Cocoa Keys

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

Cocoa keys use the prefix `NS` to distinguish them from other keys. For information about developing Cocoa Touch apps for iOS, see App Programming Guide for iOS. For information about developing Cocoa apps for macOS, 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	Platforms
GCSupportedGameControllers	(none)	Specifies the types of game controllers allowed or required for your app. See GCSupportedGameControllers for details.	tvOS 9.0 and later, iOS 7.0 and later, OS X v10.9 and later
GCSupportsMultipleMicroGamepads	(none)	Specifies that the physical Apple TV Remote and the Apple TV Remote app should operate as separate game controllers. See GCSupportsMultipleMicroGamepads for details.	tvOS
GKGameCenterBadgingDisabled	(none)	Specifies whether your app is badged. See GKGameCenterBadgingDisabled for details.	iOS 7.0 and later
GKShowChallengeBanners	(none)	Specifies whether banners are shown within an app. See GKShowChallengeBannersfor details.	iOS 7.0 and later
NETestAppMapping	(none)	Enables testing of per-app VPN app extensions without using an MDM server. See NETestAppMapping for details.	iOS 9.0 and later, OS X v10.11 and later
NSAppleMusicUsageDescription	“Privacy - Media Library Usage Description”	Specifies the reason for your app to use the media library. See NSAppleMusicUsageDescription for details.	iOS
NSAppleScriptEnabled	“Scriptable”	Specifies whether AppleScript is enabled. See NSAppleScriptEnabled for details.	macOS
NSAppTransportSecurity	(none)	Specifies changes to the default strong security for HTTP connections in iOS and macOS apps and app extensions. See NSAppTransportSecurity for details.	iOS 9.0 and later, OS X v10.11 and later
NSBluetoothPeripheralUsageDescription	“Privacy - Bluetooth Peripheral	Specifies the reason for your app to use Bluetooth. See NSBluetoothPeripheralUsageDescription for	iOS 6.0 and later

	Usage Description"	details.	
NSCalendarsUsageDescription	"Privacy – Calendars Usage Description"	Specifies the reason for your app to access the user's calendars. See NSCalendarsUsageDescription for details.	iOS 6.0 and later
NSCameraUsageDescription	"Privacy – Camera Usage Description"	Specifies the reason for your app to access the device's camera. See NSCameraUsageDescription for details.	iOS 7.0 and later
NSContactsUsageDescription	"Privacy – Contacts Usage Description"	Specifies the reason for your app to access the user's contacts. See NSContactsUsageDescription for details.	iOS 6.0 and later, OS X v10.8 and later
NSDockTilePlugIn	"Dock Tile Plugin path"	Specifies the name of app's Dock tile plug-in, if present. See NSDockTilePlugIn for details.	macOS
NSHealthShareUsageDescription	"Privacy – Health Share Usage Description"	Specifies the reason for your app to read the user's health data. See NSHealthShareUsageDescription for details.	iOS 8.0 and later
NSHealthUpdateUsageDescription	"Privacy – Health Update Usage Description"	Specifies the reason for your app to make changes to the user's health data. See NSHealthUpdateUsageDescription for details.	iOS 8.0 and later
NSHomeKitUsageDescription	"Privacy – HomeKit Usage Description"	Specifies the reason for your app to access the user's HomeKit configuration data. See NSHomeKitUsageDescription for details.	iOS, watchOS
NSHumanReadableCopyright	"Copyright (human-readable)"	(Localizable) Specifies the copyright notice for the bundle. See NSHumanReadableCopyright for details. This key replaces the obsolete `CFBundleGetInfoString` key.	macOS
NSJavaNeeded	"Cocoa Java application"	Specifies whether the program requires a running Java VM. See NSJavaNeeded for details.	macOS
NSJavaPath	"Java classpaths"	An array of paths to classes whose components are preceded by `NSJavaRoot`. See NSJavaPath for details.	macOS
NSJavaRoot	"Java root directory"	The root directory containing the java classes. See NSJavaRoot for details.	macOS
NSLocationAlwaysUsageDescription	"Privacy – Location Always Usage Description"	Specifies the reason for your app to access the user's location information at all times. See NSLocationAlwaysUsageDescription for details.	iOS 8.0 and later, OS X v10.10 and later
NSLocationUsageDescription	"Privacy – Location Usage Description"	**Unused.** Use NSLocationWhenInUseUsageDescription or NSLocationAlwaysUsageDescription instead. See NSLocationUsageDescription for details.	iOS 6.0 and later, OS X v10.9 and later. **Unused** in iOS 8 and later.

NSLocationWhenInUseUsageDescription	"Privacy – Location When In Use Usage Description"	Specifies the reason for your app to access the user's location information while your app is in use. See NSLocationWhenInUseUsageDescription for details.	iOS 8.0 and later, OS X v10.10 and later
NSMainNibFile	"Main nib file base name"	The name of an app's main nib file. See NSMainNibFile for details.	iOS, macOS
NSMicrophoneUsageDescription	"Privacy – Microphone Usage Description"	Specifies the reason for your app to access any of the device's microphones. See NSMicrophoneUsageDescription for details.	iOS 7.0 and later
NSMotionUsageDescription	"Privacy – Motion Usage Description"	Specifies the reason for your app to access the device's accelerometer. See NSMotionUsageDescription for details.	iOS 7.0 and later
NSPersistentStoreTypeKey	"Core Data persistent store type"	The type of Core Data persistent store associated with a persistent document type. See NSPersistentStoreTypeKey for details.	macOS
NSPhotoLibraryUsageDescription	"Privacy – Photo Library Usage Description"	Specifies the reason for your app to access the user's photo library. See NSPhotoLibraryUsageDescription for details.	iOS 6.0 and later
NSPrefPaneIconFile	"Preference Pane icon file"	The name of an image file resource used to represent a preference pane in the System Preferences app. See NSPrefPaneIconFile for details.	macOS
NSPrefPaneIconLabel	"Preference Pane icon label"	The name of a preference pane displayed beneath the preference pane icon in the System Preferences app. See NSPrefPaneIconLabel for details.	macOS
NSPrincipalClass	"Principal class"	The name of the bundle's main class. See NSPrincipalClass for details.	macOS
NSRemindersUsageDescription	"Privacy – Reminders Usage Description"	Specifies the reason for your app to access the user's reminders. See NSRemindersUsageDescription for details.	iOS 6.0 and later
NSServices	"Services"	An array of dictionaries specifying the services provided by an app. See NSServices for details.	macOS
NSSiriUsageDescription	(none)	Specifies the reason for your app to send user data to Siri. See NSSiriUsageDescription for details.	iOS
NSSpeechRecognitionUsageDescription	(none)	Specifies the reason for your app to send user data to Apple's speech recognition servers. See NSSpeechRecognitionUsageDescription for details.	iOS
NSSupportsAutomaticTermination	(none)	Specifies whether the app may be killed to reclaim memory. See NSSupportsAutomaticTermination for details.	OS X v10.7 and later
NSSupportsPurgeableLocalStorage	(none)	Declares that the app can depend on nonlocal storage for user data. See NSSupportsPurgeableLocalStorage for details.	iOS 9.3 and later
NSSupportsSuddenTermination	(none)	Specifies whether the app may be killed to allow for faster shut down or log out	macOS

