MobileIron AppConnect 4.6.0 for iOS App Wrapping Developers Guide

AppConnect for iOS Wrapper Library and Wrapping Tool

August 5, 2020

For complete product documentation see:
MobileIron AppConnect for iOS Product Documentation Home Page

Contents

Introducing AppConnect for iOS wrapped apps

MobileIron AppConnect secures and manages enterprise apps on mobile devices. These secure enterprise apps are called *AppConnect apps* or *secure apps.*

You can create an AppConnect app for iOS two ways:

- Wrapping the app
  The MobileIron AppConnect wrapping technology creates a secure app without any further app development.
- Using the AppConnect for iOS SDK (software development kit)
  An app developer uses the SDK to create a secure app, or turn an existing app into a secure app.

For information about choosing between wrapping and the SDK, see *Choosing Wrapping or SDK Development to Create AppConnect for iOS Apps*.

Note The Following:

- You **cannot** wrap an app if you got the app (IPA file) from the Apple App Store.
- Wrapped apps are **not** compliant with Apple's terms and conditions, and **cannot** be distributed using the Apple App Store. The app must be distributed using the MobileIron server's enterprise app storefront.
- If your app is wrapped with an older version of the AppConnect for iOS Wrapper Library, MobileIron recommends that you always rewrap the app with the current version. Using the current version ensures the app contains all new features, improvements, and resolved issues.
- Legal notices are on https://support.mobileiron.com/copyrights/ACe.
- An Apple Developer Enterprise Program account is required to distribute in-house apps. See Apple Developer Enterprise Program.

Related topics

For information about AppConnect for iOS from the perspective of a MobileIron server administrator:

- MobileIron Core or Connected Cloud: The *MobileIron Core AppConnect and AppTunnel Guide*
- MobileIron Cloud: The *MobileIron Cloud Administrator Guide*

AppConnect for iOS wrapped app overview

Wrapped app features

Secure enterprise apps that are created using the AppConnect wrapping technology can:

- Tunnel network connections to servers behind an enterprise's firewall.
  This capability means that device users do not have to separately set up VPN access on their devices to use the app.
- Authenticate an app user to an enterprise service.
  This capability means that AppConnect app users do not have to enter login credentials to access enterprise resources.
- Enforce data loss prevention.
  The MobileIron server administrator decides whether an app can:
  - copy to the iOS pasteboard
  - use drag and drop
  - use the document interaction feature (Open In and Open From)
  - use print capabilities
  - use dictation with the native iOS keyboard

  AppConnect for iOS uses these server settings to limit the app's functionality to prevent data loss through these features.
- Control custom keyboard use by your app
  The MobileIron server administrator can choose whether an app can use custom keyboards, and the AppConnect library enforces the choice.. If the administrator does not configure this choice, your app can choose to reject custom keyboard use.
- Block dictation from the native iOS keyboard
  By default, the AppConnect wrapping technology blocks using dictation from the native iOS keyboard. The MobileIron server administrator can override this behavior by adding a key-value pair to the app's configuration. The key is called MI_AC_WR_ALLOW_KEYBOARD_DICTATION. By default, the value is false. If the administrator sets the value to true, then wrapped AppConnect apps can use dictation with the native keyboard.
- Receive app-specific configuration information from the MobileIron server.
  This capability requires some additional app development. It means that device users do not have to manually enter configuration details that the app requires. By automating this process for the device users, each user has a better experience when installing and setting up apps. Also, the enterprise has fewer support calls.
- Protect AppConnect-related data on the device, such as configuration and certificates, using encryption.
  If an app enables iOS data protection on its files, and the device has a device passcode, then the app's data is also encrypted.
- Blur the app's screens when the app is not in the foreground.
  The AppConnect wrapping technology enforces this behavior.

App requirements

- You can wrap any iOS app (IPA file) **as long as you did not get the IPA file from the Apple App Store**. The app can have been built as a 64-bit app or as a 32-bit app.
- Wrapped apps are **not** compliant with Apple's terms and conditions, and **cannot** be distributed using the Apple App Store. The app must be distributed using the MobileIron server's enterprise app storefront.

