Information Property List Key Reference

General

2011-10-12

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

Figures and Tables

About Info.plist Keys

To provide a better experience for users, iOS and Mac OS X rely on the presence of special meta information in each application 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 application and the document types it supports, and some of it is used by the system frameworks to facilitate the launch of applications. The way an application 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 Application

Every application and plug-in uses an `Info.plist` file to store configuration data in a place where the system can easily access it. Mac OS X and iOS use `Info.plist` files to determine what icon to display for a bundle, what document types an application 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 19)

Launch Services Keys Describe Launch-Time Behavior

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

Relevant chapter: "Launch Services Keys" (page 37)

Cocoa Keys Describe Behavior for Cocoa and Cocoa Touch Applications

The Cocoa and Cocoa Touch frameworks use keys to identify high-level information such as your application'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 application.

Relevant chapter: "Cocoa Keys" (page 45)

Mac OS X Keys Describe Behavior for Mac OS X Applications

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

Relevant chapter: "Mac OS X Keys" (page 59)

UIKit Keys Describe Behavior for iOS Applications

An iOS application 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 applications need to augment the standard file with additional keys describing everything from the application's initial orientation to whether it supports file sharing.

Relevant chapter: "UIKit Keys" (page 63)

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 application and how it is configured. As a result, all bundled executables (plug-ins, frameworks, and applications) 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 applications, this file resides in the top-level of the bundle directory. In Mac 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.

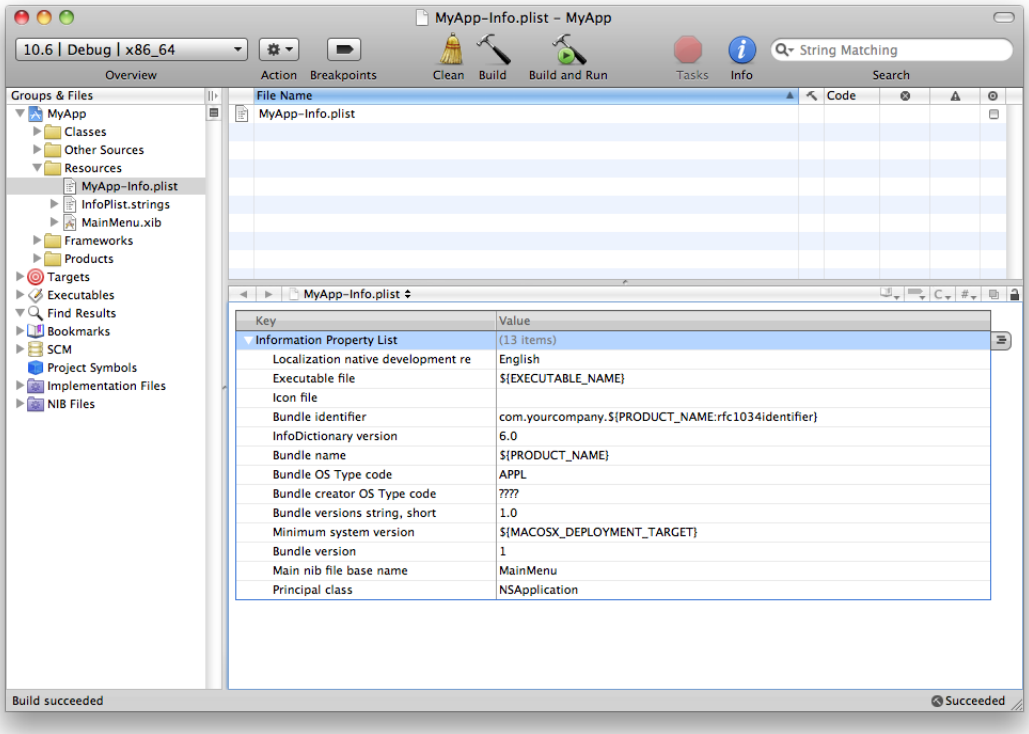
Important: In the sections that follow, pay attention to the capitalization of files and directories that reside inside a bundle. The `NSBundle` class and Core Foundation bundle functions consider case when searching for resources inside a bundle directory. Case mismatches could prevent you from finding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



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.

In addition to creating and editing property lists using Xcode, you can also create and edit them using the Property List Editor application. This application comes with Xcode and is installed in the `<Xcode>/Applications/Utilities` directory (where `<Xcode>` is the root directory of your Xcode installation).

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 applications should typically specify several additional keys. Many subsystems and system applications use the `Info.plist` file to gather information about your application. For example, when the user chooses File > Get Info for your application, the Finder displays information from many of these keys in the resulting information window.

To add keys using the Xcode property list editor, select the last item in the table and click the plus (+) button in the right margin. You can select the desired key from the list that Xcode provides or type the name of the key.

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 application, see "Recommended Info.plist Keys" (page 16).

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 application. When you localize your application, you should be sure to localize the values for these strings in addition to the rest of your application'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 application 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 Programming Topics*.

Creating Device-Specific Keys

In iOS 3.2 and later, applications 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.
- `ipod` - The key applies to iPod touch devices.
- `ipad` - The key applies to iPad devices.

When searching for a key in your application's `Info.plist` file, the system chooses the key that is most specific to the current device. For example, to indicate that you want your application to launch in a portrait orientation on iPhone and iPod touch devices but in landscape-right on iPad, you would configure your `Info.plist` with the following keys:

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

Custom Keys

Mac OS X and iOS ignore any custom keys you include in an Info.plist file. If you want to include application-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 application'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 application effectively.