		operations. See NSSupportsSuddenTermination for details.	
NSUbiquitousContainer	(none)	Specifies the iCloud Drive settings for each container. See NSUbiquitousContainers for details.	iOS, macOS
NSUbiquitousContainerIsDocumentScopePublic	(none)	Specifies whether the iCloud Drive should share the contents of this container. See NSUbiquitousContainerIsDocumentScopePublic for details.	iOS, macOS
NSUbiquitousContainerName	(none)	Specifies the name that the iCloud Drive displays for your container. See NSUbiquitousContainerName for details.	iOS, macOS
NSUbiquitousContainerSupportedFolderLevels	(none)	Specifies the maximum number of folder levels inside your container's Documents directory. See NSUbiquitousContainerSupportedFolderLevels for details.	iOS, macOS
NSUbiquitousDisplaySet	(none)	Specifies the mobile document data that the app can view. See NSUbiquitousDisplaySet for details.	iOS, macOS
NSUserActivityTypes	(none)	Specifies the user activity types that the app supports. See NSUserActivityTypes for details.	iOS, macOS
NSUserNotificationAlertStyle	(none)	Specifies whether the notification style should be `banner`, `alert`, or `none`. The default value is `banner`, which is the recommended style. See NSUserNotificationAlertStyle for details.	macOS
NSVideoSubscriberAccountUsageDescription	"Privacy – TV Provider Usage Description"	Specifies the reason for your app to access the user's TV provider account. See NSVideoSubscriberAccountUsageDescription for details.	tvOS
UTExportedTypeDeclarations	"Exported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (and owned) by the app. See UTExportedTypeDeclarations for details.	iOS 5.0 and later, OS X v10.7 and later
UTImportedTypeDeclarations	"Imported Type UTIs"	An array of dictionaries specifying the UTI-based types supported (but not owned) by the app. See UTImportedTypeDeclarations for details.	iOS, macOS

GCSupportedGameControllers

`GCSupportedGameControllers` (`array (dictionary (string : string))` – tvOS, iOS, and macOS) Optional key, used only during the App Store submission process, that specifies the types of game controllers allowed or required for your app.

The value for this key is an array. Each array element is a dictionary whose key string is "`ProfileName`" and whose value string is one of the following:

`Gamepad` – Specifies the standard gamepad, supported in iOS 7.0 and later only, corresponding to a formfitting controller for an iOS device with a limited set of controls. Used with the `GCGamepad` class.
`ExtendedGamepad` – Specifies the extended gamepad, supported in tvOS 9.0 and later, iOS 7.0 and later, OS X 10.9 and later, and corresponding to either a formfitting controller for an iOS device or a standalone controller for iOS, macOS, or tvOS with an extended set of controls. Used with the `GCExtendedGamepad` class.
`MicroGamepad` – Specifies the micro gamepad, supported in tvOS 9.0 and later, and corresponding to a Siri Remote or the Apple TV Remote app running on a connected iOS device. Used with the `GCMicroGamepad` class.

Supported in tvOS 9.0 and later, iOS 7.0 and later, OS X v10.9 and later.

GCSupportsMultipleMicroGamepads

`GCSupportsMultipleMicroGamepads` (`Boolean` - tvOS). Specifies that the Apple TV Siri Remote and devices running the Apple TV Remote app should each operate as a discrete game controller. Default value is `NO`, indicating that input from all connected remotes is unified.

Specifically, in a Game Controller framework-based tvOS app that uses a value of `NO` for this key, all connected remotes are routed to a single `GCController` object in your app. When a user presses the A button on the Siri Remote, for example, the same in-app action is invoked as if they had pressed the A button on any connected Apple TV Remote app on a device.

If you instead specify a value of `YES` for this key, your tvOS app employs an independent `GCController` object for each connected remote.

Supported in tvOS 10.0 and later.

GKGameCenterBadgingDisabled

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

GKShowChallengeBanners

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

NETestAppMapping

`NETestAppMapping` (`Dictionary` - iOS, macOS) Use this key only during development and testing to help you create a per-app VPN app extension. By using this key, you can test per-app VPN communication without the use of a mobile device management (MDM) server. For more information, refer to NETunnelProviderManager Class Reference.

> **Important:** The `NETestAppMapping` key can be used only to create app rules in apps that are signed with a Development provisioning profile. In an app signed with a Distribution provisioning profile, this key has no effect. In addition, the App Store rejects any app with this key defined in its `Info.plist` file.

NSAppleMusicUsageDescription

`NSAppleMusicUsageDescription` (`String` - iOS). This key lets you describe the reason your app accesses the user's media library. When the system prompts the user to allow access, the value that you provide for this key is displayed as part of the alert.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the media library, must statically declare the intent to do so. Include the NSAppleMusicUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the media library without a corresponding purpose string, your app exits.

This key is supported in iOS 10 and later and in macOS 10.12 and later.

NSAppleScriptEnabled

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

NSAppTransportSecurity

`NSAppTransportSecurity` (`Dictionary` - iOS, macOS) Use this key to describe your app's intended HTTP connection behavior if you require exceptions from best security practices or you want to enable new security features.

On Apple platforms, a networking security feature called App Transport Security (ATS) is available to apps and app extensions, and is enabled by default. It improves privacy and data integrity by ensuring your app's network connections employ only industry-standard protocols and ciphers without known weaknesses. This helps instill user trust that your app does not accidentally leak transmitted data to malicious parties.

By configuring this key's value in your app's `Info.plist` file, you can customize the security of your network connections in a variety of ways. You can:

- Allow insecure communication with particular servers
- Allow insecure loads for web views or for media, while maintaining ATS protections elsewhere in your app
- Enable new security features such as Certificate Transparency

The `NSAppTransportSecurity` key is supported in iOS 9.0 and later and in OS X v10.11 and later, and is available in both apps and app extensions.

Starting in iOS 10.0 and later and in macOS 10.12 and later, the following subkeys are supported:

- `NSAllowsArbitraryLoadsForMedia`
- `NSAllowsArbitraryLoadsInWebContent`
- `NSRequiresCertificateTransparency`
- `NSAllowsLocalNetworking`

> **Note:** There are two "allows arbitrary loads" keys and they employ different naming patterns. Take care to use `…ForMedia` and `…InWebContent` correctly.

In this section:

ATS Configuration Basics

App Transport Security (ATS) is enabled by default for apps linked against the iOS 9.0 or OS X v10.11 SDKs or later, as indicated by the default Boolean value of `NO` for the `NSAllowsArbitraryLoads` key. This key is at the root level of the `NSAppTransportSecurity` dictionary.

With ATS enabled, HTTP connections must use HTTPS (RFC 2818). Attempts to connect using insecure HTTP fail. ATS employs the Transport Layer Security (TLS) protocol version 1.2 (RFC 5246). For background on secure Internet connections, read HTTPS Server Trust Evaluation.

