X v10.5 and later, this key (when present) takes precedence over these type-identifier keys: `CFBundleType-Extensions`, `CFBundleType-MIMETypes`, `CFBundleTypeOSTypes`.	iOS, OS

Key	Xcode name	Type	Description	Platfor
LSHandlerRank	“Handler rank”	`String`	Determines how Launch Services ranks this app among the apps that declare themselves editors or viewers of files of this type. The possible values are: `Owner` (this app is the creator of files of this type), `Alternate` (this app is a secondary viewer of files of this type), `None` (this app must never be used to open files of this type, but it accepts drops of files of this type), `Default` (default; this app doesn’t accept drops of files of this type). Launch Services uses the value of `LSHandlerRank` to determine the app to use to open files of this type. The order of precedence is: `Owner`, `Alternate`, `None`. This key is available in OS X v10.5 and later.	iOS, OS
LSTypeIsPackage	“Document is a package or bundle”	`Boolean`	Specifies whether the document is distributed as a bundle. If set to true, the bundle directory is treated as a file. (In OS X v10.4 and later, this key is ignored if the `LSItemContentTypes` key is present.)	OS X
NSDocumentClass	“Cocoa NSDocument Class”	`String`	This key specifies the name of the `NSDocument` subclass used to instantiate instances of this document. This key is used by Cocoa apps only.	OS X

Key	Xcode name	Type	Description	Platfor
NSUbiquitousDocumentUserActivityType	None	`String`	This key specifies the activity type of the `NSUserActivity` object associated with this document.	iOS, OS
NSExportableAs	“Exportable As Document Type Names”	`Array`	This key specifies an array of strings. Each string contains the name of another document type, that is, the value of a `CFBundleTypeName` property. This value represents another data format to which this document can export its content. This key is used by Cocoa apps only. Deprecated in OS X v10.5.	OS X
NSExportableTypes	None	`Array`	This key specifies an array strings. Each string should contain a UTI defining a supported file type to which this document can export its content. Each UTI string must be spelled out explicitly, as opposed to using one of the constants defined by Launch Services. For example, to support PNG files, you would include the string “`public.png`“ in the array. This key is used by Cocoa apps only. Available in OS X v10.5 and later.	OS X

The way you specify icon files in OS X and iOS is different because of the supported file formats on each platform. In iOS, each icon resource file is typically a PNG file that contains only one image. Therefore, it is necessary to specify different image files for different icon sizes. However, when specifying icons in OS X, you use an icon file (with extension `.icns`), which is capable of storing the icon at several different resolutions.

This key is supported in iOS 3.2 and later and all versions of OS X. For detailed information about UTIs, see *Uniform Type Identifiers Overview*.

Document Roles

An app can take one of the following roles for any given document type:

- **Editor**. The app can read, manipulate, and save the type.
- **Viewer**. The app can read and present data of that type.
- **Shell**. The app provides runtime services for other processes—for example, a Java applet viewer. The name of the document is the name of the hosted process (instead of the name of the app), and a new process is created for each document opened.
- **None**. The app does not understand the data, but is just declaring information about the type (for example, the Finder declaring an icon for fonts).

The role you choose applies to all of the concrete formats associated with the document or Clipboard type. For example, the Safari app associates itself as a viewer for documents with the ".html", ".htm", "shtml, or "jhtml" filename extensions. Each of these extensions represents a concrete type of document that falls into the overall category of HTML documents. This same document can also support MIME types and legacy 4-byte OS types.

Document Icons

In iOS, the `CFBundleTypeIconFiles` key contains an array of strings with the names of the image files to use for the document icon. Table 3 lists the icon sizes you can include for each device type. You can name the image files however you want but the file names in your `Info.plist` file must match the image resource filenames exactly. (For iPhone and iPod touch, the usable area of your icon is actually much smaller.) For more information on how to create these icons, see *iOS Human Interface Guidelines*.

Table 3 Document icon sizes for iOS

Device	Sizes
iPad	64 x 64 pixels 320 x 320 pixels

Device	Sizes
iPhone and iPod touch	22 x 29 pixels 44 x 58 pixels (high resolution)

In OS X, the `CFBundleTypeIconFile` key contains the name of an icon resource file with the document icon. An icon resource file contains multiple images, each representing the same document icon at different resolutions. If you omit the filename extension, the system looks for your file with the extension `.icns`. You can create icon resource files using the Icon Composer app that comes with Xcode Tools.

Recommended Keys

The entry for each document type should contain the following keys:

- `CFBundleTypeIconFile`
- `CFBundleTypeName`
- `CFBundleTypeRole`

In addition to these keys, it must contain at least one of the following keys:

- `LSItemContentTypes`
- `CFBundleTypeExtensions`
- `CFBundleTypeMIMETypes`
- `CFBundleTypeOSTypes`

If you do not specify at least one of these keys, no document types are bound to the type-name specifier. You may use all three keys when binding your document type, if you so choose. In OS X v10.4 and later, if you specify the `LSItemContentTypes` key, the other keys are ignored. You can continue to include the other keys for compatibility with older versions of the system, however.

CFBundleExecutable

`CFBundleExecutable` (`String` - iOS, OS X) identifies the name of the bundle’s main executable file. For an app, this is the app executable. For a loadable bundle, it is the binary that will be loaded dynamically by the bundle. For a framework, it is the shared library for the framework. Xcode automatically adds this key to the information property list file of appropriate projects.

For frameworks, the value of this key is required to be the same as the framework name, minus the `.framework` extension. If the keys are not the same, the target system may incur some launch-performance penalties. The value should not include any extension on the name.

Important: You must include a valid `CFBundleExecutable` key in your bundle's information property list file. OS X uses this key to locate the bundle's executable or shared library in cases where the user renames the app or bundle directory.

CFBundleHelpBookFolder

`CFBundleHelpBookFolder` (`String` - OS X) identifies the folder containing the bundle's help files. Help is usually localized to a specific language, so the folder specified by this key represents the folder name inside the `.lproj` directory for the selected language.

CFBundleHelpBookName

`CFBundleHelpBookName` (`String` - OS X) identifies the main help page for your app. This key identifies the name of the Help page, which may not correspond to the name of the HTML file. The Help page name is specified in the `CONTENT` attribute of the help file's `META` tag.

CFBundleIconFile

`CFBundleIconFile` (`String` - iOS, OS X) identifies the file containing the icon for the bundle. The filename you specify does not need to include the extension, although it may. The system looks for the icon file in the main resources directory of the bundle.

If your Mac app uses a custom icon, you must specify this property. If you do not specify this property, the system (and other apps) display your bundle with a default icon.

Note: If you are writing an iOS app, you should prefer the use of the CFBundleIconFiles (page 34) key over this one.

CFBundleIconFiles

`CFBundleIconFiles` (`Array` - iOS) contains an array of strings identifying the icon files for the bundle. (It is recommended that you always create icon files using the PNG format.) When specifying your icon filenames, it is best to omit any filename extensions. Omitting the filename extension lets the system automatically detect high-resolution (`@2x`) versions of your image files using the standard-resolution image filename. If you include filename extensions, you must specify all image files (including the high-resolution variants) explicitly. The system looks for the icon files in the main resources directory of the bundle. If present, the values in this key take precedence over the value in the CFBundleIconFile (page 33) key.

This key is supported in iOS 3.2 and later only and an app may have differently sized icons to support different types of devices and different screen resolutions. In other words, an app icon is typically 57 x 57 pixels on iPhone or iPod touch but is 72 x 72 pixels on iPad. Icons at other sizes may also be included. The order of the items in this array does not matter. The system automatically chooses the most appropriately sized icon based on the usage and the underlying device type.

For information about how to create icons for your apps, including the size information for each one, see *iOS Human Interface Guidelines* .

CFBundleIcons

`CFBundleIcons` (`Dictionary` - iOS) contains information about all of the icons used by the app. This key allows you to group icons based on their intended usage and specify multiple icon files together with specific keys for modifying the appearance of those icons. This dictionary can contain the following keys:

- **CFBundlePrimaryIcon**—This key identifies the icons for the Home screen and Settings app among others. The value for this key is a dictionary whose contents are described in Contents of the CFBundlePrimaryIcon Dictionary (page 35).
- **UINewsstandIcon**—This key identifies default icons to use for apps presented from Newsstand. The value for this key is a dictionary whose contents are described in Contents of the UINewsstandIcon Dictionary (page 35).

The `CFBundleIcons` key is supported in iOS 5.0 and later. You may combine this key with the CFBundleIconFiles (page 34) and CFBundleIconFile (page 33) keys but in iOS 5.0 and later, this key takes precedence.

Contents of the CFBundlePrimaryIcon Dictionary

The value for the `CFBundlePrimaryIcon` key is a dictionary that identifies the icons associated with the app bundle. The icons in this dictionary are used to represent the app on the device's Home screen and in the Settings app. Table 4 lists the keys that you can include in this dictionary and their values.

Table 4 Keys for the `CFBundlePrimaryIcon` dictionary

Key	Value	Description
CFBundleIconFiles	Array of strings	(Required) Each string in the array contains the name of an icon file. You can include multiple icons of different sizes to support iPhone, iPad, and universal apps. For a list of the icons, including their sizes, that you can include in your app bundle, see the section on app icons in Advanced App Tricks in *App Programming Guide for iOS* . For information about how to create icons for your apps, see *iOS Human Interface Guidelines* .
UIPrerenderedIcon	Boolean	This key specifies whether the icon files already incorporate a shine effect. If your icons already incorporate this effect, include the key and set its value to `YES` to prevent the system from adding the same effect again. If you do not include this key, or set its value to `NO`, the system applies a shine effect to the icon files listed in the `CFBundleIconFiles` key in this dictionary.

When specifying icon filenames, it is best to omit any filename extensions. Omitting the filename extension lets the system automatically detect high-resolution (`@2x`) versions of your image files using the standard-resolution image filename. If you include filename extensions, you must specify all image files (including the high-resolution variants) explicitly. The system looks for the icon files in the main resources directory of the bundle.

Contents of the UINewsstandIcon Dictionary

The value for the `UINewsstandIcon` key is a dictionary that identifies the default icons and style options to use for apps displayed in Newsstand. Table 5 lists the keys that you can include in this dictionary and their values.

Table 5 Keys for the `UINewsstandIcon` dictionary

Key	Value	Description
CFBundleIconFiles	Array of strings	(Required) Each string in the array contains the name of an icon file. You use this key to specify a set of standard icons for your app when presented in the Newsstand. This icon is used when no cover art is available for a downloaded issue. For a list of the icons, including their sizes, that you can include in your app bundle, see the section on app icons in App-Related Resources in *App Programming Guide for iOS*. For information about how to create icons for your apps, see *iOS Human Interface Guidelines*.

UINewsstandBindingType	String	This key provides information about how to stylize any Newsstand art. The value of this key is one of the following strings:
		`UINewsstandBindingTypeMagazine`
		`UINewsstandBindingTypeNewspaper`

UINewsstandBindingEdge	String	This key provides information about how to stylize any Newsstand art. The value of this key is one of the following strings:
		`UINewsstandBindingEdgeLeft`
		`UINewsstandBindingEdgeRight`
		`UINewsstandBindingEdgeBottom`

When specifying icon filenames, it is best to omit any filename extensions. Omitting the filename extension lets the system automatically detect high-resolution (`@2x`) versions of your image files using the standard-resolution image filename. If you include filename extensions, you must specify all image files (including the high-resolution variants) explicitly. The system looks for the icon files in the main resources directory of the bundle.

CFBundleIdentifier

`CFBundleIdentifier` (`String` - iOS, OS X) uniquely identifies the bundle. Each distinct app or bundle on the system must have a unique bundle ID. The system uses this string to identify your app in many ways. For example, the preferences system uses this string to identify the app for which a given preference applies; Launch Services uses the bundle identifier to locate an app capable of opening a particular file, using the first app it finds with the given identifier; in iOS, the bundle identifier is used in validating the app's signature.

The bundle ID string must be a uniform type identifier (UTI) that contains only alphanumeric (`A`-`Z`,`a`-`z`,`0`-`9`), hyphen (-), and period (`.`) characters. The string should also be in reverse-DNS format. For example, if your company's domain is `Ajax.com` and you create an app named Hello, you could assign the string `com.Ajax.Hello` as your app's bundle identifier.

Note: Although formatted similarly to a UTI, the character set for a bundle identifier is more restrictive.

CFBundleInfoDictionaryVersion

`CFBundleInfoDictionaryVersion` (`String` - iOS, OS X) identifies the current version of the property list structure. This key exists to support future versioning of the information property list file format. Xcode generates this key automatically when you build a bundle and you should not change it manually. The value for this key is currently 6.0.

CFBundleLocalizations

`CFBundleLocalizations` (`Array` - iOS, OS X) identifies the localizations handled manually by your app. If your executable is unbundled or does not use the existing bundle localization mechanism, you can include this key to specify the localizations your app does handle.

Each entry in this property's array is a string identifying the language name or ISO language designator of the supported localization. See "Language and Locale Designations" in *Internationalization and Localization Guide* in Internationalization Documentation for information on how to specify language designators.

CFBundleName

`CFBundleName` (`String` - iOS, OS X) identifies the short name of the bundle. This name should be less than 16 characters long and be suitable for displaying in the menu bar and the app's Info window. You can include this key in the `InfoPlist.strings` file of an appropriate `.lproj` subdirectory to provide localized values for it. If you localize this key, you should also include the key CFBundleDisplayName (page 24).

CFBundlePackageType

`CFBundlePackageType` (`String` - iOS, OS X) identifies the type of the bundle and is analogous to the Mac OS 9 file type code. The value for this key consists of a four-letter code. The type code for apps is `APPL`; for frameworks, it is `FMWK`; for loadable bundles, it is `BNDL`. For loadable bundles, you can also choose a type code that is more specific than `BNDL` if you want.