Recommended Keys for iOS Applications

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

- `CFBundleDevelopmentRegion`
- `CFBundleDisplayName`
- `CFBundleExecutable`
- `CFBundleIconFiles`
- `CFBundleIdentifier`
- `CFBundleInfoDictionaryVersion`
- `CFBundlePackageType`
- `CFBundleVersion`
- `LSRequiresIPhoneOS`
- `NSMainNibFile`

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

- `UIStatusBarStyle`
- `UIInterfaceOrientation`
- `UIRequiredDeviceCapabilities`
- `UIRequiresPersistentWiFi`

Recommended Keys for Cocoa Applications

It is recommended that a Cocoa application 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`
- `CFBundleDisplayName`
- `CFBundleExecutable`
- `CFBundleIconFiles`
- `CFBundleIdentifier`
- `CFBundleInfoDictionaryVersion`
- `CFBundleName`

- `CFBundlePackageType`
- `CFBundleShortVersionString`
- `CFBundleSignature`
- `CFBundleVersion`
- `LSHasLocalizedDisplayName`
- `NSHumanReadableCopyright`

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

- `NSMainNibFile`
- `NSPrincipalClass`

Note: If you are building a Cocoa application using an Xcode template, the `NSMainNibFile` and `NSPrincipalClass` keys are typically already set in the template project.

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`
- `CFBundleName`
- `CFBundleShortVersionString`
- `NSHumanReadableCopyright`

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

Core Foundation Keys

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	Availability
CFAppleHelpAnchor	"Help file"	The bundle's initial HTML help file. See "CFAppleHelpAnchor" (page 22) for details.	Mac OS X
CFBundleAllowMixedLocalizations	"Localized resources can be mixed"	Used by Foundation tools to retrieve localized resources from frameworks. See "CFBundleAllowMixedLocalizations" (page 22) for details.	iOS, Mac OS X
CFBundleDevelopmentRegion	"Localization native development region"	(Recommended) The native region for the bundle. Usually corresponds to the native language of the author. See "CFBundleDevelopmentRegion" (page 22) for details.	iOS, Mac OS X
CFBundleDisplayName	"Bundle display name"	(Recommended, Localizable) The actual name of the bundle. See "CFBundleDisplayName" (page 22) for details.	iOS, Mac OS X
CFBundleDocumentTypes	"Document types"	An array of dictionaries describing the document types supported by the bundle. See "CFBundleDocumentTypes" (page 23) for details.	iOS, Mac OS X

Key	Xcode name	Summary	Availability
CFBundleExecutable	"Executable file"	(Recommended) Name of the bundle's executable file. See "CFBundleExecutable" (page 28) for details.	iOS, Mac OS X
CFBundleGetInfoString	"Get Info string"	Deprecated. Use the `CFBundleShortVersionString` and `NSHumanReadableCopyright` keys instead.	Mac OS X
CFBundleHelpBookFolder	"Help Book directory name"	The name of the folder containing the bundle's help files. See "CFBundleHelpBookFolder" (page 28) for details.	Mac 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 28) for details.	Mac OS X
CFBundleIconFile	"Icon file"	A legacy way to specify the application's icon. Use the "CFBundleIcons" (page 29) or "CFBundleIconFiles" (page 29) keys instead. See "CFBundleIconFile" (page 28) for details.	iOS, Mac OS X
CFBundleIconFiles	"Icon files"	A top-level key for specifying the file names of the bundle's icon image files. See "CFBundleIconFiles" (page 29) for details. See also "CFBundleIcons" (page 29) 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 29) for details.	iOS 5.0 and later
CFBundleIdentifier	"Bundle identifier"	(Recommended) An identifier string that specifies the application 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 31) for details.	iOS, Mac OS X
CFBundleInfoDictionaryVersion	"InfoDictionary version"	(Recommended) Version information for the `Info.plist` format. See "CFBundleInfoDictionaryVersion" (page 32) for details.	iOS, Mac OS X
CFBundleLocalizations	"Localizations"	Contains localization information for an application that handles its own localized resources. See "CFBundleLocalizations" (page 32) for details.	iOS, Mac OS X

Key	Xcode name	Summary	Availability
CFBundleName	"Bundle name"	(Recommended, Localizable) The short display name of the bundle. See "CFBundleName" (page 32) for details.	iOS, Mac OS X
CFBundlePackageType	"Bundle OS Type code"	The four-letter code identifying the bundle type. See "CFBundlePackageType" (page 32) for details.	iOS, Mac OS X
CFBundleShortVersionString	"Bundle versions string, short"	(Localizable) The release-version-number string for the bundle. See "CFBundleShortVersionString" (page 33) for details.	iOS, Mac OS X
CFBundleSignature	"Bundle creator OS Type code"	The four-letter code identifying the bundle creator. See "CFBundleSignature" (page 33) for details.	iOS, Mac OS X
CFBundleURLTypes	"URL types"	An array of dictionaries describing the URL schemes supported by the bundle. See "CFBundleURLTypes" (page 33) for details.	iOS, Mac OS X
CFBundleVersion	"Bundle version"	(Recommended) The build-version-number string for the bundle. See "CFBundleVersion" (page 34) for details.	iOS, Mac OS X
CFPlugInDynamicRegistration	"Plug-in should be registered dynamically"	If YES, register the plug-in dynamically; otherwise, register it statically. See "CFPlugInDynamicRegistration" (page 34) for details.	Mac OS X
CFPlugInDynamicRegistrationFunction	Plug-in dynamic registration function name"	The name of the custom, dynamic registration function. See "CFPlugInDynamicRegisterFunction" (page 34) for details.	Mac OS X
CFPlugInFactories	"Plug-in factory interfaces"	For static registration, this dictionary contains a list of UUIDs with matching function names. See "CFPlugInFactories" (page 34) for details.	Mac OS X
CFPlugInTypes	"Plug-in types"	For static registration, the list of UUIDs "CFPlugInTypes" (page 35) for details.	Mac 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 35) for details.	Mac OS X

CFAppleHelpAnchor

`CFAppleHelpAnchor` (`String` - Mac 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, Mac 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, Mac OS X) specifies the native region for the bundle. This key contains a string value that usually corresponds to the native language of the person who wrote the bundle. The language specified by this value is used as the default language if a resource cannot be located for the user's preferred region or language.

CFBundleDisplayName

`CFBundleDisplayName` (`String` - iOS, Mac 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 Mac OS X, see *File System Programming Guide*.

CFBundleDocumentTypes

`CFBundleDocumentTypes` (`Array` - iOS, Mac OS X) contains an array of dictionaries that associate one or more document types with your application. 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. For additional information about specifying the types your application supports, see "Storing Document Types Information in the Application's Property List".