The following listing represents the overall structure of the `NSAppTransportSecurity` dictionary, showing all possible keys, all of which are optional. Keep this structure in mind as you configure each element of the dictionary, as needed, for your app:

```
NSAppTransportSecurity : Dictionary {
    NSAllowsArbitraryLoads : Boolean
    NSAllowsArbitraryLoadsForMedia : Boolean
    NSAllowsArbitraryLoadsInWebContent : Boolean
```

```
    NSAllowsLocalNetworking : Boolean
    NSExceptionDomains : Dictionary {
        <domain-name-string> : Dictionary {
            NSIncludesSubdomains : Boolean
            NSExceptionAllowsInsecureHTTPLoads : Boolean
            NSExceptionMinimumTLSVersion : String
            NSExceptionRequiresForwardSecrecy : Boolean   // Default value is YES
            NSRequiresCertificateTransparency : Boolean
        }
    }
}
```

The `NSAppTransportSecurity` dictionary structure expresses two levels of configuration. At the primary level are keys to configure ATS protections for your app's network connections in general. Also at this level is the `NSExceptionDomains` key; this key lets you opt in to custom configuration for named domains, relative to ATS defaults, as needed.

The primary ATS keys are:

- `NSAllowsArbitraryLoads`

  If set to `YES`, disables all ATS restrictions for all network connections, apart from the connections to domains that you configure individually in the optional `NSExceptionDomains` dictionary. Default value is `NO`.

  > **Note:** Setting this key's value to `YES` triggers App Store review and requires justification.

- `NSAllowsArbitraryLoadsForMedia`

  If set to `YES`, disables all ATS restrictions for media that your app loads using the AV Foundation framework. Employ this key only for loading media that are already encrypted, such as files protected by FairPlay or by secure HLS, and that do not contain personalized information. Default value is `NO`.

  > **Note:** Setting this key's value to `YES` triggers App Store review and requires justification.

- `NSAllowsArbitraryLoadsInWebContent`

  If set to `YES`, disables all ATS restrictions for requests made from web views. This lets your app use an embedded browser that can display arbitrary content, without disabling ATS for the rest of your app. Default value is `NO`.

  > **Note:** Setting this key's value to `YES` triggers App Store review and requires justification.

- `NSAllowsLocalNetworking`

  If set to `YES`, allows loading of local resources without disabling ATS for the rest of your app. Default value is `NO`.

- `NSExceptionDomains`

  Optionally include this dictionary to configure ATS for one or more named domains.

  If you add this key to your `NSAppTransportSecurity` dictionary, any domains you then name within the dictionary obtain the default, full ATS protections—irrespective of the value you set for the global `NSAllowsArbitraryLoads` key. Subkeys of a domain-name key then let you alter that domain's ATS protections from its defaults.

Read important, detailed information on the preceding primary keys in Table 2.

At the secondary level are the subkeys within the optional ATS `NSExceptionDomains` dictionary. The step of including an `NSExceptionDomains` dictionary in your app's `Info.plist` file:

- Creates a container for one or more domain-specific dictionaries, letting you specify customized, per-domain HTTP connection properties
- Removes any general, app-wide ATS customizations you've specified using primary ATS keys

  For example, if you've added `NSAllowsArbitraryLoadsForMedia` key for your app in general, domains named in the exception domains dictionary do not allow arbitrary media loading.

Having thus started with default ATS protections for the named domains, you can optionally decrease or increase their protections individually. You can decrease a named domain's protections to:

- Allow insecure HTTP connections—without diminishing ATS protections for the HTTPS connections to a domain—by employing the `NSExceptionAllowsInsecureHTTPLoads` key with a value of `YES`; doing this triggers App Store review, as described in App Store Review for ATS
- Disable the perfect forward secrecy (PFS) requirement by employing the `NSExceptionRequiresForwardSecrecy` key with a value of `NO`
- Lower the minimum-allowed Transport Layer Security (TLS) version by employing the `NSExceptionMinimumTLSVersion` key

You can also increase a named domain's protections by requiring Certificate Transparency (see Certificate Transparency).

The elements of the optional `NSExceptionDomains` dictionary are:

- `<domain-name-string>`

  A domain name string, identifying a domain for which you want to specify a connection configuration. You can add multiple instances of this key, letting you name any number of domains in the one `NSExceptionDomains` dictionary. Configure each `<domain-name-string>` dictionary to contain one or more of the following child keys:

  - `NSIncludesSubdomains`

    If set to `YES`, applies a named domain's ATS configuration to all of its subdomains. Default value is `NO`.

  - `NSExceptionAllowsInsecureHTTPLoads`

    If set to `YES`, allows insecure HTTP loads for the named domain, but does not change Transport Layer Security (TLS) requirements and does not affect HTTPS loads for the named domain. Default value is `NO`.

    > **Note:** Setting this key's value to `YES` triggers App Store review and requires justification.

  - `NSExceptionMinimumTLSVersion`

    Specifies the minimum TLS version for network connections for the named domain, allowing connection using an older, less secure version of Transport Layer Security.

    > **Note:** Use of this key triggers App Store review and requires justification.

  - `NSExceptionRequiresForwardSecrecy`

    If set to `NO`, allows TLS ciphers, for the named domain, that do not support perfect forward secrecy (PFS). Default value is `YES`.

  - `NSRequiresCertificateTransparency`

    If set to `YES`, requires valid, signed Certificate Transparency timestamps for server certificates for the named domain. Default value is `NO`.

Read important, detailed information on the preceding `NSExceptionDomains` keys in Table 3.

Using ATS in Apple Frameworks

App Transport Security (ATS) is enforced by the `NSURLSession` class and all APIs that use it. ATS is automatically enabled when you link your app against the iOS 9.0 SDK or later or against the OS X v10.11 SDK or later. (The older `NSURLConnection` class also enforces ATS when you link against the iOS 9.0 SDK or later or against the OS X v10.11 SDK or later.) ATS protections are not available when using lower-level networking APIs provided by Apple, or when using third-party networking libraries.

> **Note:** Consider risks carefully before opting to use lower-level networking APIs provided by Apple, or opting to use third-party networking libraries. Such approaches lose App Transport Security protections, putting your app and your user's data at risk.

ATS is not available on operating systems older than iOS 9.0 or OS X v10.11; those older operating systems ignore the `NSAppTransportSecurity` key. When ATS is not available, the system still provides standard HTTPS security and performs server trust evaluation per RFC 2818.

If you link your app against an SDK for an operating system older than iOS 9.0 or OS X v10.11, your Internet connections continue to work but ATS is disabled, no matter which version of operating system your app is running on.

Availability of ATS for Remote and Local Connections

App Transport Security (ATS) applies only to connections made to public host names. The system does not provide ATS protection to connections made to:

- Internet protocol (IP) addresses
- Unqualified host names
- Local hosts employing the `.local` top-level domain (TLD)

To connect to an unqualified host name or to a `.local` domain, you must set the value of the `NSAllowsLocalNetworking` key to `YES`.