All bundles should provide this key. However, if this key is not specified, the bundle routines use the bundle extension to determine the type, falling back to the `BNDL` type if the bundle extension is not recognized.

CFBundleShortVersionString

`CFBundleShortVersionString` (`String` - iOS, OS X) specifies the *release version number* of the bundle, which identifies a released iteration of the app. The release version number is a string comprised of three period-separated integers. The first integer represents major revisions to the app, such as revisions that implement new features or major changes. The second integer denotes revisions that implement less prominent features. The third integer represents maintenance releases.

The value for this key differs from the value for CFBundleVersion (page 40), which identifies an iteration (released or unreleased) of the app. This key can be localized by including it in your `InfoPlist.strings` files.

CFBundleSignature

`CFBundleSignature` (`String` - iOS, OS X) identifies the creator of the bundle and is analogous to the Mac OS 9 file creator code. The value for this key is a string containing a four-letter code that is specific to the bundle. For example, the signature for the TextEdit app is `ttxt`.

CFBundleSpokenName

`CFBundleSpokenName` (`String` - iOS, OS X) contains a suitable replacement for the app name when performing text-to-speech operations. Include this key in your app bundle when the spelling of your app might be mispronounced by the speech system. For example, if the name of your app is “MyApp123”, you might set the value of this key to “My app one two three”.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

CFBundleURLTypes

`CFBundleURLTypes` (`Array` - iOS, OS X) contains an array of dictionaries, each of which describes the URL schemes (`http`, `ftp`, and so on) supported by the app. The purpose of this key is similar to that of CFBundleDocumentTypes (page 24), but it describes URL schemes instead of document types. Each dictionary entry corresponds to a single URL scheme. Table 6 lists the keys to use in each dictionary entry.

Table 6 Keys for CFBundleURLTypes dictionaries

Key	Xcode name	Type	Description	Platforms
CFBundleTypeRole	“Document Role”	`String`	This key specifies the app’s role with respect to the URL type. The value can be `Editor`, `Viewer`, `Shell`, or `None`. This key is required.	iOS, OS X
CFBundleURLIconFile	“Document Icon File Name”	`String`	This key contains the name of the icon image file (minus the extension) to be used for this URL type.	iOS, OS X
CFBundleURLName	“URL identifier”	`String`	This key contains the abstract name for this URL type. This is the main way to refer to a particular type. To ensure uniqueness, it is recommended that you use a Java-package style identifier. This name is also used as a key in the `InfoPlist.strings` file to provide the human-readable version of the type name.	iOS, OS X

Key	Xcode name	Type	Description	Platforms
CFBundleURLSchemes	"URL Schemes"	`Array`	This key contains an array of strings, each of which identifies a URL scheme handled by this type. For example, specifying the URL scheme `feed` makes other apps aware that this app is capable of viewing RSS content. Types of URL schemes include `http`, `ftp`, `mailto`, and so on.	iOS, OS X

CFBundleVersion

`CFBundleVersion` (`String` - iOS, OS X) specifies the *build version number* of the bundle, which identifies an iteration (released or unreleased) of the bundle. The build version number should be a string comprised of three non-negative, period-separated integers with the first integer being greater than zero. The string should only contain numeric (`0-9`) and period (`.`) characters. Leading zeros are truncated from each integer and will be ignored (that is, `1.02.3` is equivalent to `1.2.3`). This key is not localizable.

CFPlugInDynamicRegistration

`CFPlugInDynamicRegistration` (`String` - OS X) specifies whether how host loads this plug-in. If the value is `YES`, the host attempts to load this plug-in using its dynamic registration function. If the value is `NO`, the host uses the static registration information included in the CFPlugInFactories (page 41), and CFPlugInTypes (page 41) keys.

For information about registering plugins, see "Plug-in Registration" in *Plug-in Programming Topics* .

CFPlugInDynamicRegisterFunction

`CFPlugInDynamicRegisterFunction` (`String` - OS X) identifies the function to use when dynamically registering a plug-in. Specify this key if you want to specify one of your own functions instead of implement the default `CFPlugInDynamicRegister` function.

For information about registering plugins, see “Plug-in Registration” in *Plug-in Programming Topics* .

CFPlugInFactories

`CFPlugInFactories` (`Dictionary` - OS X) is used for static plug-in registration. It contains a dictionary identifying the interfaces supported by the plug-in. Each key in the dictionary is a universally unique ID (UUID) representing the supported interface. The value for the key is a string with the name of the plug-in factory function to call.

For information about registering plugins, see “Plug-in Registration” in *Plug-in Programming Topics* .

CFPlugInTypes

`CFPlugInTypes` (`Dictionary` - OS X) is used for static plug-in registration. It contains a dictionary identifying one or more groups of interfaces supported by the plug-in. Each key in the dictionary is a universally unique ID (UUID) representing the group of interfaces. The value for the key is an array of strings, each of which contains the UUID for a specific interface in the group. The UUIDs in the array corresponds to entries in the CFPlugInFactories (page 41) dictionary.

For information about registering plugins, see “Plug-in Registration” in *Plug-in Programming Topics* .

CFPlugInUnloadFunction

`CFPlugInUnloadFunction` (`String` - OS X) specifies the name of the function to call when it is time to unload the plug-in code from memory. This function gives the plug-in an opportunity to clean up any data structures it allocated.

For information about registering plugins, see “Plug-in Registration” in *Plug-in Programming Topics* .

Launch Services Keys

SwiftObjective-C

Launch Services (part of the Core Services framework in OS X) provides support for launching apps and matching document types to apps. As a result, the keys recognized by Launch Services allow you to specify the desired execution environment for your bundled code. (In iOS, Launch Services is a private API but is still used internally to coordinate the execution environment of iOS apps.)

Launch Services keys use the prefix LS to distinguish them from other keys. For more information about Launch Services in OS X, see *Launch Services Programming Guide* and *Launch Services Reference*.

Key Summary

Table 1 contains an alphabetical listing of Launch Services keys, the corresponding name for that key in the Xcode property list editor, a high-level description of each key, and the platforms on which you use it. Detailed information about each key is available in later sections.

Table 1 Summary of Launch Services keys

Key	Xcode name	Summary	Platforms
LSApplication-CategoryType	"Application Category"	Contains a string with the UTI that categorizes the app for the App Store. See LSApplicationCategoryType (page 44) for details.	OS X
LSArchitecturePriority	"Architecture priority"	Contains an array of strings identifying the supported code architectures and their preferred execution priority. See LSArchitecturePriority (page 46) for details.	OS X
LSBackgroundOnly	"Application is background only"	Specifies whether the app runs only in the background. (Mach-O apps only). See LSBackgroundOnly (page 47) for details.	OS X
LSEnvironment	"Environment variables"	Contains a list of key/value pairs, representing environment variables and their values. See LSEnvironment (page 47) for details.	OS X

Key	Xcode name	Summary	Platforms
LSFileQuarantine-Enabled	"File quarantine enabled"	Specifies whether the files this app creates are quarantined by default. See LSFileQuarantineEnabled (page 47).	OS X
LSFileQuarantine-ExcludedPathPatterns	None	Specifies directories for which files should not be automatically quarantined. See LSFileQuarantineExcludedPathPatterns (page 47).	OS X
LSGetAppDiedEvents	"Application should get App Died events"	Specifies whether the app is notified when a child process dies. See LSGetAppDiedEvents (page 48) for details.	OS X
LSMinimumSystem-Version	"Minimum system version"	Specifies the minimum version of OS X required for the app to run. See LSMinimumSystemVersion (page 48) for details.	OS X
LSMinimumSystem-VersionByArchitecture	"Minimum system versions, per-architecture"	Specifies the minimum version of OS X required to run a given platform architecture. See LSMinimumSystemVersionByArchitecture (page 48) for details.	OS X
LSMultipleInstances-Prohibited	"Application prohibits multiple instances"	Specifies whether one user or multiple users can launch an app simultaneously. See LSMultipleInstancesProhibited (page 49) for details.	OS X
LSRequiresIPhoneOS	"Application requires iPhone environment"	Specifies whether the app is an iOS app. See LSRequiresIPhoneOS (page 49) for details.	iOS
LSRequiresNative-Execution	"Application requires native environment"	Specifies whether the app must run natively on an Intel-based Mac, as opposed to under Rosetta emulation. See LSRequiresNativeExecution (page 49) for details.	OS X
LSUIElement	"Application is agent (UIElement)"	Specifies whether the app is an agent app, that is, an app that should not appear in the Dock or Force Quit window. See LSUIElement (page 49) for details.	OS X
LSUIPresentationMode	"Application UI Presentation Mode"	Sets the visibility of system UI elements when the app launches. See LSUIPresentationMode (page 50) for details.	OS X

Key	Xcode name	Summary	Platforms
LSVisibleInClassic	"Application is visible in Classic"	Specifies whether an agent app or background-only app is visible to other apps in the Classic environment. See LSVisibleInClassic (page 50) for details.	OS X
MinimumOSVersion	"Minimum system version"	Do not use. Use LSMinimumSystemVersion (page 48) instead.	iOS

LSApplicationCategoryType

`LSApplicationCategoryType` (`String` - OS X) is a string that contains the UTI corresponding to the app's type. The App Store uses this string to determine the appropriate categorization for the app. Table 2 lists the supported UTIs for apps.

Table 2 UTIs for app categories

Category	UTI
Business	`public.app-category.business`
Developer Tools	`public.app-category.developer-tools`
Education	`public.app-category.education`
Entertainment	`public.app-category.entertainment`
Finance	`public.app-category.finance`
Games	`public.app-category.games`
Graphics & Design	`public.app-category.graphics-design`
Healthcare & Fitness	`public.app-category.healthcare-fitness`
Lifestyle	`public.app-category.lifestyle`
Medical	`public.app-category.medical`
Music	`public.app-category.music`
News	`public.app-category.news`
Photography	`public.app-category.photography`

Category	UTI
Productivity	`public.app-category.productivity`
Reference	`public.app-category.reference`
Social Networking	`public.app-category.social-networking`
Sports	`public.app-category.sports`
Travel	`public.app-category.travel`
Utilities	`public.app-category.utilities`
Video	`public.app-category.video`
Weather	`public.app-category.weather`

Table 3 lists the UTIs that are specific to games.

Table 3 UTIs for game-specific categories

Category	UTI
Action Games	`public.app-category.action-games`
Adventure Games	`public.app-category.adventure-games`
Arcade Games	`public.app-category.arcade-games`
Board Games	`public.app-category.board-games`
Card Games	`public.app-category.card-games`
Casino Games	`public.app-category.casino-games`
Dice Games	`public.app-category.dice-games`
Educational Games	`public.app-category.educational-games`
Family Games	`public.app-category.family-games`
Kids Games	`public.app-category.kids-games`
Music Games	`public.app-category.music-games`
Puzzle Games	`public.app-category.puzzle-games`

Category	**UTI**
Racing Games	`public.app-category.racing-games`
Role Playing Games	`public.app-category.role-playing-games`
Simulation Games	`public.app-category.simulation-games`
Sports Games	`public.app-category.sports-games`
Strategy Games	`public.app-category.strategy-games`
Trivia Games	`public.app-category.trivia-games`
Word Games	`public.app-category.word-games`

LSArchitecturePriority

`LSArchitecturePriority` (`Array` - OS X) is an array of strings that identifies the architectures this app supports. The order of the strings in this array dictates the preferred execution priority for the architectures. The possible strings for this array are listed in Table 4.

Table 4 Execution architecture identifiers

String	**Description**
`i386`	The 32-bit Intel architecture.
`ppc`	The 32-bit PowerPC architecture.
`x86_64`	The 64-bit Intel architecture.
`ppc64`	The 64-bit PowerPC architecture.

If a PowerPC architecture appears before either of the Intel architectures, OS X runs the executable under Rosetta emulation on an Intel-based Mac. To force OS X to use the current platform's native architecture, include the LSRequiresNativeExecution (page 49) key in your information property list.

LSBackgroundOnly

`LSBackgroundOnly` (`Boolean` - OS X) specifies whether this app runs only in the background. If this key exists and is set to "1", Launch Services runs the app in the background only. You can use this key to create faceless background apps. You should also use this key if your app uses higher-level frameworks that connect to the window server, but are not intended to be visible to users. Background apps must be compiled as Mach-O executables. This option is not available for CFM apps.

LSEnvironment

`LSEnvironment` (`Dictionary` - OS X) defines environment variables to be set before launching this app. The names of the environment variables are the keys of the dictionary, with the values being the corresponding environment variable value. Both keys and values must be strings.

These environment variables are set only for apps launched through Launch Services. If you run your executable directly from the command line, these environment variables are not set.

LSFileQuarantineEnabled

`LSFileQuarantineEnabled` (`Boolean` - OS X) specifies whether files this app creates are quarantined by default.

Value	Description
`true`	Files created by this app are quarantined by default. When quarantining files, the system automatically associates the timestamp, app name, and the bundle identifier with the quarantined file whenever possible. Your app can also get or set quarantine attributes as needed using Launch Services.
`false`	(Default) Files created by this app are not quarantined by default.

This key is available in OS X v10.5 and later.

LSFileQuarantineExcludedPathPatterns

`LSFileQuarantineExcludedPathPatterns` (`Array` - OS X) contains an array of strings indicating the paths for which you want to disable file quarantining. You can use this key to prevent file quarantines from affecting the performance of your app. Each string in the array is a shell-style path pattern, which means that special characters such as `~`, `*`, and `?` are automatically expanded according to the standard command-line rules. For example, a string of the form `~/Library/Caches/*` would allow you to disable the quarantine for files created by your app in the user’s cache directory.