Table 2 Keys for type-definition dictionaries

Key	Xcode name	Type	Description	Platforms
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 Mac OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in Mac OS X v10.5.	Mac OS X
CFBundleTypeIconFile	"Icon File Name"	`String`	This key contains a string with the name of the icon file (`.icns`) to associate with this Mac OS X document type. For more information about specifying document icons, see "Document Icons" (page 27).	Mac 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 27).	iOS
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 Mac OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in Mac OS X v10.5.	Mac OS X

Key	Xcode name	Type	Description	Platforms
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, Mac OS X
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 Mac OS X v10.4, this key is ignored if the `LSItemContentTypes` key is present.) Deprecated in Mac OS X v10.5.	Mac OS X
CFBundleTypeRole	"Role"	`String`	This key specifies the application's role with respect to the type. The value can be `Editor`, `Viewer`, `Shell`, or `None`. This key is required.	Mac OS X
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 Mac OS X v10.5 and later, this key (when present) takes precedence over these type-identifier keys: `CFBundleTypeExtensions`, `CFBundleTypeMIMETypes`, `CFBundleTypeOSTypes`.	iOS, Mac OS X

Key	Xcode name	Type	Description	Platforms
LSHandlerRank	"Handler rank"	`String`	Determines how Launch Services ranks this application among the applications that declare themselves editors or viewers of files of this type. The possible values are: `Owner` (this application is the creator of files of this type), `Alternate` (this application is a secondary viewer of files of this type), `None` (this application must never be used to open files of this type, but it accepts drops of files of this type), `Default` (default; this application doesn't accept drops of files of this type). Launch Services uses the value of `LSHandlerRank` to determine the application to use to open files of this type. The order of precedence is: `Owner`, `Alternate`, `None`. This key is available in Mac OS X v10.5 and later.	iOS, Mac OS X
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 Mac OS X v10.4 and later, this key is ignored if the `LSItemContentTypes` key is present.)	Mac 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 applications only.	Mac OS X
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 applications only. Deprecated in Mac OS X v10.5.	Mac OS X

Key	Xcode name	Type	Description	Platforms
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 applications only. Available in Mac OS X v10.5 and later.	Mac OS X

The way you specify icon files in Mac 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 Mac 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 Mac OS X. For detailed information about UTIs, see *Uniform Type Identifiers Overview*.

Document Roles

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

- **Editor**. The application can read, manipulate, and save the type.
- **Viewer**. The application can read and present data of that type.
- **Shell**. The application 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 application), and a new process is created for each document opened.
- **None**. The application 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 application 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
iPhone and iPod touch	22 x 29 pixels 44 x 58 pixels (high resolution)

In Mac 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 application 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 Mac 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, Mac OS X) identifies the name of the bundle's main executable file. For an application, this is the application 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. Mac OS X uses this key to locate the bundle's executable or shared library in cases where the user renames the application or bundle directory.

CFBundleGetInfoString

`CFBundleGetInfoString` (`String` - Mac OS X) provides a brief description of the bundle.

The use of this key is deprecated. Instead, use the `CFBundleShortVersionString` key to specify your bundle's human-readable version information and the `NSHumanReadableCopyright` key to specify copyright information.

CFBundleHelpBookFolder

`CFBundleHelpBookFolder` (`String` - Mac 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` - Mac OS X) identifies the main help page for your application. 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, Mac 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 OS X application uses a custom icon, you must specify this property. If you do not specify this property, the system (and other applications) display your bundle with a default icon.

Note: If you are writing an iOS application, you should prefer the use of the "CFBundleIconFiles" (page 29) 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 28) key.

This key is supported in iOS 3.2 and later only and an application may have differently sized icons to support different types of devices and different screen resolutions. In other words, an application 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 key does not matter. The system automatically chooses the most appropriately sized icon based on the usage and the underlying device type.

For a list of the icons, including their sizes, that you can include in your application bundle, see the section on application icons in "Build-Time Configuration Details" in *iOS Application Programming Guide*. For information about how to create icons for your applications, see *iOS Human Interface Guidelines*.

CFBundleIcons

`CFBundleIcons` (`Dictionary` - iOS) contains information about all of the icons used by the application. 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 applications among others. The value for this key is a dictionary whose contents are described in "Contents of the CFBundlePrimaryIcon Dictionary" (page 30).
- **UINewsstandIcon**—This key identifies default icons to use for applications presented from Newsstand. The value for this key is a dictionary whose contents are described in "Contents of the UINewsstandIcon Dictionary" (page 30).

The `CFBundleIcons` key is supported in iOS 5.0 and later. You may combine this key with the "CFBundleIconFiles" (page 29) and "CFBundleIconFile" (page 28) 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 application bundle. The icons in this dictionary are used to represent the application on the device's Home screen and in the Settings application. 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 applications. For a list of the icons, including their sizes, that you can include in your application bundle, see the section on application icons in "Build-Time Configuration Details" in *iOS Application Programming Guide*. For information about how to create icons for your applications, 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 applications 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 default icons for your application 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 application bundle, see the section on application icons in "Build-Time Configuration Details" in *iOS Application Programming Guide*. For information about how to create icons for your applications, 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, Mac OS X) uniquely identifies the bundle. Each distinct application or bundle on the system must have a unique bundle ID. The system uses this string to identify your application in many ways. For example, the preferences system uses this string to identify the application for which a given preference applies; Launch Services uses the bundle identifier to locate an application capable of opening a particular file, using the first application it finds with the given identifier; in iOS, the bundle identifier is used in validating the application'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 application named Hello, you could assign the string `com.Ajax.Hello` as your application's bundle identifier.

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

CFBundleInfoDictionaryVersion

`CFBundleInfoDictionaryVersion` (`String` - iOS, Mac 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, Mac OS X) identifies the localizations handled manually by your application. If your executable is unbundled or does not use the existing bundle localization mechanism, you can include this key to specify the localizations your application 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 Programming Topics* in Internationalization Documentation for information on how to specify language designators.

CFBundleName

`CFBundleName` (`String` - iOS, Mac 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 application'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 22).