> **Note:** Although ATS is unenforced for connection to local hosts, Apple strongly recommends using Transport Layer Security (TLS) for any local connection, along with the use of a self-signed certificate to validate the local IP address.

Requirements for Connecting Using ATS

With App Transport Security (ATS) fully enabled, the system requires that your app's HTTP connections use HTTPS and that they satisfy the following security requirements:

- The X.509 digital server certificate must meet at least one of the following trust requirements:
  - Issued by a certificate authority (CA) whose root certificate is incorporated into the operating system
  - Issued by a trusted root CA and installed by the user or a system administrator
- The negotiated Transport Layer Security (TLS) version must be TLS 1.2. Attempts to connect without TLS/SSL protection, or with an older version of TLS/SSL, are denied by default.
- The connection must use either the AES-128 or AES-256 symmetric cipher. The negotiated TLS connection cipher suite must support perfect forward secrecy (PFS) through Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange, and must be one of the following:
  - `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
  - `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
  - `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
  - `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
  - `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`
  - `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
  - `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
  - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
  - `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
  - `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
  - `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
- The leaf server certificate must be signed with one of the following types of keys:
  - Rivest-Shamir-Adleman (RSA) key with a length of at least 2048 bits
  - Elliptic-Curve Cryptography (ECC) key with a size of at least 256 bits

  In addition, the leaf server certificate hashing algorithm must be Secure Hash Algorithm 2 (SHA-2) with a digest length, sometimes called a "fingerprint," of at least 256 (that is, SHA-256 or greater).

The requirements listed in this section are current as of this document's publication date, with stricter requirements possible in the future. Changes to these requirements will not break app binary compatibility.

Certificate Transparency

Certificate Transparency employs logging of X.509 certificates, using cryptographic assurance and in a manner that can be publicly audited. This system facilitates identifying certificates that were mistakenly or maliciously issued. App Transport Security lets you configure your app to require Certificate Transparency (CT) for specific, named domains. Before such a domain can connect with your app, it must prove to the system that its X.509 digital certificate is present in at least two CT logs trusted by Apple.

To require Certificate Transparency, set the value of the `NSRequiresCertificateTransparency` key, within the appropriate domain-name dictionary, to `YES`. (See the overall structure of the `NSAppTransportSecurity` dictionary, in ATS Configuration Basics, to see exactly where the `NSRequiresCertificateTransparency` key should be placed.)

Enabling Certificate Transparency does not eliminate the need for your app to revoke invalid certificates and to refuse connections that employ them. To support certificate checking and revocation, use Online Certificate Status Protocol (OCSP) stapling, specified in RFC6066.

For details on Certificate Transparency, see certificate-transparency.org.

ATS and HTTPS Server Trust Evaluation Requirements

Your ability to loosen HTTPS server trust evaluation requirements depends on whether or not App Transport Security (ATS) is enabled for a domain, as follows:

- If ATS is enabled for a domain, you cannot loosen the system's HTTPS server trust evaluation requirements.
- If ATS is not enabled for a domain, the system nonetheless performs HTTPS server trust evaluation, but you can loosen this requirement as described in HTTPS Server Trust Evaluation.

Whether or not ATS is enabled for a domain, you can tighten trust evaluation requirements, such as by implementing certificate pinning.

App Store Review for ATS

Your use of certain App Transport Security (ATS) keys triggers additional App Store review for your app, and requires you to provide justification. These keys are:

- `NSAllowsArbitraryLoads`
- `NSAllowsArbitraryLoadsForMedia`
- `NSAllowsArbitraryLoadsInWebContent`
- `NSExceptionAllowsInsecureHTTPLoads`
- `NSExceptionMinimumTLSVersion`

Some examples of justifications eligible for consideration are:

- Must connect to a server managed by another entity that does not support secure connections
- Must support connecting to devices that cannot be upgraded to use secure connections, and that must be accessed via public host names
- Must provide embedded web content from a variety of sources, but cannot use a class supported by the `NSAllowsArbitraryLoadsInWebContent` key
- App loads media content that is encrypted and that contains no personalized information

When submitting your app to the App Store, provide sufficient information for the App Store to determine why your app cannot make secure connections by default.

ATS Dictionary Details

Table 2 shows the primary keys within the `NSAppTransportSecurity` dictionary for describing your app's intended network behavior. For the sub-keys associated with the `NSExceptionDomains` dictionary, see Table 3.

Table 2 App Transport Security dictionary primary keys

Key	Xcode name	Type	Description
`NSAllowsArbitraryLoads`	"Allow Arbitrary Loads"	Boolean	An optional Boolean value that, when set to `YES`, disables App Transport Security (ATS) for all domains for which you do not explicitly reenable ATS by using an exception domain dictionary (as specified using the `NSExceptionDomains` key). Use of this key triggers App Store review and requires justification. Enable this key for cases where your app allows the user to specify connection to an arbitrary URL. Enabling this key can also be useful for debugging and development. In iOS 10 and later, and macOS 10.12 and later, the value of this key is ignored if any of the following keys are present in your app's `Info.plist` file: ▪ `NSAllowsArbitraryLoadsForMedia`

			▪ NSAllowsArbitraryLoadsInWebContent ▪ NSAllowsLocalNetworking **NOTE** Disabling ATS allows connection regardless of HTTP or HTTPS configuration, allows connection to servers with lower Transport Layer Security (TLS) versions, and allows connection using cipher suites that do not support perfect forward secrecy (PFS). This key's default value of NO results in default ATS behavior for all connections except those for which you have specified an exception domain dictionary (see Table 3).
NSAllowsArbitraryLoadsForMedia	(none)	Boolean	An optional Boolean value that, when set to YES, disables all App Transport Security restrictions for media loaded using APIs from the AV Foundation framework, as described in AV Foundation Framework Reference. Employ this key only for loading media that are already encrypted, such as files protected by FairPlay or by secure HLS, and that do not contain personalized information. If you add this key to your Info.plist file, then, irrespective of the value of the key, ATS ignores the value of the NSAllowsArbitraryLoads key. Default value is NO. Available starting in iOS 10.0 and macOS 10.12.
NSAllowsArbitraryLoadsInWebContent	(none)	Boolean	An optional Boolean value that applies only to content to be loaded into an instance of the following classes: ▪ WKWebView ▪ UIWebView (iOS only) ▪ WebView (macOS only) Set this key's value to YES to obtain exemption from ATS policies in your app's web views, without affecting the ATS-mandated security of your NSURLSession connections. Default value is NO. To support older versions of iOS and macOS, you can employ this key and still manually configure ATS. To do so, set this key's value to YES and also configure the NSAllowsArbitraryLoads subkeys. If you add this key to your Info.plist file, then, irrespective of the value of the key, ATS ignores the value of the NSAllowsArbitraryLoads key. Available starting in iOS 10.0 and macOS 10.12.
NSAllowsLocalNetworking	(none)	Boolean	An optional Boolean value that, when set to YES, removes App Transport Security protections for connections to unqualified domains and to .local domains, without disabling ATS for the rest of your app. If you set this key's value to YES, then App Transport Security ignores the value of the NSAllowsArbitraryLoads key in iOS 10 and later and in macOS 10.12 and later. This behavior supports adoption of App Transport Security protections while allowing embedded browsers to continue working in iOS 9 and earlier and in OS X v10.11 and earlier. (To