LSGetAppDiedEvents

`LSGetAppDiedEvents` (`Boolean` - OS X) indicates whether the operation system notifies this app when when one of its child process terminates. If you set the value of this key to `YES`, the system sends your app an `kAEApplicationDied` Apple event for each child process as it terminates.

LSMinimumSystemVersion

`LSMinimumSystemVersion` (`String` - OS X) indicates the minimum version of OS X required for this app to run. This string must be of the form *n*.*n*.*n* where *n* is a number. The first number is the major version number of the system. The second and third numbers are minor revision numbers. For example, to support OS X v10.4 and later, you would set the value of this key to “10.4.0”.

If the minimum system version is not available, OS X tries to display an alert panel notifying the user of that fact.

LSMinimumSystemVersionByArchitecture

`LSMinimumSystemVersionByArchitecture` (`Dictionary` - OS X) specifies the earliest OS X version for a set of architectures. This key contains a dictionary of key-value pairs. Each key corresponds to one of the architectures associated with the LSArchitecturePriority (page 46) key. The value for each key is the minimum version of OS X required for the app to run under that architecture. This string must be of the form *n*.*n*.*n* where *n* is a number. The first number is the major version number of the system. The second and third numbers are minor revision numbers. For example, to support OS X v10.4.9 and later, you would set the value of this key to “10.4.9”.

If the current system version is less than the required minimum version, Launch Services does not attempt to use the corresponding architecture. This key applies only to the selection of an execution architecture and can be used in conjunction with the LSMinimumSystemVersion (page 48) key, which specifies the overall minimum system version requirement for the app.

LSMultipleInstancesProhibited

`LSMultipleInstancesProhibited` (`Boolean` - OS X) indicates whether an app is prohibited from running simultaneously in multiple user sessions. If true, the app runs in only one user session at a time. You can use this key to prevent resource conflicts that might arise by sharing an app across multiple user sessions. For example, you might want to prevent users from accessing a custom USB device when it is already in use by a different user.

Launch Services returns an appropriate error code if the target app cannot be launched. If a user in another session is running the app, Launch Services returns a `kLSMultipleSessionsNotSupportedErr` error. If you attempt to launch a separate instance of an app in the current session, it returns `kLSMultipleInstancesProhibitedErr`.

LSRequiresIPhoneOS

`LSRequiresIPhoneOS` (`Boolean` - iOS) specifies whether the app can run only on iOS. If this key is set to `YES`, Launch Services allows the app to launch only when the host platform is iOS.

LSRequiresNativeExecution

`LSRequiresNativeExecution` (`Boolean` - OS X) specifies whether to launch the app using the subbinary for the current architecture. If this key is set to `YES`, Launch Services always runs the app using the binary compiled for the current architecture. You can use this key to prevent a universal binary from being run under Rosetta emulation on an Intel-based Mac. For more information about configuring the execution architectures, see LSArchitecturePriority (page 46).

LSUIElement

`LSUIElement` (`String` - OS X) specifies whether the app runs as an agent app. If this key is set to "1", Launch Services runs the app as an agent app. Agent apps do not appear in the Dock or in the Force Quit window. Although they typically run as background apps, they can come to the foreground to present a user interface if desired. A click on a window belonging to an agent app brings that app forward to handle events.

The Dock and loginwindow are two apps that run as agent apps.

LSUIPresentationMode

`LSUIPresentationMode` (`Number` - OS X) identifies the initial user-interface mode for the app. You would use this in apps that may need to take over portions of the screen that contain UI elements such as the Dock and menu bar. Most modes affect only UI elements that appear in the content area of the screen, that is, the area of the screen that does not include the menu bar. However, you can request that all UI elements be hidden as well.

This key is applicable to both Carbon and Cocoa apps and can be one of the following values:

Value	Description
0	Normal mode. In this mode, all standard system UI elements are visible. This is the default value.
1	Content suppressed mode. In this mode, system UI elements in the content area of the screen are hidden. UI elements may show themselves automatically in response to mouse movements or other user activity. For example, the Dock may show itself when the mouse moves into the Dock's auto-show region.
2	Content hidden mode. In this mode, system UI elements in the content area of the screen are hidden and do not automatically show themselves in response to mouse movements or user activity.
3	All hidden mode. In this mode, all UI elements are hidden, including the menu bar. Elements do not automatically show themselves in response to mouse movements or user activity.
4	All suppressed mode. In this mode, all UI elements are hidden, including the menu bar. UI elements may show themselves automatically in response to mouse movements or other user activity. This option is available only in OS X v10.3 and later.

LSVisibleInClassic

`LSVisibleInClassic` (`String` - OS X). If this key is set to "1", any agent apps or background-only apps with this key appears as background-only processes to the Classic environment. Agent apps and background-only apps without this key do not appear as running processes to Classic at all. Unless your process needs to communicate explicitly with a Classic app, you do not need to include this key.

MinimumOSVersion

`MinimumOSVersion` (`String` - iOS). When you build an iOS app, Xcode uses the iOS Deployment Target setting of the project to set the value for the `MinimumOSVersion` key. Do *not* specify this key yourself in the `Info.plist` file; it is a system-written key. When you publish your app to the App Store, the store indicates the iOS releases on which your app can run based on this key. It is equivalent to the `LSMinimumSystemVersion` key in OS X.

Cocoa Keys

Objective-CSwift
Cocoa and Cocoa Touch are the environments used to define Objective-C based apps that run in OS X and iOS respectively. The keys associated with the Cocoa environments provide support for Interface Builder nib files and provide support for other user-facing features vended by your bundle.

Cocoa keys use the prefix NS to distinguish them from other keys. For information about developing Cocoa Touch apps for iOS, see *App Programming Guide for iOS* . For information about developing Cocoa apps for OS X, see *Cocoa Fundamentals Guide* .

Key Summary

Table 1 contains an alphabetical listing of Cocoa keys, the corresponding name for that key in the Xcode property list editor, a high-level description of each key, and the platforms on which you use it. Detailed information about each key is available in later sections.

Table 1 Summary of Cocoa keys

Key	Xcode name	Summary	Platform
GKGameCenterBadging-Disabled	None	Specifies whether your app is badged. See GKGameCenterBadgingDisabled (page 55) for details.	iOS 7.0 and late
GKShowChallenge-Banners	None	Specifies whether banners are shown within an app. See GKShowChallengeBanners (page 56)for details.	iOS 7.0 and late
NSAppleScriptEnabled	“Scriptable”	Specifies whether AppleScript is enabled. See NSAppleScriptEnabled (page 56) for details.	OS X
NSBluetooth-PeripheralUsage-Description	“Privacy - Bluetooth Peripheral Usage Description”	Specifies the reason for using Bluetooth. See NSBluetoothPeripheralUsageDescription (page 56) for details.	iOS 6.0 and late
NSCalendarsUsage-Description	“Privacy - Calendars Usage Description”	Specifies the reason for accessing the user’s calendars. See NSCalendarsUsageDescription (page 56) for details.	iOS 6.0 and late

Key	Xcode name	Summary	Platform
NSCameraUsage-Description	"Privacy - Camera Usage Description"	Specifies the reason for accessing the device's camera. See NSLocationUsageDescription (page 58) for details.	iOS 7.0 and late
NSContactsUsage-Description	"Privacy - Contacts Usage Description"	Specifies the reason for accessing the user's contacts. See NSContactsUsageDescription (page 57) for details.	iOS 6.0 and late OS X v10.8 an later
NSDockTilePlugIn	"Dock Tile Plugin path"	Specifies the name of app's Dock tile plug-in, if present. See NSDockTilePlugIn (page 57) for details.	OS X
NSHumanReadable-Copyright	"Copyright (human-readable)"	(Localizable) Specifies the copyright notice for the bundle. See NSHumanReadableCopyright (page 57) for details.	OS X
NSJavaNeeded	"Cocoa Java application"	Specifies whether the program requires a running Java VM. See NSJavaNeeded (page 57) for details.	OS X
NSJavaPath	"Java classpaths"	An array of paths to classes whose components are preceded by `NSJavaRoot`. See NSJavaPath (page 57) for details.	OS X
NSJavaRoot	"Java root directory"	The root directory containing the java classes. See NSJavaRoot (page 58) for details.	OS X
NSLocationAlways-UsageDescription	None	Specifies the reason for accessing the user's location information. See NSLocationAlwaysUsageDescription (page 58) for details.	iOS 8.0 and late OS X v10.10 and late
NSLocationUsage-Description	"Privacy - Location Usage Description"	Specifies the reason for accessing the user's location information. See NSLocationUsageDescription (page 58) for details.	iOS 6.0 and late OS X v10.9 an later
NSLocationWhenInUse-UsageDescription	None	Specifies the reason for accessing the user's location information. See NSLocationWhenInUseUsageDescription (page 59) for details.	iOS 8.0 and late OS X v10.10 and late

Key	Xcode name	Summary	Platform
NSMainNibFile	"Main nib file base name"	The name of an app's main nib file. See NSMainNibFile (page 59) for details.	iOS, OS
NSMicrophoneUsage-Description	"Privacy - Microphone Usage Description"	Specifies the reason for accessing the device's microphone. See NSCalendarsUsageDescription (page 56) for details.	iOS 7.0 and late
NSMotionUsage-Description	"Privacy - Motion Usage Description"	Specifies the reason for accessing the device's accelerometer. See NSMotionUsageDescription (page 60) for details.	iOS 7.0 and late
NSPersistentStore-TypeKey	"Core Data persistent store type"	The type of Core Data persistent store associated with a persistent document type. See NSPersistentStoreTypeKey (page 60) for details.	OS X
NSPhotoLibraryUsage-Description	"Privacy - Photo Library Usage Description"	Specifies the reason for accessing the user's photo library. See NSPhotoLibraryUsageDescription (page 60) for details.	iOS 6.0 and late
NSPrefPaneIconFile	"Preference Pane icon file"	The name of an image file resource used to represent a preference pane in the System Preferences app. See NSPrefPaneIconFile (page 60) for details.	OS X
NSPrefPaneIconLabel	"Preference Pane icon label"	The name of a preference pane displayed beneath the preference pane icon in the System Preferences app. See NSPrefPaneIconLabel (page 60) for details.	OS X
NSPrincipalClass	"Principal class"	The name of the bundle's main class. See NSPrincipalClass (page 61) for details.	OS X
NSRemindersUsage-Description	"Privacy - Reminders Usage Description"	Specifies the reason for accessing the user's reminders. See NSRemindersUsageDescription (page 61) for details.	iOS 6.0 and late
NSServices	"Services"	An array of dictionaries specifying the services provided by an app. See NSServices (page 61) for details.	OS X
NSSupportsAutomatic-Termination	None	Specifies whether the app may be killed to reclaim memory. See NSSupportsAutomaticTermination (page 70) for details.	OS X v10.7 an later
NSSupportsSudden-Termination	None	Specifies whether the app may be killed to allow for faster shut down or log out operations. See NSSupportsSuddenTermination (page 71) for details.	OS X

Key	Xcode name	Summary	Platform
NSUbiquitousContainer	None	Specifies the iCloud Drive settings for each container. See NSUbiquitousContainers (page 71) for details.	iOS, OS
NSUbiquitous-ContainerIsDocument-ScopePublic	None	Specifies whether the iCloud Drive should share the contents of this container. See NSUbiquitousContainerIsDocumentScopePublic (page 71) for details.	iOS, OS
NSUbiquitous-ContainerName	None	Specifies the name that the iCloud Drive displays for your container. See NSUbiquitousContainerName (page 71) for details.	iOS, OS
NSUbiquitous-ContainerSupported-FolderLevels	None	Specifies the maximum number of folder levels inside your container's Documents directory. See NSUbiquitousContainerSupportedFolderLevels (page 71) for details.	iOS, OS
NSUbiquitousDisplaySet	None	Specifies the mobile document data that the app can view. See NSUbiquitousDisplaySet (page 72) for details.	iOS, OS
NSUserActivityTypes	None	Specifies the user activity types that the app supports. See NSUserActivityTypes (page 72) for details.	iOS, OS
NSUserNotification-AlertStyle	None	Specifies whether the notification style should be `banner`, `alert`, or `none`. The default value is `banner`, which is the recommended style. See NSUserNotificationAlertStyle (page 72) for details.	OS X
UTExportedType-Declarations	"Exported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (and owned) by the app. See UTExportedTypeDeclarations (page 73) for details.	iOS 5.0 and late OS X v10.7 an later
UTImportedType-Declarations	"Imported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (but not owned) by the app. See UTImportedTypeDeclarations (page 74) for details.	iOS, OS

GKGameCenterBadgingDisabled

`GKGameCenterBadgingDisabled` (`Boolean` - iOS). This key determines if badges are added to your turn based app icon. Set the value of this key to `YES` to opt out of badging. Defaults to `NO`.

GKShowChallengeBanners

`GKShowChallengeBanners` (`Boolean` - iOS). This key determines if challenge banners are displayed within an app. Set the value of this key to `YES` to show challenge banners in the app. Set the value to `NO` to suppress challenge-related banners.

NSAppleScriptEnabled

`NSAppleScriptEnabled` (`Boolean` or `String` - OS X). This key identifies whether the app is scriptable. Set the value of this key to `true` (when typed as `Boolean`) or "YES" (when typed as `String`) if your app supports AppleScript.

NSBluetoothPeripheralUsageDescription

`NSBluetoothPeripheralUsageDescription` (`String` - iOS) describes the reason that the app uses Bluetooth. When the system prompts the user to allow usage, this string is displayed as part of the dialog box.

This key is supported in iOS 6.0 and later.