NOTE: You can wrap an app only if it supports fast app switching, an iOS feature added in iOS 4.0. Fast app switching means that the app can go into the background for a short time without iOS terminating it. AppConnect for iOS requires that apps support this feature. Most apps support fast app switching. To ensure that an app supports fast app switching, a developer can remove the `UIApplicationExitsOnSuspend` key if it is present in the app's Info.plist.

Supported programming languages

You can wrap apps written in either Objective-C or Swift.

Related topics

AppConnect wrapper callback methods

Supported mobile development platforms

Many iOS apps are created using mobile development platforms, rather than using the Apple environment that targets only iOS devices. You can wrap iOS apps that were created using these mobile development platforms:

- PhoneGap
- IBM Worklight
- Xamarin

NOTE: Tunneling support for Xamarin apps has restrictions as described in AppTunnel support in Xamarin apps .

MobileIron AppConnect components for wrapped apps

Wrapped AppConnect apps work with the following MobileIron components:

MobileIron component	Description
MobileIron Core	The MobileIron on-premise server which provides security and management for an enterprise's devices, and for the apps and data on those devices. An administrator configures the security and management features using a web portal.
MobileIron Connected Cloud	MobileIron's cloud offering that has the same functionality as MobileIron Core.
MobileIron Cloud	MobileIron's cloud offering that provides similar functionality as MobileIron Core. However, it does not support all the AppConnect features that MobileIron Core supports.
Standalone Sentry	The MobileIron server which provides secure network traffic tunneling from your app to enterprise servers.
Mobile@Work for iOS	A MobileIron client app that runs on an iOS device. It interacts with MobileIron Core or Connected Cloud to get current security and management information for the device. It interacts with the AppConnect library to communicate necessary

MobileIron component	Description
	information to your app.
The MobileIron Go app	A MobileIron client app that runs on an iOS device. It interacts with MobileIron Cloud to get current security and management information for the device. It interacts with the AppConnect library to communicate necessary information to your app.
The MobileIron AppStation app	A MobileIron client app that runs on an iOS device. It interacts with MobileIron Cloud. It can be used on the device instead of MobileIron Go when the MobileIron Cloud tenant supports Mobile Apps Management (MAM) but not Mobile Device Management (MDM). It interacts with the AppConnect library to communicate necessary information to your app.
AppConnect for iOS Wrapper Library	Provided by the AppConnect wrapping technology, the wrapper library provides AppConnect capabilities to your app. It provides your AppConnect app management and security capabilities, and facilitates communication between your app and the MobileIron client app.

Note The Following:

- MobileIron Core, MobileIron Connected Cloud, and MobileIron Cloud are each also referred to as a MobileIron server.
- Mobile@Work, MobileIron Go, and MobileIron AppStation are each also referred to as a MobileIron client app.

IMPORTANT: Some AppConnect features depend on the version of MobileIron Core, MobileIron Cloud, Standalone Sentry, and the MobileIron client app.

Using a wrapped app

An iOS device user can use a wrapped AppConnect app only if:

- The device user has been authenticated through the MobileIron server.
  The user must use the MobileIron client app to register the device with the MobileIron server. Registration authenticates the device user. Once registered, the device user can use a secured enterprise app.
- The MobileIron server administrator has authorized the device user to use the app.
- The device user has entered a secure apps passcode or Touch ID/Face ID.
  The MobileIron server administrator configures whether a secure apps passcode, also called the AppConnect passcode, is required, and configures its complexity rules. The administrator also configures whether using Touch ID/Face ID, if available on the device, is allowed instead of the AppConnect passcode.

NOTE: The AppConnect passcode is not the same as the passcode used to unlock the device.

Product versions required

To develop a wrapped AppConnect app, you need certain products. MobileIron supports a set of product versions, and a larger set of product versions are compatible with apps wrapped with this version of the AppConnect for iOS Wrapping Library.