			obtain this behavior, set the value of this key to YES and set the value of the NSAllowsArbitraryLoads key to YES as well.) Default value is NO. Available starting in iOS 10.0 and macOS 10.12.
NSExceptionDomains	"Exception Domains"	Dictionary	An optional dictionary of ATS exceptions for specific domains. Each value in the dictionary is itself a dictionary, and describes a domain-specific network connection configuration exception. An exception domain's top-level key is the domain name string for which you want to specify a connection configuration; for example, www.apple.com. A domain name key for an exception dictionary: ■ Must be lowercased to work correctly ■ Must not include a port number ■ Must not be a numerical IP address (but rather a string) ■ Must not end with a trailing dot, unless you only want to match a domain string with a trailing dot. For example, example.com. (with a trailing dot) matches "example.com." but not "example.com". Similarly, example.com matches "example.com" but not "example.com.". For details on configuring an exception domain dictionary, see Table 3.

Table 3 shows the keys for describing server-specific exceptions to your app's overall intended network behavior.

Table 3 Exception domains dictionary keys

Key	Xcode name	Type	Description
NSIncludesSubdomains	(none)	Boolean	An optional Boolean value that, when set to YES, applies the NSExceptionDomains ATS exceptions to all subdomains (of the domain whose name is the top-level key in the NSExceptionDomains dictionary). Default value is NO.
NSRequiresCertificateTransparency	(none)	Boolean	An optional Boolean value that, when set to YES, requires that valid, signed Certificate Transparency (CT) timestamps, from known CT logs, be presented for server (X.509) certificates on a domain. Default value is NO. Available starting in iOS 10.0 and macOS 10.12.
NSExceptionAllowsInsecureHTTPLoads	(none)	Boolean	An optional Boolean value that, when set to YES, allows insecure HTTP loads but does not change Transport Layer Security (TLS) requirements. Use this key to describe your app's intended connection behavior for a domain whose security attributes you have control over. Use of this key triggers App Store review and requires justification. With this key's value set to YES, your app can make secure connections to a secure server but can also connect insecurely to a server with no certificate, or a self-signed, expired, or host-name-mismatched

			certificate. Set this key's value to `YES`, if needed, to: ■ Enable connection to an insecure HTTP server ■ Enable connection to an untrusted HTTPS server ■ Enable connection to an HTTPS server for which you want to perform your own server trust evaluation In some cases you need to use other exception-dictionary keys along with this one to establish connection. For example, to connect to an HTTPS server that uses a self-signed certificate and a TLS version lower than 1.2, set the `NSExceptionAllowsInsecureHTTPLoads` value to `YES` and also set an appropriate value for the `NSExceptionMinimumTLSVersion` key. Default value is `NO`.
`NSExceptionRequiresForwardSecrecy`	(none)	Boolean	An optional Boolean value for overriding the requirement that a server support perfect forward secrecy (PFS). Use this key to describe your app's intended connection behavior for a domain whose security attributes you have control over. Default value is `YES`, which limits the accepted ciphers to those listed in ATS Configuration Basics. Setting the value to `NO` results in the following ciphers, which do not support FS, also being accepted: ■ `TLS_RSA_WITH_AES_256_GCM_SHA384` ■ `TLS_RSA_WITH_AES_128_GCM_SHA256` ■ `TLS_RSA_WITH_AES_256_CBC_SHA256` ■ `TLS_RSA_WITH_AES_256_CBC_SHA` ■ `TLS_RSA_WITH_AES_128_CBC_SHA256` ■ `TLS_RSA_WITH_AES_128_CBC_SHA`
`NSExceptionMinimumTLSVersion`	(none)	String	An optional string value that specifies the minimum Transport Layer Security (TLS) version for network connections. Use this key to describe your app's intended connection behavior for a domain whose security attributes you have control over. Use of this key triggers App Store review and requires justification. Valid values are: ■ `TLSv1.0` ■ `TLSv1.1` ■ `TLSv1.2` Default value is `TLSv1.2`.

ATS Examples

This section shows how to specify some common networking behaviors using the `NSAppTransportSecurity` key.

Allowing Insecure Connection to a Single Server

To use ATS generally but allow connection to a specific server that does not support the HTTPS protocol—for example, a media server that your app uses—employ the following configuration pattern in your `Info.plist` file:

```
NSAppTransportSecurity
    NSExceptionDomains
```

```
        "media-server.example.com"
            NSExceptionAllowsInsecureHTTPLoads = YES
```

Important: Before implementing this exception, consider that a seemingly-benign network request can cause security problems of the sort that ATS is intended to mitigate. For example, fetching media from an insecure server entails the following risks, among others:

- An attacker can see the media file a user is accessing
- Your app's attack surface expands, for example, by allowing a bad actor to feed your app a malicious file intended to trigger a buffer overrun

Avoid this connection type if possible.

Allowing Lowered Security to a Single Server

To use a less-secure HTTPS connection to a specified server that uses an older version of TLS and that does not support perfect forward secrecy (PFS), while retaining the default, best-practice ATS behavior elsewhere, employ the following configuration pattern in your `Info.plist` file:

```
NSAppTransportSecurity
    NSExceptionDomains
        "less-secure.example.com"
            NSExceptionRequiresForwardSecrecy = NO
            NSExceptionMinimumTLSVersion = "TLSv1.0"
```

Using ATS For Your Servers and Allowing Insecure Connections Elsewhere

If your app is a web browser, or otherwise allows a user to enter an arbitrary URL, your app must be able to load resources from anywhere. In such a scenario, your app should still use ATS when communicating with servers whose security attributes you control, such as your update server.

To require ATS connections to domains that you control, while allowing insecure HTTP access to all other URLs, employ the following configuration pattern in your `Info.plist` file:

```
NSAppTransportSecurity
    NSExceptionDomains
        "domain-i-control.example.com"
            NSExceptionAllowsInsecureHTTPLoads = NO
            NSExceptionRequiresForwardSecrecy = YES
            NSExceptionMinimumTLSVersion = "TLSv1.2"
        "other-domain-i-control.example.com"
            NSExceptionAllowsInsecureHTTPLoads = NO
            NSExceptionRequiresForwardSecrecy = YES
            NSExceptionMinimumTLSVersion = "TLSv1.2"
    NSAllowsArbitraryLoads = YES
```

Debugging ATS Connections

If you are seeing Internet connection problems that you suspect are related to ATS, try the following troubleshooting approach:

1. Disable ATS entirely to confirm that ATS is involved with the connection problem. Do this by using the following configuration pattern:

   ```
   NSAppTransportSecurity
       NSAllowsArbitraryLoads = YES
   ```

   If entirely disabling ATS solves your connection problem, proceed with step 2. (If you still cannot connect with ATS disabled, the problem lies somewhere other than with ATS.)