NSCalendarsUsageDescription

`NSCalendarsUsageDescription` (`String` - iOS) describes the reason that the app accesses the user's calendars. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 6.0 and later.

NSCameraUsageDescription

`NSCameraUsageDescription` (`String` - iOS) describes the reason that the app accesses the device's camera. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 7.0 and later.

NSContactsUsageDescription

`NSContactsUsageDescription` (`String` - iOS) describes the reason that the app accesses the user's contacts. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 6.0 and later.

NSDockTilePlugIn

`NSDockTilePlugIn` (`String` - OS X). This key contains the name of a plug-in bundle with the `.docktileplugin` filename extension and residing in the app's `Contents/PlugIns` directory. The bundle must contain the Dock tile plug-in for the app. For information about creating a Dock tile plug-in, see *Dock Tile Programming Guide* .

NSHumanReadableCopyright

`NSHumanReadableCopyright` (`String` - OS X). This key contains a string with the copyright notice for the bundle; for example, `© 2008, My Company`. You can load this string and display it in an About dialog box. This key can be localized by including it in your `InfoPlist.strings` files. This key replaces the obsolete `CFBundleGetInfoString` key.

NSJavaNeeded

`NSJavaNeeded` (`Boolean` or `String` - OS X). This key specifies whether the Java VM must be loaded and started up prior to executing the bundle code. This key is required only for Cocoa Java apps to tell the system to launch the Java environment. If you are writing a pure Java app, do not include this key.

You can also specify a string type with the value "YES" instead of a Boolean value if desired.

Deprecated in OS X v10.5.

NSJavaPath

`NSJavaPath` (`Array` - OS X). This key contains an array of paths. Each path points to a Java class. The path can be either an absolute path or a relative path from the location specified by the key NSJavaRoot (page 58). The development environment (or, specifically, its jamfiles) automatically maintains the values in the array.

Deprecated in OS X v10.5.

NSJavaRoot

`NSJavaRoot` (`String` - OS X). This key contains a string identifying a directory. This directory represents the root directory of the app's Java class files.

NSLocationAlwaysUsageDescription

`NSLocationAlwaysUsageDescription` (`String` - iOS) describes the reason why the app accesses the user's location information. Include this key when your app uses location services in a potentially nonobvious way while running in the foreground or the background. For example, a social app might include this key when it uses location information to track the user's location and display other users that are nearby. In this case, the fact that the app is tracking the user's location might not be readily apparent. The system includes the value of this key in the alert panel displayed to the user when requesting permission to use location services.

This key is required when you use the `requestAlwaysAuthorization` method of the `CLLocationManager` class to request authorization for location services. If this key is not present and you call the `requestAlwaysAuthorization` method, the system ignores your request and prevents your app from using location services.

This key is supported in iOS 8.0 and later. If your `Info.plist` file includes both this key and the `NSLocationUsageDescription` (page 58) key, the system uses this key and ignores the `NSLocationUsageDescription` key.

NSLocationUsageDescription

`NSLocationUsageDescription` (`String` - iOS) describes the reason why the app accesses the user's location information. When the system prompts the user to allow access, this string is displayed as part of the alert panel.

This key is supported in iOS 6.0 and later and is ignored in iOS 8 and later. In iOS 8, you must use the NSLocationAlwaysUsageDescription (page 58) or NSLocationWhenInUseUsageDescription (page 59) key instead.

NSLocationWhenInUseUsageDescription

`NSLocationWhenInUseUsageDescription` (`String` - iOS) describes the reason why the app accesses the user's location normally while running in the foreground. Include this key when your app uses location services to track the user's current location directly. This key does not support using location services to monitor regions or monitor the user's location using the significant location change service. The system includes the value of this key in the alert panel displayed to the user when requesting permission to use location services.

This key is required when you use the `requestWhenInUseAuthorization` method of the `CLLocationManager` class to request authorization for location services. If the key is not present when you call the `requestWhenInUseAuthorization` method without including this key, the system ignores your request.

This key is supported in iOS 8.0 and later. If your `Info.plist` file includes both this key and the `NSLocationUsageDescription` (page 58) key, the system uses this key and ignores the `NSLocationUsageDescription` key.

NSMainNibFile

`NSMainNibFile` (`String` - iOS, OS X). This key contains a string with the name of the app's main nib file (minus the `.nib` extension). A nib file is an Interface Builder archive containing the description of a user interface along with any connections between the objects of that interface. The main nib file is automatically loaded when an app is launched.

This key is mutually exclusive with the UIMainStoryboardFile (page 89) key. You should include one of the keys in your `Info.plist` file but not both.

NSMicrophoneUsageDescription

`NSMicrophoneUsageDescription` (`String` - iOS) describes the reason that the app accesses the device's microphone. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 7.0 and later.

NSMotionUsageDescription

`NSMotionUsageDescription` (`String` - iOS) describes the reason that the app accesses the device's accelerometer. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 7.0 and later.

NSPersistentStoreTypeKey

`NSPersistentStoreTypeKey` (`String` - OS X). This key contains a string that specifies the type of Core Data persistent store associated with a document type (see CFBundleDocumentTypes (page 24)).

NSPhotoLibraryUsageDescription

`NSPhotoLibraryUsageDescription` (`String` - iOS) describes the reason that the app accesses the user's photo library. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 6.0 and later.

NSPrefPaneIconFile

`NSPrefPaneIconFile` (`String` - OS X). This key contains a string with the name of an image file (including extension) containing the preference pane's icon. This key should only be used by preference pane bundles. The image file should contain an icon 32 by 32 pixels in size. If this key is omitted, the System Preferences app looks for the image file using the `CFBundleIconFile` key instead.

NSPrefPaneIconLabel

`NSPrefPaneIconLabel` (`String` - OS X). This key contains a string with the name of a preference pane. This string is displayed below the preference pane's icon in the System Preferences app. You can split long names onto two lines by including a newline character ('\n') in the string. If this key is omitted, the System Preferences app gets the name from the `CFBundleName` key.

This key can be localized and included in the `InfoPlist.strings` files of a bundle.

NSPrincipalClass

`NSPrincipalClass` (`String` - OS X). This key contains a string with the name of a bundle's principal class. This key is used to identify the entry point for dynamically loaded code, such as plug-ins and other dynamically-loaded bundles. The principal class of a bundle typically controls all other classes in the bundle and mediates between those classes and any classes outside the bundle. The class identified by this value can be retrieved using the `principalClass` method of `NSBundle`. For Cocoa apps, the value for this key is `NSApplication` by default.

NSRemindersUsageDescription

`NSRemindersUsageDescription` (`String` - iOS) describes the reason that the app accesses the user's reminders. When the system prompts the user to allow access, this string is displayed as part of the dialog box.

This key is supported in iOS 6.0 and later.

NSServices

`NSServices` (`Array` - OS X). This key contains an array of dictionaries specifying the services provided by the app. Table 2 lists the keys for specifying a service:

Table 2 Keys for NSServices dictionaries

Key	Xcode name	Type	Description	Platforms
NSPortName	"Incoming service port name"	`String`	This key specifies the name of the port your app monitors for incoming service requests. Its value depends on how the service provider app is registered. In most cases, this is the app name. For more information, see *Services Implementation Guide*.	OS X

Key	Xcode name	Type	Description	Platforms
NSMessage	"Instance method name"	`String`	This key specifies the name of the instance method to invoke for the service. In Objective-C, the instance method must be of the form `messageName:userData:error:`. In Java, the instance method must be of the form `messageName(NSPaste-Board,String)`.	OS X
NSSendFileTypes	None	`Array`	This key specifies an array of strings. Each string should contain a UTI defining a supported file type. Only UTI types are allowed; pasteboard types are not permitted. To specify pasteboard types, continue to use the `NSSendTypes` key. By assigning a value to this key, your service declares that it can operate on files whose type conforms to one or more of the given file types. Your service will receive a pasteboard from which you can read file URLs. Available in OS X v10.6 and later. For information on UTIs, see *Uniform Type Identifiers Overview*.	OS X

Key	Xcode name	Type	Description	Platforms
NSSendTypes	“Send Types”	`Array`	This key specifies an optional array of data type names that can be read by the service. The `NSPasteboard` class description lists several common data types. You must include this key, the `NSReturnTypes` key, or both. In OS X v10.5 and earlier, this key is required. In OS X v10.6 and later, you should use the `NSSendFileTypes` key instead.	OS X
NSServiceDescription	None	`String`	This key specifies a description of your service that is suitable for presentation to users. This description string may be long to give users adequate information about your service. To localize the menu item text, create a `ServicesMenu.strings` file for each localization in your bundle. This strings file should contain this key along with the translated description string as its value. For more information about creating strings files, see *Resource Programming Guide* . Available in OS X v10.6 and later.	OS X

Key	Xcode name	Type	Description	Platforms
NSRequiredContext	None	`Dictionary` or `Array`	This key specifies a dictionary with the conditions under which your service is made available to the user. Alternatively, you can specify an array of dictionaries, each of which contains a set of conditions for enabling your service. See the discussion after this table for information about specifying the value of this key. Available in OS X v10.6 and later.	OS X
NSRestricted	None	`Boolean`	Specifying a value of `true` for this key prevents the service from being invoked by a sandboxed app. You should set the value to `true` if your service performs privileged or potentially dangerous operations that would allow a sandboxed app to escape its containment. For example, you should set it to `true` if your service executes arbitrary files or text strings as scripts, reads or writes any file specified by a path, or retrieves the contents of an arbitrary URL from the network on behalf of the client of the service. The default value for this key is `false`. Available in OS X v10.7 and later.	OS X

Key	Xcode name	Type	Description	Platforms
NSReturnTypes	"Return Types"	`Array`	This key specifies an array of data type names that can be returned by the service. The `NSPasteboard` class description lists several common data types. You must include this key, the `NSSendTypes` key, or both.	OS X

Key	Xcode name	Type	Description	Platforms
NSMenuItem	“Menu”	`Dictionary`		OS X

Key	Xcode name	Type	Description	Platforms
			This key contains a dictionary that specifies the text to add to the Services menu. The only key in the dictionary is called `default` and its value is the menu item text. In OS X v10.5 and earlier, menu items must be unique. You can ensure a unique name by combining the app name with the command name and separating them with a slash character "`/`". This effectively creates a submenu for your services. For example, `Mail/Send` would appear in the Services menu as a menu named Mail with an item named Send. Submenus are not supported (or necessary) in OS X v10.6 and later. If you specify a slash character in OS X v10.6 and later, the slash and any text preceding it are discarded. Instead, services with the same name are disambiguated by adding the app name in parenthesis after the menu item text. To localize the menu item text, create a `ServicesMenu.strings` file for each localization in your bundle. This strings file should contain the `default` key along with	

Key	Xcode name	Type	Description	Platforms
			the translated menu item text as its value. For more information about creating strings files, see *Resource Programming Guide* .	
NSKeyEquivalent	“Menu key equivalent”	`Dictionary`	This key is optional and contains a dictionary with the keyboard equivalent used to invoke the service menu command. Similar to `NSMenuItem`, the only key in the dictionary is called `default` and its value is a single character. Users invoke this keyboard equivalent by pressing the Command modifier key along with the character. The character is case sensitive, so you can assign different commands to the uppercase and lowercase versions of a character. To specify the uppercase character, the user must press the Shift key in addition to the other keys.	OS X
NSUserData	“User Data”	`String`	This key is an optional string that contains a value of your choice.	OS X
NSTimeout	“Timeout value (in milliseconds)”	`String`	This key is an optional numerical string that indicates the number of milliseconds Services should wait for a response from the app providing a service when a response is required.	OS X

In OS X v10.6 and later, the `NSRequiredContext` key may contain a dictionary or an array of dictionaries describing the conditions under which the service appears in the Services menu. If you specify a single dictionary, all of the conditions in that dictionary must be met for the service to appear. If you specify an array of dictionaries, all of the conditions in only one of those dictionaries must be met for the service to appear. Each dictionary may contain one or more of the keys listed in Table 3. All keys in the dictionary are optional.

Table 3 Contents of the `NSRequiredContext` dictionary

Key	Xcode name	Type	Description	Platform
NSApplicationIdentifier	None	`String` or `Array`	The value of this key is a string or an array of strings, each of which contains the bundle ID (`CFBundleIdentifier` key) of an app. Your service appears only if the bundle ID of the current app matches one of the specified values.	OS X
NSTextScript	None	`String` or `Array`	The value of this key is a string or an array of strings, each of which contains a standard four-letter script tag, such as `Latn` or `Cyrl`. Your service appears only if the dominant script of the selected text matches one of the specified script values.	OS X
NSTextLanguage	None	`String` or `Array`	The value of this key is a string or an array of strings, each of which contains a BCP-47 tag indicating the language of the desired text. Your service appears if the overall language of the selected text matches one of the specified values. Matching is performed using a prefix-matching scheme. For example, specifying the value en matches text whose full BCP-47 code is `en-US`, `en-GB`, or `en-AU`.	OS X

Key	Xcode name	Type	Description	Platform
NSWordLimit	None	`Number`	The value of this key is an integer indicating the maximum number of selected words on which the service can operate. For example, a service to look up a stock by ticker symbol might have a value of 1 because ticker symbols cannot contain spaces.	OS X
NSTextContext	None	`String` or `Array`	The value of this key is a string or an array of strings, each of which contains one of the following values: `URL`, `Date`, `Address`, `Email`, or `FilePath`. The service is displayed only if the selected text contains data of a corresponding type. For example, if the selected text contained an `http`-based link, the service would be displayed if the value of this key were set to `URL`. Note that all of the selected text is provided to the service-vending app, not just the parts found to contain the given data types.	OS X

For additional information about implementing services in your app, see *Services Implementation Guide* .