- **Supported product versions:** The functionality of the product and version with currently supported releases was systematically tested as part of the current release and, therefore, will be supported.
- **Compatible product versions:** The functionality of the product and version with currently supported releases has not been systematically tested as part of the current release, and therefore not supported. Based on previous testing (if applicable), the product and version is expected to function with currently supported releases.

The following table summarizes supported and compatible product versions. This information is current at the time of this release. For MobileIron product versions released after this release, see that product version's release notes for the most current support and compatibility information.

TABLE 1. SUPPORT AND COMPATIBILITY

Product	Supported versions	Compatible versions
iOS	11.0.0 - 13.5.1	9.0 and lower are not supported
MobileIron Core and Connected Cloud	10.5.0.0, 10.6.0.0, 10.7.0.0	10.3.0.0 - 10.4.0.0
MobileIron Cloud	70	Not applicable
MobileIron Go	5.4.0	4.0.0 - 5.1.0
Standalone Sentry	9.7.3, 9.8.1	9.5.0 - 9.6.0
Mobile@Work for iOS	12.3.0, 12.2.2	11.0.0 - 12.1.0
MobileIron AppStation	1.3.0	Not applicable
Xcode (Xcode command-line tools are used for re-signing the wrapped app and also used by the wrapping tool)	11	10
OS X (OS X is used for re-signing the wrapped app and for running the wrapping tool)	10.13	10.10.5 - 10.14.3
Xamarin (Indicates support and compatibility for wrapping apps built with specific Xamarin versions)	8.4.0	Between 7.2.1 and 8.4.0 After 8.4.0

IMPORTANT: Some AppConnect features depend on the version of MobileIron Core, MobileIron Cloud, Standalone Sentry, and the MobileIron client app.

Related topics

For information about AppConnect for iOS and available features from the perspective of a MobileIron server administrator:

- MobileIron Core or Connected Cloud: The MobileIron Core AppConnect and AppTunnel Guide
- MobileIron Cloud: The *MobileIron Cloud Administrator Guide*

Securing and managing a wrapped iOS AppConnect app

A MobileIron server administrator configures how mobile device users can use secure enterprise applications. The administrator sets the following app-related settings that impact your wrapped app's behavior:

- Authorization
- AppConnect passcode and Touch ID/Face ID policy
- Tunneling
- Certificate authentication to enterprise services
- Data loss prevention policies
- Custom keyboard control
- Log messages based on log levels
- App-specific configuration from the MobileIron server
  This capability requires some additional app development.

Additionally, the AppConnect passcode and the device passcode impact data encryption of AppConnect-related data such as configurations and certificates, and app-specific data. See Data encryption.

The following steps show the flow of information from the MobileIron server to a wrapped app:

1. The MobileIron server administrator decides which app-related settings to apply to a device or set of devices.
2. The server sends the information to the MobileIron client app on the device.

The MobileIron client app passes the information to the wrapped AppConnect app. The MobileIron client app and the AppConnect for iOS Wrapper Library enforce the app-related settings.

Authorization

The MobileIron server administrator determines:

- whether or not each device user is authorized to use each secure enterprise app.
  When an unauthorized user launches the app, the MobileIron client app displays a message to the user, and the app exits.
- the situations that cause an authorized device user to become unauthorized.
  These situations include, for example, when the device OS is compromised. The MobileIron client app reports device information to the MobileIron server. The server then determines whether to change the user to unauthorized based on security policies on the server.
  When a user becomes unauthorized, the MobileIron client app displays a message to the user, and the app exits.
- the situations that retire the app.
  Retiring an app means that the user is not authorized to use it and the app's data is deleted. The MobileIron client app displays a message to the user, and the app exits. Furthermore, the AppConnect for iOS Wrapper

Library removes data associated with the app. Specifically, the wrapper library removes all data in the application's sandbox and in the application's keychain. It also resets the application's default settings.