CFBundlePackageType

`CFBundlePackageType` (`String` - iOS, Mac 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 applications 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, Mac OS X) specifies the **release version number** of the bundle, which identifies a released iteration of the application. The release version number is a string comprised of three period-separated integers. The first integer represents major revisions to the application, 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 34), which identifies an iteration (released or unreleased) of the application. This key can be localized by including it in your `InfoPlist.strings` files.

CFBundleSignature

`CFBundleSignature` (`String` - iOS, Mac 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 application is `ttxt`.

CFBundleURLTypes

`CFBundleURLTypes` (`Array` - iOS, Mac OS X) contains an array of dictionaries, each of which describes the URL schemes (`http`, `ftp`, and so on) supported by the application. The purpose of this key is similar to that of "CFBundleDocumentTypes" (page 23), 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 application's role with respect to the URL type. The value can be `Editor`, `Viewer`, `Shell`, or `None`. This key is required.	iOS, Mac 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, Mac 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, Mac 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. Examples of URL schemes include `http`, `ftp`, `mailto`, and so on.	iOS, Mac OS X

CFBundleVersion

`CFBundleVersion` (`String` - iOS, Mac OS X) specifies the **build version number** of the bundle, which identifies an iteration (released or unreleased) of the bundle. This is a monotonically increased string, comprised of one or more period-separated integers. This key is not localizable.

CFPlugInDynamicRegistration

`CFPlugInDynamicRegistration` (`String` - Mac 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 34), and "CFPlugInTypes" (page 35) keys.

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

CFPlugInDynamicRegisterFunction

`CFPlugInDynamicRegisterFunction` (`String` - Mac 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-ins*.

CFPlugInFactories

`CFPlugInFactories` (`Dictionary` - Mac 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-ins*.

CFPlugInTypes

`CFPlugInTypes` (`Dictionary` - Mac 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 34) dictionary.

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

CFPlugInUnloadFunction

`CFPlugInUnloadFunction` (`String` - Mac 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-ins*.

Launch Services Keys

Launch Services (part of the Core Services framework in Mac OS X) provides support for launching applications and matching document types to applications. 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 applications.)

Launch Services keys use the prefix `LS` to distinguish them from other keys. For more information about Launch Services in Mac 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	Avail
LSApplicationCategoryType	"Application Category"	Contains a string with the UTI that categorizes the application for the App Store. See "LSApplicationCategoryType" (page 38) for details.	Mac
LSArchitecturePriority	"Architecture priority"	Contains an array of strings identifying the supported code architectures and their preferred execution priority. See "LSArchitecturePriority" (page 40) for details.	Mac
LSBackgroundOnly	"Application is background only"	Specifies whether the application runs only in the background. (Mach-O applications only). See "LSBackgroundOnly" (page 41) for details.	Mac
LSEnvironment	"Environment variables"	Contains a list of key/value pairs, representing environment variables and their values. See "LSEnvironment" (page 41) for details.	Mac
LSFileQuarantineEnabled	"File quarantine enabled"	Specifies whether the files this application creates are quarantined by default. See "LSFileQuarantineEnabled" (page 41).	Mac
LSFileQuarantineExcludedPathPatterns	None	Specifies directories for which files should not be automatically quarantined. See "LSFileQuarantineExcludedPathPatterns" (page 42).	Mac

Key	Xcode name	Summary	Avail
LSGetAppDiedEvents	"Application should get App Died events"	Specifies whether the application is notified when a child process dies. See "LSGetAppDiedEvents" (page 42) for details.	Mac
LSMinimumSystemVersion	"Minimum system version"	Specifies the minimum version of Mac OS X required for the application to run. See "LSMinimumSystemVersion" (page 42) for details.	Mac
LSMinimumSystemVersionByArchitecture	"Minimum system versions, per-architecture"	Specifies the minimum version of Mac OS X required to run a given platform architecture. See "LSMinimumSystemVersionByArchitecture" (page 42) for details.	Mac
LSMultipleInstancesProhibited	"Application prohibits multiple instances"	Specifies whether one user or multiple users can launch an application simultaneously. See "LSMultipleInstancesProhibited" (page 43) for details.	Mac
LSRequiresIPhoneOS	"Application requires iPhone environment"	Specifies whether the application is an iOS application. See "LSRequiresIPhoneOS" (page 43) for details.	iOS
LSRequiresNativeExecution	"Application requires native environment"	Specifies whether the application must run natively on Intel-based Macintosh computers, as opposed to under Rosetta emulation. See "LSRequiresNativeExecution" (page 43) for details.	Mac
LSUIElement	"Application is agent (UIElement)"	Specifies whether the application is an agent application, that is, an application that should not appear in the Dock or Force Quit window. See "LSUIElement" (page 43) for details.	Mac
LSUIPresentationMode	"Application UI Presentation Mode"	Sets the visibility of system UI elements when the application launches. See "LSUIPresentationMode" (page 44) for details.	Mac
LSVisibleInClassic	"Application is visible in Classic"	Specifies whether an agent application or background-only application is visible to other applications in the Classic environment. See "LSVisibleInClassic" (page 44) for details.	Mac
MinimumOSVersion	"Minimum system version"	Do not use. Use "LSMinimumSystemVersion" (page 42) instead.	iOS

LSApplicationCategoryType

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

Table 2 UTIs for application 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`
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`

Category	UTI
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`
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` - Mac OS X) is an array of strings that identifies the architectures this application 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.

String	Description
`ppc64`	The 64-bit PowerPC architecture.

if a PowerPC architecture appears before either of the Intel architectures, Mac OS X runs the executable under Rosetta emulation on Intel-based Macintosh computers regardless by default. To force Mac OS X to use the current platform's native architecture, include the "LSRequiresNativeExecution" (page 43) key in your information property list.

LSBackgroundOnly

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

LSEnvironment

`LSEnvironment` (`Dictionary` - Mac OS X) defines environment variables to be set before launching this application. 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 applications launched through Launch Services. If you run your executable directly from the command line, these environment variables are not set.

LSFileQuarantineEnabled

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

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

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

LSFileQuarantineExcludedPathPatterns

`LSFileQuarantineExcludedPathPatterns` (`Array` - Mac 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 application. 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 application in the user's cache directory.