NSSupportsAutomaticTermination

`NSSupportsAutomaticTermination` (`Boolean` - OS X). This key contains a boolean that indicates whether the app supports automatic termination in OS X v10.7 and later. Automatic termination allows an app that is running to be terminated automatically by the system when certain conditions apply. Primarily, the app can be terminated when it is hidden or does not have any visible windows and is not currently being used. The system may terminate such an app in order to reclaim the memory used by the app.

An app may programmatically disable and reenable automatic termination support using the `disableAutomaticTermination` and `enableAutomaticTermination` methods of `NSProcessInfo`. The app might do this to prevent being terminated during a critical operation.

NSSupportsSuddenTermination

`NSSupportsSuddenTermination` (`Boolean` - OS X). This key contains a boolean that indicates whether the system may kill the app outright in order to log out or shut down more quickly. You use this key to specify whether the app can be killed immediately after launch. The app can still enable or disable sudden termination at runtime using the methods of the `NSProcessInfo` class. The default value of this key is `NO`.

NSUbiquitousContainers

`NSUbiquitousContainers` (`Dictionary` - iOS and OS X) Specifies the iCloud Drive settings for each container. This dictionary's keys are the container identifiers for your app's iCloud containers. The values are dictionaries containing the `NSUbiquitousContainerIsDocumentScopePublic`, `NSUbiquitousContainerName` and `NSUbiquitousContainerSupportedFolderLevels` entries for each container. You must specify the sharing permissions separately for each container.

NSUbiquitousContainerIsDocumentScopePublic

`NSUbiquitousContainerIsDocumentScopePublic` (`Boolean` - iOS and OS X) Specifies whether the iCloud drive should share the contents of this container. Defaults to `NO`.

NSUbiquitousContainerName

`NSUbiquitousContainerName` (`String` - iOS and OS X) Specifies the name that the iCloud Drive displays for your container. By default, the iCloud Drive will use the name of the bundle that owns the container.

NSUbiquitousContainerSupportedFolderLevels

`NSUbiquitousContainerSupportedFolderLevels` (`String` - iOS and OS X) Specifies the maximum number of folder levels inside your container's Documents directory. This key can take three different values:

- `None`

The iCloud Drive only has access to the container's Documents directory. Your app promises that it does not create any directories inside the Document's directory. On OS X, the Finder prevents users from creating subdirectories inside your iCloud Drive directory.

- `One`

  The iCloud Drive has access to the container's Documents directory and one additional layer of subdirectories. Your app promises that it only creates a single layer of directories inside the Documents directory. On OS X, the Finder prevents users from creating more than one layer of subdirectories inside your iCloud Drive directory.

- `Any`

  The iCloud Drive has complete access to your container's Documents directory. Both your app and the Finder can create as many layers of subdirectories as you (or the user) desire.

NSUbiquitousDisplaySet

`NSUbiquitousDisplaySet` (`String` - iOS, OS X) contains the identifier string that you configured in iTunesConnect for managing your app's storage. The assigned display set determines from which mobile data folder (in the user's mobile account) the app retrieves its data files.

If you create multiple apps, you can use the same display set for your apps or assign different display sets to each. For example, if you create a "lite" version of your app, in addition to a full-featured version, you might use the same display set for both versions because they create and use the same basic data files. Each app should recognize the file types stored in its mobile data folder and be able to open them.

NSUserActivityTypes

`NSUserActivityTypes` (Array of strings - iOS and OS X) Specifies the user activity types that the app supports. This key is valid in iOS 8 and OS X v10.10 and later.

NSUserNotificationAlertStyle

`NSUserNotificationAlertStyle` (`String` - OS X) specifies the notification style the app should use. The default value, `banner`, is recommended; most apps should not need to use the `alert` style.

UTExportedTypeDeclarations

`UTExportedTypeDeclarations` (`Array` - iOS, OS X) declares the uniform type identifiers (UTIs) owned and exported by the app. You use this key to declare your app's custom data formats and associate them with UTIs. Exporting a list of UTIs is the preferred way to register your custom file types; however, Launch Services recognizes this key and its contents only in OS X v10.5 and later. This key is ignored on versions of OS X prior to version 10.5.

The value for the `UTExportedTypeDeclarations` key is an array of dictionaries. Each dictionary contains a set of key-value pairs identifying the attributes of the type declaration. Table 4 lists the keys you can include in this dictionary along with the typical values they contain. These keys can also be included in array of dictionaries associated with the UTImportedTypeDeclarations (page 74) key.

Table 4 UTI property list keys

Key	Xcode name	Type	Description	P
UTTypeConformsTo	"Conforms to UTIs"	`Array`	(Required) Contains an array of strings. Each string identifies a UTI to which this type conforms. These keys represent the parent categories to which your custom file format belongs. For example, a JPEG file type conforms to the `public.image` and `public.data` types. For a list of high-level types, see *Uniform Type Identifiers Overview*.	i
UTTypeDescription	"Description"	`String`	A user-readable description of this type. The string associated with this key may be localized in your bundle's `InfoPlist.strings` files.	i
UTTypeIconFile	"Icon file name"	`String`	The name of the bundle icon resource to associate with this UTI. You should include this key only for types that your app exports. This file should have a `.icns` filename extension. You can create this file using the Icon Composer app that comes with Xcode Tools.	C
UTTypeIdentifier	"Identifier"	`String`	(Required) The UTI you want to assign to the type. This string uses the reverse-DNS format, whereby more generic types come first. For example, a custom format for your company would have the form `com.<yourcompany>.<type>.<subtype>`.	i
UTTypeReferenceURL	"Reference URL"	`String`	The URL for a reference document that describes this type.	C

Key	Xcode name	Type	Description	P
UTTypeSize64IconFile	None	`String`	The name of the 64 x 64 pixel icon resource file (located in the app's bundle) to associate with this UTI. You should include this key only for types that your app exports.	i
UTTypeSize320IconFile	None	`String`	The name of the 320 x 320 pixel icon resource file (located in the app's bundle) to associate with this UTI. You should include this key only for types that your app exports.	i
UTTypeTagSpecification	"Equivalent Types"	`Dictionary`	(Required) A dictionary defining one or more equivalent type identifiers. The key-value pairs listed in this dictionary identify the filename extensions, MIME types, OSType codes, and pasteboard types that correspond to this type. For example, to specify filename extensions, you would use the key `public.filename-extension` and associate it with an array of strings containing the actual extensions. For more information about the keys for this dictionary, see *Uniform Type Identifiers Overview*.	i

The way you specify icon files in OS X and iOS is different because of the supported file formats on each platform. In iOS, each icon resource file is typically a PNG file that contains only one image. Therefore, it is necessary to specify different image files for different icon sizes. However, when specifying icons in OS X, you use an icon file (with extension `.icns`), which is capable of storing the icon at several different resolutions.

This key is supported in iOS 3.2 and later and OS X v10.5 and later. For more information about UTIs and their use, see *Uniform Type Identifiers Overview*.

UTImportedTypeDeclarations

`UTImportedTypeDeclarations` (`Array` - iOS, OS X) declares the uniform type identifiers (UTIs) inherently supported (but not owned) by the app. You use this key to declare any supported types that your app recognizes and wants to ensure are recognized by Launch Services, regardless of whether the app that owns them is present. For example, you could use this key to specify a file format that is defined by another company but which your program can read and export.

The value for this key is an array of dictionaries and uses the same keys as those for the UTExportedTypeDeclarations (page 73) key. For a list of these keys, see Table 4 (page 73).

This key is supported in iOS 3.2 and later and OS X v10.5 and later. For more information about UTIs and their use, see *Uniform Type Identifiers Overview* .

OS X Keys

The keys in this chapter define assorted functionality related to OS X bundles.

Key Summary

Table 1 contains an alphabetical listing of OS X–specific keys, the corresponding name for that key in the Xcode property list editor, and a high-level description of each key. Detailed information about each key is available in later sections.

Table 1 Summary of OS X keys

Key	Xcode name	Summary
APInstallerURL	"Installation directory base file URL"	A URL-based path to the files you want to install. See APInstallerURL (page 77) for details.
APFiles	"Installation files"	An array of dictionaries describing the files or directories that can be installed. See APFiles (page 77) for details.
ATSApplicationFontsPath	"Application fonts resource path"	The path to a single font file or directory of font files in the bundle's `Resources` directory. See ATSApplicationFontsPath (page 77) for details.
CSResourcesFileMapped	"Resources should be file-mapped"	If true, Core Services routines map the bundle's resource files into memory instead of reading them. See CSResourcesFileMapped (page 78) for details.
NSMainStoryboardFile	"Main storyboard file base name"	Specifies the name of the app's storyboard resource file. See NSMainStoryboardFile (page 78) for details.
QLSandboxUnsupported	None	Specifies that a Quick Look plug-in does not support sandboxing. See for details.
QuartzGLEnable	None	Specifies whether the app uses Quartz GL. See QuartzGLEnable (page 78) for details.

APInstallerURL

`APInstallerURL` (`String` - OS X) identifies the base path to the files you want to install. You must specify this path using the form `file://localhost/path/`. All installed files must reside within this directory.

APFiles

`APFiles` (`Array` - OS X) specifies a file or directory you want to install. You specify this key as a dictionary, the contents of which contains information about the file or directory you want to install. To specify multiple items, nest the `APFiles` key inside itself to specify files inside of a directory. Table 2 lists the keys for specifying information about a single file or directory.

Table 2 Keys for APFiles dictionary

Key	Xcode name	Type	Description
APFileDescriptionKey	"Install file description text"	`String`	A short description of the item to display in the Finder's Info window
APDisplayedAsContainer	"Display with folder icon"	`String`	If "Yes" the item is shown with a folder icon in the Info panel; otherwise, it is shown with a document icon
APFileDestinationPath	"File destination path"	`String`	Where to install the component as a path relative to the app bundle
APFileName	"Install file name"	`String`	The name of the file or directory
APFileSourcePath	"Install file source path"	`String`	The path to the component in the app package relative to the `APInstallerURL` path.
APInstallAction	"File install action"	`String`	The action to take with the component: "Copy" or "Open"

ATSApplicationFontsPath

`ATSApplicationFontsPath` (`String` - OS X) identifies the location of a font file or directory of fonts in the bundle's `Resources` directory. If present, OS X activates the fonts at the specified path for use by the bundled app. The fonts are activated only for the bundled app and not for the system as a whole. The path itself should be specified as a relative directory of the bundle's Resources directory. For example, if a directory of fonts was at the path `/Applications/MyApp.app/Contents/Resources/Stuff/MyFonts/`, you should specify the string `Stuff/MyFonts/` for the value of this key.

CSResourcesFileMapped

`CSResourcesFileMapped` (`Boolean` - OS X) specifies whether to map this app's resource files into memory. Otherwise, they are read into memory normally. File mapping can improve performance in situations where you are frequently accessing a small number of resources. However, resources are mapped into memory read-only and cannot be modified.

NSMainStoryboardFile

`NSMainStoryboardFile` (`String` - OS X) contains a string with the name of the app's main storyboard file (minus the `.storyboard` filename extension). A storyboard file is an Interface Builder archive containing the app's view controllers, the connections between those view controllers and their immediate views, and the segues between view controllers. When this key is present, the main storyboard file is loaded automatically at launch time and its initial view controller is installed in the app's window.

This key is mutually exclusive with the NSMainNibFile (page 59) key. You should include one of these keys in your `Info.plist` file but not both. This key is supported in OS X v10.10 and later.

QLSandboxUnsupported

`QLSandboxUnsupported` (`Boolean` - OS X) allows a Quick Look plug-in to opt out of sandboxing. A code-signed Quick Look plug-in—which includes all plug-ins that are bundled in apps available in the Mac App Store—is sandboxed by default. Use this key to temporarily disable sandboxing while you update your plug-in to be compatible with it.

QuartzGLEnable

`QuartzGLEnable` (`Boolean` - OS X) specifies whether this app uses Quartz GL.

Value	Description
`true`	Turn on Quartz GL for the app's windows. (This works only when the computer has at least 1 GB of RAM).
`false`	Disable Quartz GL. Quartz GL will not be available, even after using `[<NSWindow> setPreferredBackingLocation:]`.

Quartz GL is not supported on computers with more than one video card installed.

To turn on Quartz QL for testing use the Quartz Debug app, located in `<Xcode>/Applications`.

This key is available in OS X v10.5 and later.

iOS Keys

SwiftObjective-C
The iOS frameworks provide the infrastructure you need for creating iOS apps. You use the keys associated with this framework to configure the appearance of your app at launch time and the behavior of your app once it is running.

UIKit keys use the prefix `UI` to distinguish them from other keys. Other frameworks also use appropriate prefixes. For more information about configuring the information property-list file of your iOS app, see *App Programming Guide for iOS* .

Key Summary

Table 1 contains an alphabetical listing of iOS keys, the corresponding name for that key in the Xcode property list editor, a high-level description of each key, and the platforms on which you use it. Detailed information about each key is available in later sections.