NOTE: When an app is retired, the wrapper library removes the app's data. When a user is unauthorized but the app is not retired, the app cannot run, so the user cannot access the data. However, the wrapper library does not remove the data. The reason is that an unauthorized user can become authorized again, and therefore the data should become available again.

AppConnect passcode and Touch ID/Face ID policy

The MobileIron server administrator determines:

- whether the AppConnect passcode or Touch ID/Face ID is required, which requires the device user to enter a passcode or Touch ID/Face ID to access any secure enterprise apps.
- the complexity of the AppConnect passcode.
- the auto-lock time for the AppConnect passcode or Touch ID/Face ID. After this period of inactivity in AppConnect apps, the device user is locked out of the apps until he enters the AppConnect passcode or Touch ID/Face ID.

The AppConnect for iOS Wrapper Library and the MobileIron client app enforce the AppConnect passcode or Touch ID/Face ID policy as follows:

- The MobileIron server notifies the MobileIron client app when the server administrator has enabled an AppConnect passcode or Touch ID/Face ID. The client app prompts the user to set the AppConnect passcode or enter the Touch ID/Face ID the next time that the device user launches or switches to a secure enterprise app.
- The client app prompts the user to set the AppConnect passcode the next time the device user launches or switches to a secure enterprise app after the server has notified the client app that the passcode's complexity rules have changed.
- The user is prompted to enter the AppConnect passcode or Touch ID/Face ID when the user subsequently launches or switches to a secure enterprise app but the auto-lock time has expired.
- The user is prompted to enter the passcode or Touch ID/Face ID when the auto-lock time expires *while* the user is running a secure enterprise app.

Tunneling

Using MobileIron's AppTunnel feature, a secure enterprise app can securely tunnel HTTP and HTTPS network connections from the app to servers behind a company's firewall. A Standalone Sentry is necessary to support AppTunnel with HTTP/S tunneling. The MobileIron server administrator handles all HTTP/S tunneling configuration on the server. Once the administrator has configured tunneling for the app on the server, the AppConnect for iOS Wrapper Library, the MobileIron client app, and a Standalone Sentry handle tunneling for the app.

Consider the following information to ensure that your wrapped app can successfully tunnel network connections:

- AppTunnel supports only NSURLConnection and NSURLSession
- Accessing sockets directly does not use AppTunnel
- AppTunnel supports redirects and authentication requests on HTTP/S upload
- AppTunnel support in Xamarin apps
- AppTunnel with TCP tunneling

- When to make network requests when using AppTunnel

AppTunnel supports only NSURLConnection and NSURLSession

An app accesses its enterprise servers as it normally would using URL requests, using the iOS APIs NSURLConnection and NSURLSession.

Note The Following:

- AppTunnel with HTTP/S tunneling does not support using NSURLSession in a background session. The traffic does not reach its destination.
- Apps can also use networking libraries that use NSURLConnection or NSURLSession. For example, apps can use AFNetworking 3.0 because it uses NSURLSession.
- An app that uses WKWebView cannot use AppTunnel with HTTP/S tunneling.

Accessing sockets directly does not use AppTunnel

AppTunnel with HTTP/S tunneling is not supported if the app:

- accesses sockets directly.
- uses APIs that access sockets directly.

In these cases, the app cannot access a host behind the enterprise's firewall using AppTunnel with HTTP/S tunneling.

For example, AppTunnel with HTTP/S tunneling is not supported with the following APIs:

- Apple's reachability APIs that detect network and host connectivity
- CFNetwork APIs
- ASIHTTPRequest

NOTE: Network connections using sockets for TCP connections can tunnel data by using AppTunnel with TCP tunneling. See AppTunnel with TCP tunneling.

AppTunnel supports redirects and authentication requests on HTTP/S upload

When an app uses AppTunnel with HTTP/S tunneling, AppTunnel handles the following HTTP/S upload scenarios:

- HTTP/S redirect responses from the network server (HTTP/S 3XX status code).
  If a network server redirects an HTTP/S upload request (tunneled or not) to another URL that the MobileIron server administrator has configured for tunneling, the request is tunneled.
- Authentication required response from the network server (HTTP/S 401 status code).
  The AppTunnel feature handles sending a second HTTP/S request with authentication credentials.