LSGetAppDiedEvents

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

LSMinimumSystemVersion

`LSMinimumSystemVersion` (`String` - Mac OS X) indicates the minimum version of Mac OS X required for this application 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 Mac 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, Mac OS X tries to display an alert panel notifying the user of that fact.

LSMinimumSystemVersionByArchitecture

`LSMinimumSystemVersionByArchitecture` (`Dictionary` - Mac OS X) specifies the earliest Mac 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 40) key. The value for each key is the minimum version of Mac OS X required for the application 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 Mac 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 42) key, which specifies the overall minimum system version requirement for the application.

LSMultipleInstancesProhibited

`LSMultipleInstancesProhibited` (`Boolean` - Mac OS X) indicates whether an application is prohibited from running simultaneously in multiple user sessions. If true, the application runs in only one user session at a time. You can use this key to prevent resource conflicts that might arise by sharing an application 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 application cannot be launched. If a user in another session is running the application, Launch Services returns a `kLSMultipleSessionsNotSupportedErr` error. If you attempt to launch a separate instance of an application in the current session, it returns `kLSMultipleInstancesProhibitedErr`.

LSRequiresIPhoneOS

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

LSRequiresNativeExecution

`LSRequiresNativeExecution` (`Boolean` - Mac OS X) specifies whether to launch the application using the subbinary for the current architecture. If this key is set to `YES`, Launch Services always runs the application 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 Macintosh computer. For more information about configuring the execution architectures, see "LSArchitecturePriority" (page 40).

LSUIElement

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

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

LSUIPresentationMode

`LSUIPresentationMode` (`Number` - Mac OS X) identifies the initial user-interface mode for the application. You would use this in applications 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 applications 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 Mac OS X 10.3 and later.

LSVisibleInClassic

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

MinimumOSVersion

`MinimumOSVersion` (`String` - iOS). When you build an iOS application, 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 application to the App Store, the store indicates the iOS releases on which your application can run based on this key. It is equivalent to the `LSMinimumSystemVersion` key in Mac OS X.

Cocoa Keys

Cocoa and Cocoa Touch are the environments used to define Objective-C based applications that run in Mac 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 applications for iOS, see *iOS Application Programming Guide*. For information about developing Cocoa applications for Mac 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	Availability
NSAppleScriptEnabled	"Scriptable"	Specifies whether AppleScript is enabled. See "NSAppleScriptEnabled" (page 47) for details.	Mac OS X
NSDockTilePlugIn	"Dock Tile Plugin path"	Specifies the name of application's Dock tile plug-in, if present. See "NSDockTilePlugIn" (page 47) for details.	Mac OS X
NSHumanReadableCopyright	"Copyright (human-readable)"	(Localizable) Specifies the copyright notice for the bundle. See "NSHumanReadableCopyright" (page 47) for details.	Mac OS X
NSJavaNeeded	"Cocoa Java application"	Specifies whether the program requires a running Java VM. See "NSJavaNeeded" (page 47) for details.	Mac OS X
NSJavaPath	"Java classpaths"	An array of paths to classes whose components are preceded by `NSJavaRoot`. See "NSJavaPath" (page 47) for details.	Mac OS X
NSJavaRoot	"Java root directory"	The root directory containing the java classes. See "NSJavaRoot" (page 48) for details.	Mac OS X

Key	Xcode name	Summary	Availability
NSMainNibFile	"Main nib file base name"	The name of an application's main nib file. See "NSMainNibFile" (page 48) for details.	iOS, Mac OS X
NSPersistentStoreTypeKey	"Core Data persistent store type"	The type of Core Data persistent store associated with a persistent document type. See "NSPersistentStoreTypeKey" (page 48) for details.	Mac OS X
NSPrefPaneIconFile	"Preference Pane icon file"	The name of an image file resource used to represent a preference pane in the System Preferences application. See "NSPrefPaneIconFile" (page 48) for details.	Mac OS X
NSPrefPaneIconLabel	"Preference Pane icon label"	The name of a preference pane displayed beneath the preference pane icon in the System Preferences application. See "NSPrefPaneIconLabel" (page 48) for details.	Mac OS X
NSPrincipalClass	"Principal class"	The name of the bundle's main class. See "NSPrincipalClass" (page 49) for details.	Mac OS X
NSServices	"Services"	An array of dictionaries specifying the services provided by an application. See "NSServices" (page 49) for details.	Mac OS X
NSSupportsAutomaticTermination	None	Specifies whether the application may be killed to reclaim memory. See "NSSupportsAutomaticTermination" (page 54) for details.	Mac OS X 10.7 and later
NSSupportsSuddenTermination	None	Specifies whether the application may be killed to allow for faster shut down or log out operations. See "NSSupportsSuddenTermination" (page 54) for details.	Mac OS X
NSUbiquitousDisplaySet	None	Specifies the mobile document data that the application can view. See "NSUbiquitousDisplaySet" (page 55) for details.	iOS, Mac OS X
UTExportedTypeDeclarations	"Exported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (and owned) by the application. See "UTExportedTypeDeclarations" (page 55) for details.	iOS 5.0 and later, Mac OS X 10.7 and later
UTImportedTypeDeclarations	"Imported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (but not owned) by the application. See "UTImportedTypeDeclarations" (page 57) for details.	iOS, Mac OS X

NSAppleScriptEnabled

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

NSDockTilePlugIn

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

NSHumanReadableCopyright

`NSHumanReadableCopyright` (`String` - Mac 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` - Mac 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 applications to tell the system to launch the Java environment. If you are writing a pure Java application, 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 Mac OS X v10.5.

NSJavaPath

`NSJavaPath` (`Array` - Mac 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 48). The development environment (or, specifically, its jamfiles) automatically maintains the values in the array.

Deprecated in Mac OS X v10.5.

NSJavaRoot

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

NSMainNibFile

`NSMainNibFile` (`String` - iOS, Mac OS X). This key contains a string with the name of the application'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 application is launched.

This key is mutually exclusive with the "UIMainStoryboardFile" (page 67) key. You should include one of the keys in your `Info.plist` file but not both.