2. Test whether a specific domain under your control is causing the connection problem. Do this by reenabling ATS except on that specific domain, using the following configuration pattern:

```
NSAppTransportSecurity
    NSAllowsArbitraryLoads = NO  // Shown for clarity; this is the default
    NSExceptionDomains
        "secure-server-i-control.example.com"
            NSExceptionAllowsInsecureHTTPLoads = YES
            NSExceptionRequiresForwardSecrecy = NO
            NSExceptionMinimumTLSVersion = "TLSv1.0"
```

   If connection continues to work with this configuration, it is likely that the problem lies with the specific domain named in your `NSExceptionDomains` dictionary.

   If connection fails with this configuration, the problem is likely one of two things:

   - An ATS misconfiguration on the server named in the `NSExceptionDomains` dictionary
   - An ATS misconfiguration on another server that the connection is redirected to

   Once you have pinpointed the server at blame for the connection problem, proceed with step 3.

3. Use the TLSTool sample code project to investigate the details of the TLS version, server certificate, and cipher suite the problematic server is using.

   If the new information you've gathered allows you to reconfigure the server to support ATS, do so.

   If you are unable to reconfigure the server, use the information you've gathered using the TLSTool app to define an appropriate `NSExceptionDomains` dictionary for that server.

In addition, you can enable logging of `NSURLSession` class errors by employing the following environment variable in your Xcode project:

```
CFNETWORK_DIAGNOSTICS=1
```

For more information about using this environment variable, read CFNetwork Diagnostic Logging. For help interpreting error codes, read Security Framework Error Codes.

Using the nscurl Tool to Diagnose ATS Connection Issues

In OS X v10.11 and later, you can use the `/usr/bin/nscurl` tool to help diagnose connection issues due to App Transport Security.

The `--ats-diagnostics` option tries to connect with the specified URL using different combinations of values for the `NSAllowsArbitraryLoads`, `NSExceptionMinimumTLSVersion`, `NSExceptionRequiresForwardSecrecy`, and `NSExceptionAllowsInsecureHTTPLoads` keys shown in Table 3. A summary of the results is printed to the command line.

The format for the command is:

```
/usr/bin/nscurl --ats-diagnostics [--verbose] URL
```

URL. The URL for the host. This is required.
verbose. Specifying this option includes more information for each connection attempt including the keys and associated values used.

Listing 1 shows partial output of diagnosing a connection to `https://apple.com`.

Listing 1 Partial output of `nscurl`

```
> /usr/bin/nscurl --ats-diagnostics https://apple.com

Starting ATS Diagnostics


Configuring ATS Info.plist keys and displaying the result of HTTPS loads to https://apple.com.

A test will "PASS" if URLSession:task:didCompleteWithError: returns a nil error.

Use '--verbose' to view the ATS dictionaries used and to display the error received in
URLSession:task:didCompleteWithError:.

================================================================================
```

```
Default ATS Secure Connection

---

ATS Default Connection

2015-09-09 09:53:01.592 nscurl[9207:5187047] CFNetwork SSLHandshake failed (-9824)

2015-09-09 09:53:01.593 nscurl[9207:5187047] NSURLSession/NSURLConnection HTTP load failed
(kCFStreamErrorDomainSSL, -9824)

Result : FAIL

---


================================================================================


Allowing Arbitrary Loads


---

Allow All Loads

Result : PASS

---


================================================================================


Configuring TLS exceptions for apple.com


---

TLSv1.2

2015-09-09 09:53:02.145 nscurl[9207:5187047] CFNetwork SSLHandshake failed (-9824)

2015-09-09 09:53:02.146 nscurl[9207:5187047] NSURLSession/NSURLConnection HTTP load failed
(kCFStreamErrorDomainSSL, -9824)

Result : FAIL

---


---

TLSv1.1

2015-09-09 09:53:02.270 nscurl[9207:5187047] CFNetwork SSLHandshake failed (-9824)

2015-09-09 09:53:02.271 nscurl[9207:5187047] NSURLSession/NSURLConnection HTTP load failed
(kCFStreamErrorDomainSSL, -9824)

Result : FAIL

---


---

TLSv1.0

2015-09-09 09:53:02.407 nscurl[9207:5187047] CFNetwork SSLHandshake failed (-9824)

2015-09-09 09:53:02.408 nscurl[9207:5187047] NSURLSession/NSURLConnection HTTP load failed
(kCFStreamErrorDomainSSL, -9824)

Result : FAIL

---


================================================================================


Configuring PFS exceptions for apple.com

…
```

Listing 2 shows partial output when using the `--verbose` option. The two main differences are showing the values of the keys from `Info.plist` from lines 10 to 17, and a longer error result shown on line 19.

Listing 2 Partial output of nscurl using --verbose

```
> /usr/bin/nscurl --ats-diagnostics --verbose https://apple.com

Starting ATS Diagnostics

…


Configuring PFS exceptions and allowing insecure HTTP for apple.com


---

Disabling Perfect Forward Secrecy and Allowing Insecure HTTP

ATS Dictionary:

{

    NSExceptionDomains =     {

        "apple.com" =         {

            NSExceptionAllowsInsecureHTTPLoads = YES;

            NSExceptionRequiresForwardSecrecy = NO;

        };

    };

}

Result : FAIL

Error : Error Domain=NSURLErrorDomain Code=-1022 "The resource could not be loaded because the App
Transport Security policy requires the use of a secure connection." UserInfo=
{NSUnderlyingError=0x7fc6a9d11900 {Error Domain=kCFErrorDomainCFNetwork Code=-1022 "(null)"},
NSErrorFailingURLStringKey=http://www.apple.com/apple-events/september-2015/,
NSErrorFailingURLKey=http://www.apple.com/apple-events/september-2015/, NSLocalizedDescription=The
resource could not be loaded because the App Transport Security policy requires the use of a secure
connection.}

---

…
```

NSBluetoothPeripheralUsageDescription

`NSBluetoothPeripheralUsageDescription` (`String` - iOS) This key lets you describe the reason your app uses Bluetooth. When the system prompts the user to allow usage, the value that you provide for this key is displayed as part of the alert.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the Bluetooth interface, must statically declare the intent to do so. Include the NSBluetoothPeripheralUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the Bluetooth interface without a corresponding purpose string, your app exits.

This key is supported in iOS 6.0 and later.

NSCalendarsUsageDescription

`NSCalendarsUsageDescription` (`String` - iOS) This key lets you describe the reason your app accesses the user's calendars. When the system prompts the user to allow access, this string is displayed as part of the alert.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's calendars, must statically declare the intent to do so. Include the NSCalendarsUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's calendars without a corresponding purpose string, your app exits.

This key is supported in iOS 6.0 and later.

NSCameraUsageDescription

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

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the device's camera, must statically declare the intent to do so. Include the NSCameraUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the device's camera without a corresponding purpose string, your app exits.

This key is supported in iOS 7.0 and later.