AppTunnel support in Xamarin apps

Apps built with the Xamarin development platform are written in C#. They can access network servers various ways. AppTunnel with HTTP/S tunneling is supported only as follows:

- The app uses the NSURLConnection or NSURLSession APIs exposed to C# through the Xamarin.iOS binding.

- The app uses the ModernHttpClient library with NSURLSession. The ModernHttpClient library with CFNetwork will not work.
  For example, the app initializes the instance of the ModernHttpClient as follows:

```
var httpClient = new HttpClient (new NativeMessageHandler ());
```

AppTunnel with TCP tunneling

AppTunnel can tunnel TCP traffic between an app and a server behind the company's firewall. AppTunnel with TCP tunneling does not require an app to be an AppConnect app; both AppConnect apps and standard apps can use AppTunnel with TCP tunneling. The MobileIron server administrator configures AppTunnel with TCP tunneling, including installing MobileIron Tunnel (an iOS app) on the device.

When to make network requests when using AppTunnel

If a wrapped app makes HTTP/S network requests before the AppConnect library in the app has received the AppTunnel rules from the MobileIron server, the network requests will fail for URLs behind the enterprise's firewall.

When this occurs, an app should try the request again. For example, the app can try the request again after some time has elapsed, or the next time it becomes active.

Alternatively, an app can wait to make a network request until after the AppConnect library has received the AppTunnel rules. An AppConnect wrapper callback method is available for the app to know when the rules have been received. See Callback method involving network requests with AppTunnel.

Certificate authentication to enterprise services

An AppConnect app can send a certificate to identify and authenticate the app user to an enterprise service when the app uses an HTTPS connection. The MobileIron server administrator configures on the server which certificate for the app to use, and which connections use it. The AppConnect library, which is part of every AppConnect app, makes sure the connection uses the certificate. No additional development is required for the app.

Supported networking methods

Certificate authentication to enterprise services is supported only if your app uses one of the following to access the enterprise service:

- NSURLConnection
- NSURLSession

Certificate authentication to enterprise services does not support using NSURLSession in a background session.

- Networking libraries that use NSURLConnection or NSURLSession.
- UIWebView

Unsupported networking methods

Certificate authentication to enterprise services using other networking methods is not supported. For example, the following are not supported:

- accessing sockets directly
- WKWebView and other APIs that access sockets directly
  For example, these APIs are not supported: CFNetwork, ASIHTTPRequest, and Apple's reachability APIs that detect network and host connectivity.

Data loss prevention policies

An app can leak data if it uses iOS features such as copying to the iOS pasteboard, document interaction (Open In), and print capabilities. A MobileIron server administrator specifies on the server whether each app is allowed to use each of these features. The AppConnect for iOS Wrapper Library enforces the policies in the app.

Specifically:

- the print policy indicates whether the app is allowed to use: AirPrint, any future iOS printing feature, any current or future third-party libraries or apps that provide printing capabilities.
- The pasteboard policy specifies whether your app is allowed to copy content *to* the iOS pasteboard. If copying content is allowed, the policy specifies whether all apps, or only AppConnect apps, can paste the copied content *from* the pasteboard.
- The drag and drop policy specifies whether AppConnect apps can drag content to all other apps, to only other AppConnect apps, or not at all.
- The Open In policy specifies the apps, including the extensions that apps provide, with which your app can share documents. The policy specifies no apps, all apps, all AppConnect apps, or a set of apps. A set of apps is called the whitelist. Whether your app can share documents with the native iOS mail app is also controlled by the Open In policy.

In iOS 11 through the most recently released version as supported by MobileIron, regardless of the Open In policy, iOS always displays all apps that support the document type as possible target apps. However, if a user taps on an app that is not allowed based on the Open In policy, nothing happens. On iOS versions prior to iOS 11, only allowed apps are displayed. The iOS behavioral change impacts all wrapped apps, regardless what version of the wrapper they are wrapped with.