NSPersistentStoreTypeKey

`NSPersistentStoreTypeKey` (`String` - Mac OS X). This key contains a string that specifies the type of Core Data persistent store associated with a document type (see "CFBundleDocumentTypes" (page 23)).

NSPrefPaneIconFile

`NSPrefPaneIconFile` (`String` - Mac 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 application looks for the image file using the `CFBundleIconFile` key instead.

NSPrefPaneIconLabel

`NSPrefPaneIconLabel` (`String` - Mac 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 application. 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 application 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` - Mac 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 applications, the value for this key is `NSApplication` by default.

NSServices

`NSServices` (`Array` - Mac OS X). This key contains an array of dictionaries specifying the services provided by the application. 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 application monitors for incoming service requests. Its value depends on how the service provider application is registered. In most cases, this is the application name. For more information, see *Services Implementation Guide*.	Mac OS X
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(NSPasteBoard,String)`.	Mac OS X

Key	Xcode name	Type	Description	Platforms
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 Mac OS X v10.6 and later. For information on UTIs, see *Uniform Type Identifiers Overview*.	Mac OS X
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 Mac OS X v10.5 and earlier, this key is required. In Mac OS X v10.6 and later, you should use the `NSSendFileTypes` key instead.	Mac 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 Mac OS X v10.6 and later.	Mac 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 Mac OS X v10.6 and later.	Mac OS X
NSRestricted	None	`Boolean`	Specifying a value of `true` for this key prevents the service from being invoked by a sandboxed application. You should set the value to `true` if your service performs privileged or potentially dangerous operations that would allow a sandboxed application 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 Mac OS X v10.7 and later.	Mac OS X
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.	Mac OS X

Key	Xcode name	Type	Description	Platforms
NSMenuItem	"Menu"	`Dictionary`	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 Mac OS X v10.5 and earlier, menu items must be unique. You can ensure a unique name by combining the application 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 Mac OS X v10.6 and later. If you specify a slash character in Mac 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 application 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 the translated menu item text as its value. For more information about creating strings files, see *Resource Programming Guide*.	Mac OS X
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.	Mac OS X

Key	Xcode name	Type	Description	Platforms
NSUserData	"User Data"	`String`	This key is an optional string that contains a value of your choice.	Mac 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 application providing a service when a response is required.	Mac OS X

In Mac 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 application. Your service appears only if the bundle ID of the current application matches one of the specified values.	Mac 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.	Mac 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`.	Mac 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.	Mac 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 application, not just the parts found to contain the given data types.	Mac OS X

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

NSSupportsAutomaticTermination

`NSSupportsAutomaticTermination` (`Boolean` - Mac OS X). This key contains a boolean that indicates whether the application supports automatic termination in Mac OS X 10.7 and later. Automatic termination allows an application that is running to be terminated automatically by the system when certain conditions apply. Primarily, the application 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 application in order to reclaim the memory used by the application.

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

NSSupportsSuddenTermination

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

NSUbiquitousDisplaySet

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

If you create multiple applications, you can use the same display set for your applications or assign different display sets to each. For example, if you create a lite version of your application, 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 application should recognize the file types stored in its mobile data folder and be able to open them.

UTExportedTypeDeclarations

`UTExportedTypeDeclarations` (`Array` - iOS, Mac OS X) declares the uniform type identifiers (UTIs) owned and exported by the application. You use this key to declare your application'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 Mac OS X v10.5 and later. This key is ignored on versions of Mac 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 57) key.

Table 4 UTI property list keys

Key	Xcode name	Type	Description	Platforms
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*.	iOS, Mac OS X
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.	iOS, Mac OS X
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 application exports. This file should have a `.icns` filename extension. You can create this file using the Icon Composer application that comes with Xcode Tools.	Mac OS X

Key	Xcode name	Type	Description	Platforms
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>`.	iOS, Mac OS X
UTTypeReferenceURL	"Reference URL"	`String`	The URL for a reference document that describes this type.	Mac OS X
UTTypeSize64IconFile	None	`String`	The name of the 64 x 64 pixel icon resource file (located in the application's bundle) to associate with this UTI. You should include this key only for types that your application exports.	iOS
UTTypeSize320IconFile	None	`String`	The name of the 320 x 320 pixel icon resource file (located in the application's bundle) to associate with this UTI. You should include this key only for types that your application exports.	iOS
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*.	iOS, Mac OS X

The way you specify icon files in Mac 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 Mac 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 Mac OS X 10.5 and later. For more information about UTIs and their use, see *Uniform Type Identifiers Overview*.

UTImportedTypeDeclarations

`UTImportedTypeDeclarations` (`Array` - iOS, Mac OS X) declares the uniform type identifiers (UTIs) inherently supported (but not owned) by the application. You use this key to declare any supported types that your application recognizes and wants to ensure are recognized by Launch Services, regardless of whether the application 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 55) key. For a list of these keys, see Table 4 (page 55).

This key is supported in iOS 3.2 and later and Mac OS X 10.5 and later. For more information about UTIs and their use, see *Uniform Type Identifiers Overview*.

Mac OS X Keys

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

Key Summary

Table 1 contains an alphabetical listing of Mac OS X–specific 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 Mac OS X keys

Key	Xcode name	Summary	Availability
APInstallerURL	"Installation directory base file URL"	A URL-based path to the files you want to install. See "APInstallerURL" (page 59) for details.	Mac OS X
APFiles	"Installation files"	An array of dictionaries describing the files or directories that can be installed. See "APFiles" (page 60) for details.	Mac OS X
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 60) for details.	Mac OS X
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 61) for details.	Mac OS X
QuartzGLEnable	None	Specifies whether the application uses Quartz GL. See "QuartzGLEnable" (page 61) for details.	Mac OS X 10.5 and later

APInstallerURL

`APInstallerURL` (`String` - Mac 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` - Mac 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	Platform
APFileDescriptionKey	"Install file description text"	`String`	A short description of the item to display in the Finder's Info window	Mac OS X
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	Mac OS X
APFileDestinationPath	"File destination path"	`String`	Where to install the component as a path relative to the application bundle	Mac OS X
APFileName	"Install file name"	`String`	The name of the file or directory	Mac OS X
APFileSourcePath	"Install file source path"	`String`	The path to the component in the application package relative to the `APInstallerURL` path.	Mac OS X
APInstallAction	"File install action"	`String`	The action to take with the component: "Copy" or "Open"	Mac OS X

ATSApplicationFontsPath

`ATSApplicationFontsPath` (`String` - Mac OS X) identifies the location of a font file or directory of fonts in the bundle's `Resources` directory. If present, Mac OS X activates the fonts at the specified path for use by the bundled application. The fonts are activated only for the bundled application 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

`CSResourcesFileMappedBoolean` - Mac OS X) specifies whether to map this application'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.