Table 1 Summary of UIKit keys

Key	Xcode name	Summary	Availability
MKDirections-ApplicationSupported-Modes	None	Specifies the types of directions the app can deliver. See MKDirectionsApplicationSupportedModes (page 82) for details.	iOS 6.0 and later
NSHealthShareUsage-Description	"Privacy - Health Share Usage Description"	Specifies a localized string that describes why the app wants to read HealthKit data. See NSHealthShareUsageDescription (page 83) for details.	iOS 8.0 and later
NSHealthUpdateUsage-Description	"Privacy - Health Update Usage Description"	Specifies a localized string that describes why the app wants to write data to the HealthKit store. See NSHealthUpdateUsageDescription (page 83) for details.	iOS 8.0 and later
UIAppFonts	"Fonts provided by application"	Specifies a list of app-specific fonts. See UIAppFonts (page 84) for details.	iOS 3.2 and later

Key	Xcode name	Summary	Availability
UIApplicationExits-OnSuspend	"Application does not run in background"	Specifies whether the app terminates instead of run in the background. See UIApplicationExitsOnSuspend (page 84) for details.	iOS 4.0 and later
UIBackgroundModes	"Required background modes"	Specifies that the app needs to continue running in the background. See UIBackgroundModes (page 84) for details.	iOS 4.0 and later
UIDeviceFamily	"Targeted device family"	Inserted automatically by Xcode to define the target device of the app. See UIDeviceFamily (page 86) for details.	iOS 3.2 and later
UIFileSharingEnabled	"Application supports iTunes file sharing"	Specifies whether the app shares files with the user's computer through iTunes. See UIFileSharingEnabled (page 86) for details.	iOS 3.2 and later
UIInterfaceOrientation	"Initial interface orientation"	Specifies the initial orientation of the app's user interface. See UIInterfaceOrientation (page 86) for details.	iOS
UILaunchImageFile	"Launch image"	Specifies the name of the app's launch image. See UILaunchImageFile (page 87) for details.	iOS 3.2 and later
UILaunchImages	None	Specifies the launch images to use for the app. See UILaunchImages (page 88) for details.	iOS 7.0 and later
UIMainStoryboardFile	"Main storyboard file base name"	Specifies the name of the app's storyboard resource file. See UIMainStoryboardFile (page 89) for details.	iOS 5.0 and later
UINewsstandApp	None	Specifies whether the app presents its content in Newsstand. See UINewsstandApp (page 89) for details.	iOS 5.0 and later
UIPrerenderedIcon	"Icon already includes gloss effects"	Specifies whether the app's icon already includes a shine effect. See UIPrerenderedIcon (page 90) for details.	iOS
UIRequiredDevice-Capabilities	"Required device capabilities"	Specifies the device-related features required for the app to run. See UIRequiredDeviceCapabilities (page 90) for details.	iOS 3.0 and later

Key	Xcode name	Summary	Availability
UIRequiresPersistent-WiFi	"Application uses Wi-Fi"	Specifies whether this app requires a Wi-Fi connection. See UIRequiresPersistentWiFi (page 93) for details.	iOS
UIStatusBarHidden	"Status bar is initially hidden"	Specifies whether the status bar is initially hidden when the app launches. See UIStatusBarHidden (page 94) for details.	iOS
UIStatusBarStyle	"Status bar style"	Specifies the style of the status bar as the app launches. See UIStatusBarStyle (page 94) for details.	iOS
UISupportedExternal-AccessoryProtocols	"Supported external accessory protocols"	Specifies the communications protocols supported for communication with attached hardware accessories. See UISupportedExternalAccessoryProtocols (page 94) for details.	iOS 3.0 and later
UISupportedInterface-Orientations	"Supported interface orientations"	Specifies the orientations that the app supports. See UISupportedInterfaceOrientations (page 95) for details.	iOS 3.2 and later
UIViewController-BasedStatusBar-Appearance	None	Specifies whether the view controller determines the status bar style. See UIViewControllerBasedStatusBarAppearance (page 95) for details.	iOS 7.0 and later
UIViewEdgeAntialiasing	"Renders with edge antialiasing"	Specifies whether Core Animation layers use antialiasing when drawing does not align to pixel boundaries. See UIViewEdgeAntialiasing (page 96) for details.	iOS 3.0 and later
UIViewGroupOpacity	"Renders with group opacity"	Specifies whether Core Animation layers inherit the opacity of their superlayer. See UIViewGroupOpacity (page 96) for details.	iOS 3.0 and later

MKDirectionsApplicationSupportedModes

`MKDirectionsApplicationSupportedModes` (`Array` - iOS) specifies the modes of transportation for which the app is capable of giving directions. The array contains one or more strings, each of which indicates a supported mode. Apps may include the following strings in this array:

- `MKDirectionsModeCar`

- `MKDirectionsModeBus`
- `MKDirectionsModeTrain`
- `MKDirectionsModeSubway`
- `MKDirectionsModeStreetCar`
- `MKDirectionsModePlane`
- `MKDirectionsModeBike`
- `MKDirectionsModeFerry`
- `MKDirectionsModeTaxi`
- `MKDirectionsModePedestrian`
- `MKDirectionsModeOther`

If this key is in your app's `Info.plist` file, you must upload a GeoJSON file to iTunes Connect that indicates the regions for which your app is capable of providing directions. For more information about specifying this file, see *Location and Maps Programming Guide* .

This key is supported in iOS 6.0 and later.

NSHealthShareUsageDescription

`NSHealthShareUsageDescription` (`String` - iOS) specifies a localized string that describes why the app wants to read HealthKit data. The system displays this description on the permissions sheet when you call the HealthKit store's `requestAuthorizationToShareTypes:readTypes:completion:` method.

This key is supported in iOS 8.0 and later.

NSHealthUpdateUsageDescription

`NSHealthUpdateUsageDescription` (`String` - iOS) specifies a localized string that describes why the app wants to write data to the HealthKit store. The system displays this description on the permissions sheet when you call the HealthKit store's `requestAuthorizationToShareTypes:readTypes:completion:` method.

This key is supported in iOS 8.0 and later.

UIAppFonts

`UIAppFonts` (`Array` - iOS) specifies any app-provided fonts that should be made available through the normal mechanisms. Each item in the array is a string containing the name of a font file (including filename extension) that is located in the app's bundle. The system loads the specified fonts and makes them available for use by the app when that app is run.

This key is supported in iOS 3.2 and later.

UIApplicationExitsOnSuspend

`UIApplicationExitsOnSuspend` (`Boolean` - iOS) specifies that the app should be terminated rather than moved to the background when it is quit. Apps linked against iOS SDK 4.0 or later can include this key and set its value to `YES` to prevent being automatically opted-in to background execution and app suspension. When the value of this key is `YES`, the app is terminated and purged from memory instead of moved to the background. If this key is not present, or is set to `NO`, the app moves to the background as usual.

This key is supported in iOS 4.0 and later.

UIBackgroundModes

`UIBackgroundModes` (`Array` - iOS) specifies that the app provides specific background services and must be allowed to continue running while in the background. These keys should be used sparingly and only by apps providing the indicated services. Where alternatives for running in the background exist, those alternatives should be used instead. For example, apps can use the signifiant location change interface to receive location events instead of registering as a background location app.

Table 2 lists the possible string values that you can put into the array associated with this key. You can include any or all of these strings but your app must provide the indicated services.

Table 2 Values for the `UIBackgroundModes` array

Value	Description
`audio`	The app plays audible content in the background.
`location`	The app provides location-based information to the user and requires the use of the standard location services (as opposed to the significant change location service) to implement this feature.

Value	Description
`voip`	The app provides Voice-over-IP services. Apps with this key are automatically launched after system boot so that the app can reestablish VoIP services. Apps with this key are also allowed to play background audio.
`fetch`	The app requires new content from the network on a regular basis. When it is convenient to do so, the system launches or resumes the app in the background and gives it a small amount of time to download any new content. This value is supported in iOS 7.0 and later.
`remote-notification`	The app uses remote notifications as a signal that there is new content available for download. When a remote notification arrives, the system launches or resumes the app in the background and gives it a small amount of time to download the new content. This value is supported in iOS 7.0 and later.
`newsstand-content`	The app processes content that was recently downloaded in the background using the Newsstand Kit framework, so that the content is ready when the user wants it. This value is supported in iOS 5.0 and later.
`external-accessory`	The app communicates with an accessory that delivers data at regular intervals. This value is supported in iOS 5.0 and later.
`bluetooth-central`	The app uses the CoreBluetooth framework to communicate with a Bluetooth accessory while in the background. This value is supported in iOS 5.0 and later.
`bluetooth-peripheral`	The app uses the CoreBluetooth framework to communicate in peripheral mode with a Bluetooth accessory. The system will alert the user to the potential privacy implications of apps with this key set. See Best Practices for Maintaining User Privacy for more information on privacy. This value is supported in iOS 6.0 and later.

This key is supported in iOS 4.0 and later.

UIDeviceFamily

`UIDeviceFamily` (`Number` or `Array` - iOS) specifies the underlying hardware type on which this app is designed to run.

Important: Do not insert this key manually into your `Info.plist` files. Xcode inserts it automatically based on the value in the Targeted Device Family build setting. You should use that build setting to change the value of the key.

The value of this key is usually an integer but it can also be an array of integers. Table 3 lists the possible integer values you can use and the corresponding devices.

Table 3 Values for the `UIDeviceFamily` key

Value	Description
1	(Default) The app runs on iPhone and iPod touch devices.
2	The app runs on iPad devices.

This key is supported in iOS 3.2 and later.

UIFileSharingEnabled

`UIFileSharingEnabled` (`Boolean` - iOS) specifies whether the app shares files through iTunes. If this key is `YES`, the app shares files. If it is not present or is `NO`, the app does not share files. Apps must put any files they want to share with the user in their <*Application_Home*>`/Documents` directory, where <*Application_Home*> is the path to the app's home directory.

In iTunes, the user can access an app's shared files from the File Sharing section of the Apps tab for the selected device. From this tab, users can add and remove files from the directory.

This key is supported in iOS 3.2 and later.

UIInterfaceOrientation

`UIInterfaceOrientation` (`String` - iOS) specifies the initial orientation of the app's user interface. This key is ignored if the UISupportedInterfaceOrientations (page 95) key is present.

This value is based on the `UIInterfaceOrientation` constants declared in the `UIApplication.h` header file. The default style is `UIInterfaceOrientationPortrait`.

UILaunchImageFile

`UILaunchImageFile` (`String` - iOS) specifies the name of the launch image file for the app on older versions of iOS. If this key and the UILaunchImages (page 88) key are both present, the app uses the UILaunchImages key. If neither key is present, the system looks for a launch image file with the name `Default.png`.

This key is supported in iOS 3.2 and later.

Naming Your Launch Image Files

The name of each launch image conveys its purpose and how it is used. It is recommended that you use the following format for launch image filenames:

<basename> *<usage_specific_modifiers>* `.png`

The *<basename>* portion of the filename is the string that you specify using the `UILaunchImageFile` key. (If you do not specify the key, iOS uses the string `Default` for the launch image name.) To the base name, you can add several different types of modifiers:

- **High-resolution image modifier**—Use the `@2x` modifier to identify images that are intended for Retina displays.
- **Platform-specific modifiers**—Use the modifiers `~iphone` or `~ipad` to specify images targeting a specific size of device.
- **Orientation-specific modifiers**—Use the strings `-Landscape`, `-Portrait`, `-LandscapeLeft`, `-LandscapeRight`, or `-PortraitUpsideDown` to specify launch images when the device is in a specific orientation. More specific orientation modifiers take precedence over less-specific orientation modifiers. For example, an image with the `-LandscapeLeft` modifier takes precedence over an image with the `-Landscape` modifier when the device is in the correct orientation.
- **iPhone 5 modifier**—Use the string `-568h` to specify a launch image intended for devices whose screen is 568 points high. Because such devices also have Retina displays, you should also include the `@2x` modifier in the image name.
- **URL scheme modifiers**—Include the name of your app's custom URL scheme in launch image names if you want those launch images displayed when your app is launched to open a URL of the specified type. The format of a scheme modifier is -*<url_scheme>* where *<url_scheme>* is your custom scheme name. For example, if your app opens URLs of the form `myscheme://example.com`, you would include `-myscheme` in your launch image names.

UILaunchImages

`UILaunchImages` (`Array` - iOS) explicitly specifies the launch images to use for the app. This key contains an array of dictionaries. Each dictionary contains detailed information about a single launch image and how it is used. Xcode fills in the value of each dictionary based on information you provide in your project settings.

Table 4 lists the keys that can be included in each dictionary of the array.

Table 4 Keys for launch image dictionaries

Key	Description
`UILaunchImageName` (required)	A string containing the name of the PNG image file. The image file must reside at the top level of the app bundle. The name you specify for this key should not include a filename extension, nor should it include modifiers such as `@2x`, `-568h`, `~iphone`, or `~ipad`. On disk, your image filenames may still include the `@2x`, `-568h`, `~iphone`, or `~ipad` modifiers as appropriate, although they are not required. The system automatically considers such modifiers when choosing which file to load.
`UILaunchImageMinimum-OSVersion` (required)	A string representing the minimum iOS version number for which the image is intended. The form of the version number is the same as for the LSMinimumSystemVersion (page 48) key, which is a string of the form *n*.*n*.*n* where *n* is a number. For example, images targeting iOS 7 and later should specify a string "7.0".
`UILaunchImageSize`	A string containing the width and height of the image. This string represents the size of the display for which the image is intended. You must specify the width and height with respect to the device in a portrait orientation. In other words, portrait and landscape images targeting the same device would have the same width and height. The format of this string is `{width, height}` where width and height are the size of the image in points. (The format of this string is the same as that generated by the `NSStringFromCGSize` function.) For example, the string `{320, 480}` specifies an image that can be used on an iPhone 4 or iPhone 4S. If you do not specify this key, the image size is assumed to be `{320, 480}`.

Key	Description
`UILaunchImage-Orientation`	A string containing the orientation of the image. The value of this key is one of the following values: • `Portrait` • `PortraitUpsideDown` • `Landscape` • `LandscapeLeft` • `LandscapeRight` If you do not specify a value for this key, the orientation is assumed to be `Portrait`.

If this key is present, iOS 7 uses it exclusively to obtain launch images. The system does not fall back to the older naming conventions used prior to iOS 7.

This key is supported in iOS 7.0 and later.