NSContactsUsageDescription

`NSContactsUsageDescription` (`String` - iOS) The key lets you describe the reason your app accesses the user's contacts. When the system prompts the user to allow access, this string is displayed as part of the alert.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's contacts, must statically declare the intent to do so. Include the NSContactsUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's contacts without a corresponding purpose string, your app exits.

This key is supported in iOS 6.0 and later.

NSDockTilePlugIn

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

NSHealthShareUsageDescription

NSHealthShareUsageDescription (`String` - iOS). This key lets you describe the reason your app reads the user's health data. The system prompts the user to allow access when you call the `requestAuthorizationToShareTypes:readTypes:completion:` method, and this string is displayed as part of the alert. For more information, read HKHealthStore Class Reference and Setting Up HealthKit. This string is localizable.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which reads the user's health data, must statically declare the intent to do so. Include the NSHealthShareUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to read the user's health data without a corresponding purpose string, your app exits.

This key is supported in iOS 8.0 and later.

NSHealthUpdateUsageDescription

NSHealthUpdateUsageDescription (`String` - iOS). This key lets you describe the reason your app makes changes to the user's health data. The system prompts the user to allow access when you call the `requestAuthorizationToShareTypes:readTypes:completion:` method, and this string is displayed as part of the alert. For more information, read HKHealthStore Class Reference and Setting Up HealthKit. This string is localizable.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which reads the user's health data, must statically declare the intent to do so. Include the NSHealthUpdateUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to change the user's health data without a corresponding

purpose string, your app exits.

This key is supported in iOS 8.0 and later.

NSHomeKitUsageDescription

NSHomeKitUsageDescription (`String` – iOS, watchOS). This key lets you describe the reason your app access the user's HomeKit configuration data. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses user's HomeKit configuration data, must statically declare the intent to do so. Include the NSHomeKitUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to change the user's HomeKit configuration data without a corresponding purpose string, your app exits.

NSHumanReadableCopyright

`NSHumanReadableCopyright` (`String` – macOS). This key contains a string with the copyright notice for the bundle; for example, `© 2016, My Company`. You can load this string and display it in an About dialog box. The system uses this string in the app's Info window in Finder.

This key can be localized by including it in your `InfoPlist.strings` files.

This key replaces the obsolete `CFBundleGetInfoString` key.

See also CFBundleShortVersionString.

NSJavaNeeded

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

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

Deprecated in OS X v10.5.

NSJavaPath

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

Deprecated in OS X v10.5.

NSJavaRoot

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

NSLocationAlwaysUsageDescription

`NSLocationAlwaysUsageDescription` (`String` – iOS) This key lets you describe the reason your app accesses the user's location information at all times. Include this key when your app uses location services in a potentially nonobvious way while running in the foreground or the background. For example, a social app might include this key when it uses location

information to track the user's location and display other users that are nearby. In this case, the fact that the app is tracking the user's location might not be readily apparent. The system includes the value of this key in the alert panel displayed to the user when requesting permission to use location services.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's location information, must statically declare the intent to do so. Include the NSLocationAlwaysUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's location information without a corresponding purpose string, your app exits.

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

This key is supported in iOS 8.0 and later.

NSLocationUsageDescription

`NSLocationUsageDescription` (`String` - iOS) **Unused** in iOS 8 and later. If you link your app on or after iOS 8, use the NSLocationAlwaysUsageDescription or NSLocationWhenInUseUsageDescription key instead.

This key lets you describe the reason your app accesses the user's location information. When the system prompts the user to allow access, this string is displayed as part of the alert panel.

This key is supported in iOS 6.0 through iOS 7. This key is ignored in iOS 8 and later.

NSLocationWhenInUseUsageDescription

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

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's location information, must statically declare the intent to do so. Include the NSLocationWhenInUseUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's location information without a corresponding purpose string, your app exits.

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

This key is supported in iOS 8.0 and later.

NSMainNibFile

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

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

NSMicrophoneUsageDescription

`NSMicrophoneUsageDescription` (`String` - iOS) This key lets you describe the reason your app (including an iMessage app) accesses any of the device's microphones. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses any of the device's microphones, must statically declare the intent to do so. Include the NSMicrophoneUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access any of the device's microphones without a corresponding purpose string, your app exits.

This key is supported in iOS 7.0 and later.

NSMotionUsageDescription

`NSMotionUsageDescription` (`String` – iOS) This key lets you describe the reason your app accesses the device's accelerometer. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the device's accelerometer, must statically declare the intent to do so. Include the NSMotionUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the device's accelerometer without a corresponding purpose string, your app exits.

This key is supported in iOS 7.0 and later.

NSPersistentStoreTypeKey

`NSPersistentStoreTypeKey` (`String` – macOS). This key contains a string that specifies the type of Core Data persistent store associated with a document type (see CFBundleDocumentTypes).

NSPhotoLibraryUsageDescription

`NSPhotoLibraryUsageDescription` (`String` – iOS) This key lets you describe the reason your app accesses the user's photo library. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's photo library, must statically declare the intent to do so. Include the NSPhotoLibraryUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's photo library without a corresponding purpose string, your app exits.

This key is supported in iOS 6.0 and later.

NSPrefPaneIconFile

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

NSPrefPaneIconLabel

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

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

NSPrincipalClass

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

NSRemindersUsageDescription

`NSRemindersUsageDescription` (`String` - iOS) This key lets you describe the reason your app accesses the user's reminders. When the system prompts the user to allow access, this string is displayed as part of the alert.

> **Important:** To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses the user's reminders, must statically declare the intent to do so. Include the NSRemindersUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's reminders without a corresponding purpose string, your app exits.

This key is supported in iOS 6.0 and later.

NSServices

`NSServices` (`Array` - macOS). This key contains an array of dictionaries specifying the services provided by the app. Table 4 lists the keys for specifying a service:

Table 4 Keys for NSServices dictionaries

Key	Xcode name	Type	Description	Platforms
NSPortName	"Incoming service port name"	`String`	This key specifies the name of the port your app monitors for incoming service requests. Its value depends on how the service provider app is registered. In most cases, this is the app name. For more information, see Services Implementation Guide.	macOS
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)`.	macOS
NSSendFileTypes	(none)	`Array`	This key specifies an array of strings. Each string should contain a UTI defining a supported file type. Only UTI types are allowed; pasteboard types are not permitted. To specify pasteboard types, continue to use the `NSSendTypes` key. By assigning a value to this key, your service declares that it can operate on files whose type conforms to one or more of the given file types. Your service will receive a pasteboard from which you can read file URLs. Available in OS X v10.6 and later. For information on UTIs, see Uniform Type Identifiers Overview.	macOS
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.	macOS