QuartzGLEnable

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

Value	Description
`true`	Turn on Quartz GL for the application's windows. (This works only when the computer has at least 1GM 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 application, located in `<Xcode>/Applications`.

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

UIKit Keys

The UIKit framework provides the infrastructure you need for creating iOS applications. You use the keys associated with this framework to configure the appearance of your application at launch time and the behavior of your application once it is running.

UIKit keys use the prefix `UI` to distinguish them from other keys. For more information about using UIKit to create and configure iOS applications, see *iOS Application Programming Guide*.

Key Summary

Table 1 contains an alphabetical listing of UIKit 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
UIAppFonts	"Fonts provided by application"	Specifies a list of application-specific fonts. See "UIAppFonts" (page 65) for details.	iOS 3.2 and later
UIApplicationExitsOnSuspend	"Application does not run in background"	Specifies whether the application terminates instead of run in the background. See "UIApplicationExitsOnSuspend" (page 65) for details.	iOS 4.0 and later
UIBackgroundModes	"Required background modes"	Specifies that the application needs to continue running in the background. See "UIBackgroundModes" (page 65) for details.	iOS 4.0 and later
UIDeviceFamily	"Targeted device family"	Inserted automatically by Xcode to define the target device of the application. See "UIDeviceFamily" (page 66) for details.	iOS 3.2 and later
UIFileSharingEnabled	"Application supports iTunes file sharing"	Specifies whether the application shares files with the user's computer through iTunes. See "UIFileSharingEnabled" (page 66) for details.	iOS 3.2 and later
UIInterfaceOrientation	"Initial interface orientation"	Specifies the initial orientation of the application's user interface. See "UIInterfaceOrientation" (page 67) for details.	iOS

Key	Xcode name	Summary	Availability
UILaunchImageFile	"Launch image"	Specifies the name of the application's launch image. See "UIMainStoryboardFile" (page 67) for details.	iOS 3.2 and later
UIMainStoryboardFile	None	Specifies the name of the application's storyboard resource file. See "UIMainStoryboardFile" (page 67) for details.	iOS 5.0 and later
UINewsstandApp	None	Specifies whether the application presents its content in Newsstand. See "UINewsstandApp" (page 67) for details.	iOS 5.0 and later
UIPrerenderedIcon	"Icon already includes gloss effects"	Specifies whether the application's icon already includes a shine effect. See "UIPrerenderedIcon" (page 68) for details.	iOS
UIRequiredDeviceCapabilities	"Required device capabilities"	Specifies the device-related features required for the application to run. See "UIRequiredDeviceCapabilities" (page 68) for details.	iOS 3.0 and later
UIRequiresPersistentWiFi	"Application uses Wi-Fi"	Specifies whether this application requires a Wi-Fi connection. See "UIRequiresPersistentWiFi" (page 70) for details.	iOS
UIStatusBarHidden	"Status bar is initially hidden"	Specifies whether the status bar is initially hidden when the application launches. See "UIStatusBarHidden" (page 70) for details.	iOS
UIStatusBarStyle	"Status bar style"	Specifies the style of the status bar as the application launches. See "UIStatusBarStyle" (page 71) for details.	iOS
UISupportedExternalAccessoryProtocols	"Supported external accessory protocols"	Specifies the communications protocols supported for communication with attached hardware accessories. See "UISupportedExternalAccessoryProtocols" (page 71) for details.	iOS 3.0 and later
UISupportedInterfaceOrientations	"Supported interface orientations"	Specifies the orientations that the application supports. See "UISupportedInterfaceOrientations" (page 71) for details.	iOS 3.2 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 72) 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 72) for details.	iOS 3.0 and later

UIAppFonts

`UIAppFonts` (`Array` - iOS) specifies any application-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 application's bundle. The system loads the specified fonts and makes them available for use by the application when that application is run.

This key is supported in iOS 3.2 and later.

UIApplicationExitsOnSuspend

`UIApplicationExitsOnSuspend` (`Boolean` - iOS) specifies that the application should be terminated rather than moved to the background when it is quit. Applications 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 application suspension. When the value of this key is `YES`, the application is terminated and purged from memory instead of moved to the background. If this key is not present, or is set to `NO`, the application moves to the background as usual.

This key is supported in iOS 4.0 and later.

UIBackgroundModes

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

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 application must provide the indicated services.

Table 2 Values for the `UIBackgroundModes` array

Value	Description
`audio`	The application plays audible content in the background.
`location`	The application 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.
`voip`	The application provides Voice-over-IP services. Applications with this key are automatically launched after system boot so that the application can reestablish VoIP services. Applications with this key are also allowed to play background audio.

Value	Description
`newsstand-content`	The application 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 application communicates with an accessory that delivers data at regular intervals. This value is supported in iOS 5.0 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 application 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 application runs on iPhone and iPod touch devices.
2	The application runs on iPad devices.

This key is supported in iOS 3.2 and later.

UIFileSharingEnabled

`UIFileSharingEnabled` (`Boolean` - iOS) specifies whether the application shares files through iTunes. If this key is `YES`, the application shares files. If it is not present or is `NO`, the application does not share files. Applications 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 application's home directory.

In iTunes, the user can access an application'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 application's user interface.

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 application. If this key is not specified, the system assumes a name of `Default.png`. This key is typically used by universal applications when different sets of launch images are needed for iPad versus iPhone or iPod touch devices.