UIMainStoryboardFile

`UIMainStoryboardFile` (`String` - iOS) contains a string with the name of the app's main storyboard file (minus the `.storyboard` extension). A storyboard file is an Interface Builder archive containing the app's view controllers, the connections between those view controllers and their immediate views, and the segues between view controllers. When this key is present, the main storyboard file is loaded automatically at launch time and its initial view controller installed in the app's window.

This key is mutually exclusive with the NSMainNibFile (page 59) key. You should include one of the keys in your `Info.plist` file but not both. This key is supported in iOS 5.0 and later.

UINewsstandApp

`UINewsstandApp` (`Boolean` - iOS) specifies the whether the app presents its content in Newsstand. Publishers of newspaper and magazine content use the Newsstand Kit framework to handle the downloading of new issues. However, instead of those apps showing up on the user's Home screen, they are collected and presented through Newsstand. This key identifies the apps that should be presented that way.

Such apps must also provide default Newsstand icons as described in Contents of the UINewsstandIcon Dictionary (page 35).

This key is supported in iOS 5.0 and later.

UIPrerenderedIcon

`UIPrerenderedIcon` (`Boolean` - iOS) specifies whether the app's icon already contains a shine effect. If the icon already has this effect, you should set this key to `YES` to prevent the system from adding the same effect again. All icons automatically receive a rounded bezel regardless of the value of this key.

Value	Description
`YES`	iOS does not apply a shine effect to the app icon.
`NO`	(Default) iOS applies a shine effect to the app icon.

UIRequiredDeviceCapabilities

`UIRequiredDeviceCapabilities` (`Array` or `Dictionary` - iOS) lets iTunes and the App Store know which device-related features an app requires in order to run. iTunes and the mobile App Store use this list to prevent customers from installing apps on a device that does not support the listed capabilities.

If you use an array, the presence of a given key indicates the corresponding feature is required. If you use a dictionary, you must specify a Boolean value for each key. If the value of this key is true, the feature is required. If the value of the key is false, the feature must not be present on the device. In both cases, omitting a key indicates that the feature is not required but that the app is able to run if the feature is present.

Dictionary keys for the UIRequiredDeviceCapabilities key lists the keys that you can include in the array or dictionary associated with the `UIRequiredDeviceCapabilities` key. You should include keys only for the features that your app absolutely requires. If your app can accommodate missing features by avoiding the code paths that use those features, do not include the corresponding key.

Table 5 Dictionary keys for the `UIRequiredDeviceCapabilities` key

Key	Description	Minimum iOS Required
`accelerometer`	Include this key if your app requires (or specifically prohibits) the presence of accelerometers on the device. Apps use the Core Motion framework to receive accelerometer events. You do not need to include this key if your app detects only device orientation changes.	iOS 3.0
`armv6`	Include this key if your app is compiled only for the armv6 instruction set.	iOS 2.0
`armv7`	Include this key if your app is compiled only for the armv7 instruction set.	iOS 3.1
`auto-focus-camera`	Include this key if your app requires (or specifically prohibits) autofocus capabilities in the device's still camera. Although most developers should not need to include this key, you might include it if your app supports macro photography or requires sharper images in order to perform some sort of image processing.	iOS 3.0
`bluetooth-le`	Include this key if your app requires (or specifically prohibits) the presence of Bluetooth low-energy hardware on the device.	iOS 5.0
`camera-flash`	Include this key if your app requires (or specifically prohibits) the presence of a camera flash for taking pictures or shooting video. Apps use the `UIImagePicker-Controller` interface to control the enabling of this feature.	iOS 3.0
`front-facing-camera`	Include this key if your app requires (or specifically prohibits) the presence of a forward-facing camera. Apps use the `UIImagePickerController` interface to capture video from the device's camera.	iOS 3.0
`gamekit`	Include this key if your app requires (or specifically prohibits) Game Center.	iOS 4.1

Key	Description	Minimum iOS Required
`gps`	Include this key if your app requires (or specifically prohibits) the presence of GPS (or AGPS) hardware when tracking locations. (You should include this key only if you need the higher accuracy offered by GPS hardware.) If you include this key, you should also include the `location-services` key. You should require GPS only if your app needs location data more accurate than the cellular or Wi-fi radios might otherwise provide.	iOS 3.0
`gyroscope`	Include this key if your app requires (or specifically prohibits) the presence of a gyroscope on the device. Apps use the Core Motion framework to retrieve information from gyroscope hardware.	iOS 3.0
`healthkit`	Include this key if your app requires (or specifically prohibits) HealthKit.	iOS 8.0
`location-services`	Include this key if your app requires (or specifically prohibits) the ability to retrieve the device's current location using the Core Location framework. (This key refers to the general location services feature. If you specifically need GPS-level accuracy, you should also include the `gps` key.)	iOS 3.0
`magnetometer`	Include this key if your app requires (or specifically prohibits) the presence of magnetometer hardware. Apps use this hardware to receive heading-related events through the Core Location framework.	iOS 3.0
`metal`	Include this key if your app requires (or specifically prohibits) Metal.	iOS 8.0
`microphone`	Include this key if your app uses the built-in microphone or supports accessories that provide a microphone.	iOS 3.0
`opengles-1`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 1.1 interfaces.	iOS 3.0
`opengles-2`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 2.0 interfaces.	iOS 3.0

Key	Description	Minimum iOS Required
`opengles-3`	Include this key if your app requires (or specifically prohibits) the presence of the OpenGL ES 3.0 interfaces.	iOS 7.0
`peer-peer`	Include this key if your app requires (or specifically prohibits) peer-to-peer connectivity over a Bluetooth network.	iOS 3.1
`sms`	Include this key if your app requires (or specifically prohibits) the presence of the Messages app. You might require this feature if your app opens URLs with the `sms` scheme.	iOS 3.0
`still-camera`	Include this key if your app requires (or specifically prohibits) the presence of a camera on the device. Apps use the `UIImagePickerController` interface to capture images from the device's still camera.	iOS 3.0
`telephony`	Include this key if your app requires (or specifically prohibits) the presence of the Phone app. You might require this feature if your app opens URLs with the `tel` scheme.	iOS 3.0
`video-camera`	Include this key if your app requires (or specifically prohibits) the presence of a camera with video capabilities on the device. Apps use the `UIImagePickerController` interface to capture video from the device's camera.	iOS 3.0
`wifi`	Include this key if your app requires (or specifically prohibits) access to the networking features of the device.	iOS 3.0

This key is supported in iOS 3.0 and later.

UIRequiresPersistentWiFi

`UIRequiresPersistentWiFi` (`Boolean` - iOS) specifies whether the app requires a Wi-Fi connection. iOS maintains the active Wi-Fi connection open while the app is running.

Value	**Description**
`YES`	iOS opens a Wi-Fi connection when this app is launched and keeps it open while the app is running. Use with Wi-Fi–based apps.
`NO`	(Default) iOS closes the active Wi-Fi connection after 30 minutes.

Note: If an iPad contains apps that use push notifications and subsequently goes to sleep, the device's active WiFi connection automatically remains associated with the current access point if cellular service is unavailable or out of range.

UIStatusBarHidden

`UIStatusBarHidden` (`Boolean` - iOS) specifies whether the status bar is initially hidden when the app launches.

Value	**Description**
`YES`	Hides the status bar.
`NO`	Shows the status bar.

UIStatusBarStyle

`UIStatusBarStyle` (`String` - iOS) specifies the style of the status bar as the app launches.

This value is based on the `UIStatusBarStyle` constants declared in `UIApplication.h` header file. The default style is `UIStatusBarStyleDefault`.

UISupportedExternalAccessoryProtocols

`UISupportedExternalAccessoryProtocols` (`Array` - iOS) specifies the protocols that your app supports and can use to communicate with external accessory hardware. Each item in the array is a string listing the name of a supported communications protocol. Your app can include any number of protocols in this list and the protocols can be in any order. The system does not use this list to determine which protocol your app should choose; it uses it only to determine if your app is capable of communicating with the accessory. It is up to your code to choose an appropriate communications protocol when it begins talking to the accessory.

This key is supported in iOS 3.0 and later. For more information about communicating with external accessories, see *External Accessory Programming Topics* .

UISupportedInterfaceOrientations

`UISupportedInterfaceOrientations` (`Array` - iOS) specifies the interface orientations your app supports. The system uses this information (along with the current device orientation) to choose the initial orientation in which to launch your app. The value for this key is an array of strings. Table 6 lists the possible string values you can include in the array.

Table 6 Supported orientations

Value	Description
`UIInterface-OrientationPortrait`	The device is in portrait mode, with the device held upright and the home button at the bottom. If you do not specify any orientations, this orientation is assumed by default.
`UIInterface-OrientationPortrait-UpsideDown`	The device is in portrait mode but upside down, with the device held upright and the home button at the top.
`UIInterface-OrientationLandscapeLeft`	The device is in landscape mode, with the device held upright and the home button on the left side.
`UIInterface-OrientationLandscapeRight`	The device is in landscape mode, with the device held upright and the home button on the right side.

This key is supported in iOS 3.2 and later.

UIViewControllerBasedStatusBarAppearance

`UIViewControllerBasedStatusBarAppearance` (Boolean - iOS) specifies whether the status bar appearance is based on the style preferred by the view controller that is currently under the status bar. When this key is not present or its value is set to `YES`, the view controller determines the status bar style. When the key is set to `NO`, view controllers (or the app) must each set the status bar style explicitly using the `UIApplication` object.

This key is supported in iOS 7.0 and later.

UIViewEdgeAntialiasing

`UIViewEdgeAntialiasing` (`Boolean` - iOS) specifies whether Core Animation layers use antialiasing when drawing a layer that is not aligned to pixel boundaries.

Value	Description
`YES`	Use antialiasing when drawing a layer that is not aligned to pixel boundaries. This option allows for more sophisticated rendering in the simulator but can have a noticeable impact on performance.
`NO`	(Default) Do not use antialiasing.

This key is supported in iOS 3.0 and later.

UIViewGroupOpacity

`UIViewGroupOpacity` (`Boolean` - iOS) specifies whether Core Animation sublayers inherit the opacity of their superlayer.

Value	Description
`YES`	(Default on iOS 7 and later) Inherit the opacity of the superlayer. This option allows for more sophisticated rendering in the simulator but can have a noticeable impact on performance.
`NO`	(Default on iOS 6 and earlier) Do not inherit the opacity of the superlayer.

This key is supported in iOS 3.0 and later.

App Extension Keys

SwiftObjective-C

App extensions enable you to provide features to other apps in iOS and OS X. Each app extension type defines keys that let you declare to the system information about an app extension's capabilities and intents.

Keys for app extensions follow a particular hierarchical structure, with the NSExtension (page 100) dictionary at the top. Most app extension types employ an NSExtensionAttributes (page 100) dictionary as an immediate child of the NSExtension key. Here is a typical structure of the keys for an app extension:

```
<key>NSExtension</key>
<dict>
  <key>NSExtensionAttributes</key>
  <dict>
    <key>NSExtensionJavaScriptPreprocessingFile</key>
    <string>Action</string>
  </dict>
  <key>NSExtensionPointIdentifier</key>
  <string>com.apple.ui-services</string>
  <key>NSExtensionPrincipalClass</key>
  <string>MyActionRequestHandler</string>
</dict>
```

The NSExtensionPointIdentifier key uniquely identifies the type of app extension, such as *Action* or *Custom Keyboard*. For a complete list of the available values for the extension point identifier key, see NSExtensionPointIdentifier (page 101).

For more information about configuring the information property-list file of an app extension, see *App Extension Programming Guide*.

Key Summary

Table 1 is an alphabetical list of app extension keys, brief descriptions, and availability by platform and version. In iOS 8 and OS X v10.10, none of these keys has an alternate Xcode name. See later sections in this chapter for detailed information on each of these keys.