			In OS X v10.5 and earlier, this key is required. In OS X v10.6 and later, you should use the `NSSendFileTypes` key instead.	
NSServiceDescription	(none)	`String`	This key specifies a description of your service that is suitable for presentation to users. This description string may be long to give users adequate information about your service. To localize the menu item text, create a `ServicesMenu.strings` file for each localization in your bundle. This strings file should contain this key along with the translated description string as its value. For more information about creating strings files, see Resource Programming Guide. Available in OS X v10.6 and later.	macOS
NSRequiredContext	(none)	`Dictionary` or `Array`	This key specifies a dictionary with the conditions under which your service is made available to the user. Alternatively, you can specify an array of dictionaries, each of which contains a set of conditions for enabling your service. See the discussion after this table for information about specifying the value of this key. Available in OS X v10.6 and later.	macOS
NSRestricted	(none)	`Boolean`	Specifying a value of `YES` for this key prevents the service from being invoked by a sandboxed app. You should set the value to `YES` if your service performs privileged or potentially dangerous operations that would allow a sandboxed app to escape its containment. For example, you should set it to `YES` if your service executes arbitrary files or text strings as scripts, reads or writes any file specified by a path, or retrieves the contents of an arbitrary URL from the network on behalf of the client of the service. The default value for this key is `false`. Available in OS X v10.7 and later.	macOS
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.	macOS
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 OS X v10.5 and earlier, menu items must be unique. You can ensure a unique name by combining the app name with the command name and separating them with a slash character "/". This effectively creates a submenu for your services. For example, `Mail/Send` would appear in the Services menu as a menu named Mail with an item named Send. Submenus are not supported (or necessary) in OS X v10.6 and later. If you specify a slash character in OS X v10.6 and later, the slash and any text preceding it are discarded. Instead, services with the same name are disambiguated by adding the app name in parenthesis after the menu item text. To localize the menu item text, create a `ServicesMenu.strings` file for each	macOS

			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.	
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.	macOS
NSUserData	"User Data"	`String`	This key is an optional string that contains a value of your choice.	macOS
NSTimeout	"Timeout value (in milliseconds)"	`String`	This key is an optional numerical string that indicates the number of milliseconds Services should wait for a response from the app providing a service when a response is required.	macOS

In OS X v10.6 and later, the `NSRequiredContext` key may contain a dictionary or an array of dictionaries describing the conditions under which the service appears in the Services menu. If you specify a single dictionary, all of the conditions in that dictionary must be met for the service to appear. If you specify an array of dictionaries, all of the conditions in only one of those dictionaries must be met for the service to appear. Each dictionary may contain one or more of the keys listed in Table 5. All keys in the dictionary are optional.

Table 5 Contents of the `NSRequiredContext` dictionary

Key	Xcode name	Type	Description	Platform
NSApplicationIdentifier	(none)	`String` or `Array`	The value of this key is a string or an array of strings, each of which contains the bundle ID (`CFBundleIdentifier` key) of an app. Your service appears only if the bundle ID of the current app matches one of the specified values.	macOS
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.	macOS
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`.	macOS
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.	macOS
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	macOS

			type. For example, if the selected text contained an HTTP-based link, the service would be displayed if the value of this key were set to `URL`. Note that all of the selected text is provided to the service-vending app, not just the parts found to contain the given data types.	

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

NSSiriUsageDescription

NSSiriUsageDescription (`String` - iOS) This key lets you describe the reason your app sends user data to Siri. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which accesses Siri, must statically declare the intent to do so. Include the NSSiriUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access Siri without a corresponding purpose string, your app exits.

NSSpeechRecognitionUsageDescription

NSSpeechRecognitionUsageDescription (`String` - iOS) This key lets you describe the reason your app sends user data to Apple's speech recognition servers. When the system prompts the user to allow access, this string is displayed as part of the alert.

Important: To protect user privacy, an iOS app linked on or after iOS 10.0, and which sends user data to Apple's speech recognition servers, must statically declare the intent to do so. Include the NSSpeechRecognitionUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to use Apple's speech recognition service without a corresponding purpose string, your app exits.

NSSupportsAutomaticTermination

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

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

NSSupportsPurgeableLocalStorage

`NSSupportsPurgeableLocalStorage` (`Boolean` - iOS). This key contains a Boolean value that indicates whether the app is designed to work, without disruption to the user, with the local data container treated as a volatile cache by the system. The default value of this key is `NO`. If your app supports Shared iPad (a feature of iOS device management), set this key's value to `YES`.

When set to `YES`, the system is enabled to purge local storage, at the system's discretion, when the user is logged out.

Supporting Shared iPad entails different work depending on where your app chooses to store nonlocal user data, as follows:

- User data stored in iCloud is automatically restored by the system, as needed.
- Nonlocal user data that is not stored in iCloud must be restored explicitly, by your app, from the app's non-iCloud

service.

If your app uses only local storage and does not depend on its persistence (for example, a simple calculator app), you can declare support for Shared iPad by setting this key's value to `YES`.

NSSupportsSuddenTermination

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

NSUbiquitousContainers

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

NSUbiquitousContainerIsDocumentScopePublic

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

NSUbiquitousContainerName

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

NSUbiquitousContainerSupportedFolderLevels

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

- `None`

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

- `One`

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

- `Any`

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

NSUbiquitousDisplaySet

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

account) the app retrieves its data files.

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

NSUserActivityTypes

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

NSUserNotificationAlertStyle

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

NSVideoSubscriberAccountUsageDescription

NSVideoSubscriberAccountUsageDescription (`String` - tvOS). This key lets you describe the reason your app access the user's TV provider account. When the system prompts the user to allow access, this string is displayed as part of the alert.

> **Important:** To protect user privacy, a tvOS app linked on or after tvOS 10.0, and which accesses user's TV provider account, must statically declare the intent to do so. Include the NSVideoSubscriberAccountUsageDescription key in your app's `Info.plist` file and provide a purpose string for this key. If your app attempts to access the user's TV provider account without a corresponding purpose string, your app exits.

UTExportedTypeDeclarations

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

The value for the `UTExportedTypeDeclarations` key is an array of dictionaries. Each dictionary contains a set of key-value pairs identifying the attributes of the type declaration. Table 6 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 key.

Table 6 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, macOS
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, macOS

UTTypeIconFile	"Icon file name"	`String`	The name of the bundle icon resource to associate with this UTI. You should include this key only for types that your app exports. This file should have a `.icns` filename extension. You can create this file using the Icon Composer app that comes with Xcode Tools.	macOS
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, macOS
UTTypeReferenceURL	"Reference URL"	`String`	The URL for a reference document that describes this type.	macOS
UTTypeSize64IconFile	(none)	`String`	The name of the 64 x 64 pixel icon resource file (located in the app's bundle) to associate with this UTI. You should include this key only for types that your app exports.	iOS
UTTypeSize320IconFile	(none)	`String`	The name of the 320 x 320 pixel icon resource file (located in the app's bundle) to associate with this UTI. You should include this key only for types that your app exports.	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, macOS

The way you specify icon files in macOS 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 macOS, 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 in OS X v10.5 and later. For more information about UTIs and their use, see Uniform Type Identifiers Overview.

UTImportedTypeDeclarations

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

The value for this key is an array of dictionaries and uses the same keys as those for the UTExportedTypeDeclarations key. For a list of these keys, see Table 6.

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