If you include this key in your `Info.plist` file, any launch images you include in your application's bundle should be based on the string. For example, suppose you want to include portrait and landscape launch images for iPad using the base name `MyiPadImage.png`. You would include the `UILaunchImageFile~ipad` key in your `Info.plist` file and set its value to `MyiPadImage.png`. You would then include a `MyiPadImage-Portrait.png` file and a `MyiPadImage-Landscape.png` file in your bundle to specify the corresponding launch images.

This key is supported in iOS 3.2 and later.

UIMainStoryboardFile

`UIMainStoryboardFile` (`String` - iOS) contains a string with the name of the application's main storyboard file (minus the `.storyboard` extension). A storyboard file is an Interface Builder archive containing the application'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 application's window.

This key is mutually exclusive with the "NSMainNibFile" (page 48) 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 application 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 applications showing up on the user's Home screen, they are collected and presented through Newsstand. This key identifies the applications that should be presented that way.

Such applications must also provide default Newsstand icons as described in "Contents of the UINewsstandIcon Dictionary" (page 30).

This key is supported in iOS 5.0 and later.

UIPrerenderedIcon

`UIPrerenderedIcon` (`Boolean` - iOS) specifies whether the application'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 application icon.
`NO`	(Default) iOS applies a shine effect to the application icon.

UIRequiredDeviceCapabilities

`UIRequiredDeviceCapabilities` (`Array` or `Dictionary` - iOS) lets iTunes and the App Store know which device-related features an application requires in order to run. iTunes and the mobile App Store use this list to prevent customers from installing applications 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 application is able to run if the feature is present.

Table 4 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 application absolutely requires. If your application can accommodate missing features by avoiding the code paths that use those features, do not include the corresponding key.

Table 4 Dictionary keys for the `UIRequiredDeviceCapabilities` key

Key	Description
`telephony`	Include this key if your application requires (or specifically prohibits) the presence of the Phone application. You might require this feature if your application opens URLs with the `tel` scheme.
`wifi`	Include this key if your application requires (or specifically prohibits) access to the networking features of the device.
`sms`	Include this key if your application requires (or specifically prohibits) the presence of the Messages application. You might require this feature if your application opens URLs with the `sms` scheme.

Key	Description
`still-camera`	Include this key if your application requires (or specifically prohibits) the presence of a camera on the device. Applications use the `UIImagePickerController` interface to capture images from the device's still camera.
`auto-focus-camera`	Include this key if your application requires (or specifically prohibits) auto-focus capabilities in the device's still camera. Although most developers should not need to include this key, you might include it if your application supports macro photography or requires sharper images in order to do some sort of image processing.
`front-facing-camera`	Include this key if your application requires (or specifically prohibits) the presence of a forward-facing camera. Applications use the `UIImagePickerController` interface to capture video from the device's camera.
`camera-flash`	Include this key if your application requires (or specifically prohibits) the presence of a camera flash for taking pictures or shooting video. Applications use the `UIImagePickerController` interface to control the enabling of this feature.
`video-camera`	Include this key if your application requires (or specifically prohibits) the presence of a camera with video capabilities on the device. Applications use the `UIImagePickerController` interface to capture video from the device's camera.
`accelerometer`	Include this key if your application requires (or specifically prohibits) the presence of accelerometers on the device. Applications use the classes of the Core Motion framework to receive accelerometer events. You do not need to include this key if your application detects only device orientation changes.
`gyroscope`	Include this key if your application requires (or specifically prohibits) the presence of a gyroscope on the device. Applications use the Core Motion framework to retrieve information from gyroscope hardware.
`location-services`	Include this key if your application 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.)
`gps`	Include this key if your application requires (or specifically prohibits) the presence of GPS (or AGPS) hardware for greater accuracy when tracking locations. If you include this key, you should also include the `location-services` key. You should require GPS only if your application needs more accurate location data than the cell or Wi-fi radios might otherwise allow.
`magnetometer`	Include this key if your application requires (or specifically prohibits) the presence of magnetometer hardware. Applications use this hardware to receive heading-related events through the Core Location framework.
`gamekit`	Include this key if your application requires (or specifically prohibits) Game Center (iOS 4.1 and later.)
`microphone`	Include this key if your application uses the built-in microphone or supports accessories that provide a microphone.

Key	Description
`opengles-1`	Include this key if your application requires (or specifically prohibits) the presence of the OpenGL ES 1.1 interfaces.
`opengles-2`	Include this key if your application requires (or specifically prohibits) the presence of the OpenGL ES 2.0 interfaces.
`armv6`	Include this key if your application is compiled only for the armv6 instruction set. (iOS v3.1 and later.)
`armv7`	Include this key if your application is compiled only for the armv7 instruction set. (iOS v3.1 and later.)
`peer-peer`	Include this key if your application requires (or specifically prohibits) peer-to-peer connectivity over Bluetooth. (iOS v3.1 and later.)

This key is supported in iOS 3.0 and later.

UIRequiresPersistentWiFi

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

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

Note: If an iPad contains applications 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 application 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 application 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 application 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 application 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 application should choose; it uses it only to determine if your application 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 "Communicating with External Accessories" in *iOS Application Programming Guide*.

UISupportedInterfaceOrientations

`UISupportedInterfaceOrientations` (`Array` - iOS) specifies the interface orientations your application supports. The system uses this information (along with the current device orientation) to choose the initial orientation in which to launch your application. The value for this key is an array of strings. Table 5 lists the possible string values you can include in the array.

Table 5 Supported orientations

Value	Description
`UIInterfaceOrientationPortrait`	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.
`UIInterfaceOrientationPortrait-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.

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`	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) Do not inherit the opacity of the superlayer.

This key is supported in iOS 3.0 and later.

Document Revision History

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

Date	Notes
2011-10-12	Incorporates keys introduced in iOS 5.0.
2011-06-06	Added information about key support in Mac OS X 10.7.
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.
2010-03-23	Added keys specific to iOS 3.2.
	Removed the `LSHasLocalizedDisplayName` key, which was deprecated in Mac OS X 10.2.
2009-10-19	New document describing the keys you can use in a bundle's Info.plist file.