Table 1 Summary of app extension keys

Key	Extension point and description	Availability
IsASCIICapable	Custom Keyboard: A Boolean value that specifies whether a custom keyboard supports the `UIKeyboardTypeASCIICapable` keyboard type trait. See IsASCIICapable (page 104) for details.	iOS 8 and later
NSExtension	General: A dictionary of keys and values that describe an app extension. See NSExtension (page 100) for details.	iOS 8 and later OS X v10.10 and later
NSExtensionAction-WantsFullScreen-Presentation	Action: A Boolean value that specifies whether the Action extension should be presented in full screen. See NSExtensionActionWantsFullScreenPresentation (page 102) for details.	iOS 8 and later
NSExtension-ActivationRule	Share, Action: A dictionary that specifies the semantic data types that a Share extension or Action extension supports. See NSExtensionActivationRule (page 102) for details.	iOS 8 and later
NSExtension-Attributes	General: A dictionary of keys and values that specify aspects of an app extension. See NSExtensionAttributes (page 100) for details.	iOS 8 and later OS X v10.10 and later
NSExtensionFile-ProviderDocument-Group	Document Picker: A string that specifies the identifier of a shared container that can be accessed by a document picker and its associated file provider. See NSExtensionFileProviderDocumentGroup (page 105) for details.	iOS 8 and later
NSExtensionJava-ScriptPreprocessing-File	Share, Action: A string that specifies the name of a JavaScript file supplied by a Share extension or an Action extension. See NSExtensionJavaScriptPreprocessingFile (page 103) for details.	iOS 8 and later OS X v10.10 and later

Key	Extension point and description	Availability
NSExtensionMain-Storyboard	General: A string that specifies the name of the app extension's main storyboard file, minus the `.storyboard` filename extension. See NSExtensionMainStoryboard (page 100) for details.	iOS 8 and later OS X v10.10 and later
NSExtensionPoint-Identifier	General: A string that specifies the extension point that supports an app extension, in reverse-DNS notation. See NSExtensionPointIdentifier (page 101) for details.	iOS 8 and later OS X v10.10 and later
NSExtensionPrincipal-Class	General: A string that specifies the name of the custom class that implements an app extension's primary view or functionality. See NSExtensionPrincipalClass (page 101) for details.	iOS 8 and later OS X v10.10 and later
NSExtensionService-AllowsToolbarItem	Action: A Boolean value that specifies whether an Action extension can display a toolbar button in a window's toolbar. See NSExtensionServiceAllowsToolbarItem (page 103) for details.	OS X v10.10 and later
NSExtensionService-RoleType	Action: A string that specifies the type of task an Action extension can perform. See NSExtensionServiceRoleType (page 103) for details.	OS X v10.10 and later
NSExtensionService-ToolbarIconFile	Action: A string that species the icon file for an OS X Action extension's toolbar button. See NSExtensionServiceToolbarIconFile (page 104) for details.	OS X v10.10 and later
NSExtensionService-ToolbarPaletteLabel	Action: A string that specifies the toolbar item title that gets displayed in the toolbar customization dialog for an Action extension. See NSExtensionServiceToolbarPaletteLabel (page 104) for details.	OS X v10.10 and later
PHSupportedMedia-Types	Photo Editing: An array that specifies the types of assets a Photo Editing extension can edit. See PHSupportedMediaTypes (page 106) for details.	iOS 8 and later
PrefersRightToLeft	Custom Keyboard: A Boolean value that specifies whether a custom keyboard is for a right-to-left language. See PrefersRightToLeft (page 105) for details.	iOS 8 and later
PrimaryLanguage	Custom Keyboard: A string conforming to BCP 47 - Tags for Identifying Languages, that specifies the primary language for custom keyboard. See PrimaryLanguage (page 105) for details.	iOS 8 and later

Key	Extension point and description	Availability
RequestsOpenAccess	Custom Keyboard: A Boolean value that specifies whether a custom keyboard wants to use a shared container and wants network access. See RequestsOpenAccess (page 105) for details.	iOS 8 and later
UIDocumentPicker-Modes	Document Picker: An array that specifies the supported document picker modes. See UIDocumentPickerModes (page 106) for details.	iOS 8 and later
UIDocumentPicker-SupportedFileTypes	Document Picker: An array that specifies the supported Uniform Type Identifiers (UTIs) for a document picker. See UIDocumentPickerSupportedFileTypes (page 106) for details.	iOS 8 and later

General App Extension Keys

The keys in this section pertain to all app extensions.

NSExtension

`Dictionary` - iOS, OS X. Keys and values that describe an app extension. Any other app extension key used for an app extension must be placed within this dictionary.

The supported direct children of this key are the NSExtensionAttributes (page 100), NSExtensionMainStoryboard (page 100), NSExtensionPointIdentifier (page 101), and NSExtensionPrincipalClass (page 101) keys.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionAttributes

`Dictionary` - iOS, OS X. Keys and values that specify aspects of an app extension. This key, if used, must be placed as an immediate child of the NSExtension (page 100) key.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionMainStoryboard

`String` - iOS, OS X. Specifies the name of the app extension's main storyboard file, minus the `.storyboard` filename extension. This key, if used, must be placed as an immediate child of the NSExtension (page 100) key.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionPointIdentifier

`String` - iOS, OS X. Specifies the extension point that supports an app extension, in reverse-DNS notation. This key is required for every app extension, and must be placed as an immediate child of the NSExtension (page 100) key. Each Xcode app extension template is preconfigured with the appropriate extension point identifier key. Valid values are shown in Table 2.

Table 2 Values for the `NSExtensionPointIdentifier` key

Extension point	Platforms	Extension point identifier
Action (UI)	iOS, OS X	`com.apple.ui-services`
Action (non-UI)	iOS	`com.apple.services`
Custom Keyboard	iOS	`com.apple.keyboard-service`
Document Picker	iOS	`com.apple.fileprovider-ui`
File Provider	iOS	`com.apple.fileprovider-nonui`
Finder Sync	OS X	`com.apple.FinderSync`
Photo Editing	iOS	`com.apple.photo-editing`
Share	iOS, OS X	`com.apple.share-services`
Today	iOS, OS X	`com.apple.widget-extension`
WatchKit App	iOS	`com.apple.watchkit`

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionPrincipalClass

`String` - iOS, OS X. Specifies the name of the custom class that implements an app extension's primary view or functionality. This key, if used, must be placed as an immediate child of the NSExtension (page 100) key.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

Action Extension Keys

The keys in this section pertain to app extensions employing the Action extension point. These keys, if used, must be placed as immediate children of the NSExtensionAttributes (page 100) key, unless noted otherwise.

NSExtensionActionWantsFullScreenPresentation

`Boolean` - iOS. Specifies whether an iOS Action extension should be presented in full screen. This key, if used, must be placed as an immediate child of the NSExtension (page 100) key.

If the value of this key is `YES`, the modal presentation style of the extension's view controller is `UIModalPresentationFullScreen`; otherwise, the modal presentation style is `UIModalPresentationFormSheet`.

This key is supported in iOS 8 and later.

NSExtensionActivationRule

`Dictionary` - iOS, OS X. Also supported in a Share extension. Specifies the semantic data types that an app extension supports. Each key in the dictionary represents a data type, such as *image*, *video*, or *web URL*. Add a key to this dictionary, and provide a nonzero value, if your app extension can handle files of the corresponding data type.

Table 3 lists the keys that you can include in the dictionary associated with the `NSExtensionActivationRule` dictionary. The value for each key is an integer that specifies the maximum number of files of the corresponding data type that your extension can handle.

Table 3 Keys for the `NSExtensionActivationRule` dictionary

Key	Description
`NSExtensionActivationSupports-AttachmentsWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports a maximum number of attachments.
`NSExtensionActivationSupports-AttachmentsWithMinCount`	Include this key to indicate to the system and to other apps that your app supports a minimum number of attachments.
`NSExtensionActivationSupports-FileWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports files in general.
`NSExtensionActivationSupports-ImageWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports image files.

Key	Description
`NSExtensionActivationSupports-MovieWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports movie files.
`NSExtensionActivationSupportsText`	Include this key to indicate to the system and to other apps that your app supports text.
`NSExtensionActivationSupports-WebURLWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports web URLs.
`NSExtensionActivationSupports-WebPageWithMaxCount`	Include this key to indicate to the system and to other apps that your app supports web pages.

If these predefined keys cannot express the activation rule you need, you can provide an entry to the NSExtensionActivationRule dictionary that is a string representing a predicate. The system evaluates the predicate against dictionary representations of `NSItemProvider` objects. (To learn more about creating a predicate statement for an activation rule, see Declaring Supported Data Types for a Share or Action Extension.)

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionJavaScriptPreprocessingFile

`String` - iOS, OS X. Also supported in a Share extension. Specifies the name of a JavaScript file supplied by a Share extension. If you provide a JavaScript file, Safari runs the functions in the file when your Share extension starts and stops. You might want to include a JavaScript file in your Share extension if you want to get information from a webpage to display in your extension, or update a webpage when your extension completes its task.

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionServiceAllowsToolbarItem

`Boolean` - OS X. Specifies whether an Action extension can display a toolbar button in a window’s toolbar. When a toolbar allows extension items, an enabled Action extension can appear in the toolbar customization dialog.

This key is supported in OS X v10.10 and later.

NSExtensionServiceRoleType

`String` - OS X. specifies the type of task an Action extension can perform. The available values for this key are shown in Table 4.

Table 4 Values for the `NSExtensionServiceRoleType` key

Value	Description
`NSExtensionService-RoleTypeEditor`	Indicates an Action extension that supports user editing and returns edited content to the host app.
`NSExtensionService-RoleTypeViewer`	Indicates an Action extension that supports viewing of content but not editing.

This key is supported in OS X v10.10 and later.

NSExtensionServiceToolbarIconFile

`String` - OS X. Species the icon file for an OS X Action extension's toolbar button. If this key is not present, the toolbar button uses the icon specified by the app extension's CFBundleIconFile (page 33) key.

This key is supported in OS X v10.10 and later.

NSExtensionServiceToolbarPaletteLabel

`String` - OS X. Specifies the toolbar item title that gets displayed in the toolbar customization dialog for an Action extension. If this key is not present, the toolbar customization dialog displays the title specified by the app extension's CFBundleDisplayName (page 24) key.

This key is supported in OS X v10.10 and later.

Custom Keyboard Extension Keys

The keys in this section pertain to app extensions employing the Custom Keyboard extension point (iOS only). These keys, if used, must be placed as immediate children of the NSExtensionAttributes (page 100) key.

IsASCIICapable

`Boolean` - iOS. Specifies whether a custom keyboard supports the `UIKeyboardTypeASCIICapable` keyboard type trait.

This key is supported in iOS 8 and later.

PrefersRightToLeft

`Boolean` - iOS. Specifies whether a custom keyboard is for a right-to-left language.

This key is supported in iOS 8 and later.

PrimaryLanguage

`String` - iOS. Specifies the primary language for a custom keyboard, using a string that conforms to BCP 47 - Tags for Identifying Languages. The system uses this value to indicate the keyboard language in Settings and in the system globe key.

This key is supported in iOS 8 and later.

RequestsOpenAccess

`Boolean` - iOS. Specifies whether a custom keyboard wants access to such resources as a shared container (shared with the containing app) and network access. Default value is `NO`. If you set this key's value to `YES`, your keyboard gains a variety of capabilities:

- Network access, supporting server-side processing of keystrokes
- Shared container access with the keyboard's containing app, supporting features such as an editing interface for the keyboard's custom autocorrect lexicon
- Access to Location Services and the Address Book database, with user approval
- Via containing app, ability to participate in Game Center and In-App Purchase
- Ability to to work with managed apps, if custom keyboard supports mobile device management (MDM)

This key is supported in iOS 8 and later.

Document Picker and File Provider Extension Keys

The keys in this section pertain to app extensions employing the Document Picker or File Provider extension points (iOS only). These keys, if used, must be placed as immediate children of the NSExtensionAttributes (page 100) key.

NSExtensionFileProviderDocumentGroup

`String` - iOS). Specifies the identifier of a shared container that can be accessed by a document picker and its associated file provider.

This key is supported in iOS 8 and later.

UIDocumentPickerModes

`Array` - iOS). Specifies the supported document picker modes.The available values for this key are shown in Table 5 (page 106).

Table 5 Values for the `UIDocumentPickerModes` key

Value	Description
`UIDocumentPickerModeImport`	Indicates that the document picker can import files.
`UIDocumentPickerModeOpen`	Indicates that the document picker can open files.
`UIDocumentPickerModeExportToService`	Indicates that the document picker can export files.
`UIDocumentPickerModeMoveToService`	Indicates that the document picker can move files.

This key is supported in iOS 8 and later.

UIDocumentPickerSupportedFileTypes

`Array` - iOS). Specifies the supported file Uniform Type Identifiers (UTIs) for a document picker. The app extension is included in a document menu only if one of the requested UTIs matches one of the UTIs in this array. The move and export modes of a document picker operate only on files with supported UTIs.

This key is supported in iOS 8 and later.

Photo Editing Extension Keys

The key in this section pertains to app extensions employing the Photo Editing extension point (iOS only). These keys, if used, must be placed as immediate children of the NSExtensionAttributes (page 100) key.

PHSupportedMediaTypes

`Array` - iOS. Specifies the types of assets a Photo Editing extension can edit. The available values for this key are `Image` (which is the default value) and `Video`.

This key is supported in iOS 8 and later.

Share Extension Keys

The keys in this section pertain to app extensions employing the Share extension point. These keys, if used, must be placed as immediate children of the NSExtensionAttributes (page 100) key.

NSExtensionActivationRule

`Dictionary` - iOS, OS X. Also supported in an Action extension. Specifies the semantic data types that an app extension supports. For complete information, see NSExtensionActivationRule (page 102).

This key is supported in iOS 8 and later and in OS X v10.10 and later.

NSExtensionJavaScriptPreprocessingFile

`String` - iOS, OS X. Also supported in an Action extension. Specifies the name of a JavaScript file supplied by a Share extension. For complete information, see NSExtensionJavaScriptPreprocessingFile (page 103).

This key is supported in iOS 8 and later and in OS X v10.10 and later.

Document Revision History

This table describes the changes to *Information Property List Key Reference*.

Date	Notes
2015-03-09	Added the NSExtensionPointIdentifier (page 101) string for a WatchKit App extension.
2014-09-17	Added new keys introduced in iOS 8.
2014-02-11	Added the opengles-3 key for required device capabilities.
2013-09-18	Added OS X v10.9 availability for NSLocationUsageDescription. Added keys introduced in iOS 7.
2012-09-19	Added keys related to the configuration of an app to support 3rd party map directions and keys related to user privacy.
2012-02-16	Added info to CFBundleURLTypes regarding RSS feeds.
2011-10-12	Added a new value to the UIRequiredDeviceCapabilities key to reflect support for low-power Bluetooth hardware. Incorporates keys introduced in iOS 5.0.
2011-06-06	Added information about key support in OS X v10.7.
2010-11-15	Corrected the availability of the LSMinimumSystemVersion and MinimumOSVersion keys.
2010-08-20	Fixed some typographical errors.
2010-07-08	Changed references of iOS to iOS.
2010-06-14	Added new keys introduced in iOS 4.0.

Date	Notes
2010-03-23	Added keys specific to iOS 3.2. Removed the `LSHasLocalizedDisplayName` key, which was deprecated in OS X v10.2.
2009-10-19	New document describing the keys you can use in a bundle's Info.plist file.