- The Open From policy specifies the apps, including the extensions that apps provide, from which your app can receive documents when the other app uses the Open In iOS feature. The policy specifies no apps, all apps, all AppConnect apps, or a set of apps. A set of apps is called the whitelist.

The administrator applies the appropriate policies to a set of devices. Sometimes more than one set of policies exists on the MobileIron server for an app if different users require different policies.

Custom keyboard control

Custom keyboard extensions sometimes send data to servers when a device user enters data into an app. They send this data for assistance with word-prediction, for example. To stop this potentially harmful data loss, the

MobileIron server administrator configures whether custom keyboards are allowed for an app by setting a key-value pair in the app's configuration. The key is called MI_AC_IOS_ALLOW_CUSTOM_KEYBOARDS. The key-value pair is consumed by the AppConnect library; your app does not receive it.

When the key is present, the AppConnect library controls custom keyboard use according to the key's value. If the value is true, the AppConnect library allows the AppConnect app to use custom keyboards. If the value is false, the AppConnect library does not allow custom keyboard use.

If the server administrator does not include the key-value pair for your app, the AppConnect library allows the app to use custom keyboards. However, in this case, the AppConnect library gives precedence to the behavior your app specifies in its implementation of the `-shouldAllowExtensionPointIdentifier:` method on your AppDelegate. For example, your`-shouldAllowExtensionPointIdentifier:` can reject all custom keyboards.

Log messages based on log levels

In a wrapped app, the AppConnect for iOS Wrapper Library supports logging messages according to the log level that the MobileIron server administrator specifies for the app. The wrapper library logs these messages to the device's console. It also logs the messages to log files if specified by the administrator. The log data provides information to help troubleshoot issues with the apps.

App-specific configuration from the MobileIron server

Handling app-specific configuration from the MobileIron server requires some application development before wrapping the app. If you do not use this feature, the app continues to set up its configuration as it always has.

Typically, wrapped apps do not use this feature. However, if you have application developer resources, you can take advantage of this feature.

You determine the app-specific configuration that your app requires from the MobileIron server. Examples are:

- the address of a server that the app interacts with
- whether particular features of the app are enabled for the user
- user-related information from LDAP, such as the user's ID and password
- certificates for authenticating the user to the server that the app interacts with

For details about how to receive app-specific configuration from the MobileIron server, see App-specific configuration callback methods.

Data encryption

AppConnect-related data

The MobileIron client app and the Wrapper library work together to use encryption to protect AppConnect-related data, such as configurations and certificates, on the device.

The encryption key is not stored on the device. It is either:

- Derived from the device user's AppConnect passcode.
- Protected by the device passcode if the administrator does not require an AppConnect passcode.
- Protected by the device passcode if the device user uses Touch ID/Face ID to access AppConnect apps.

If no AppConnect passcode or device passcode exists, the data is encrypted, but the encryption key is not protected by either passcode.

App data files

The AppConnect passcode does not impact encryption of the app's data. The app's data is encrypted only if both of the following are true:

- the device has a device passcode.
  The MobileIron server administrator determines whether a device passcode is required.
- the app enables iOS data protection on its files.
  The wrapper ensures that data that the app writes with the following APIs has a data protection level of either NSFileProtectionCompleteUntilFirstUserAuthentication or NSFileProtectionComplete:
  - NSArray
  - NSData
  - NSDictionary
  - NSFileManager
  - NSFileWrapper
  - NSKeyedArchiver
  - NSString
  - UIDocument
  - NSPersistentStoreCoordinator

Note that Apple defines the data protection levels as follows:

- NSFileProtectionCompleteUntilFirstUserAuthentication
  The file is stored in an encrypted format on disk and cannot be accessed until after the device has booted. After the user unlocks the device for the first time, your app can access the file and continue to access it even if the user subsequently locks the device.
- NSFileProtectionComplete
  The file is stored in an encrypted format on disk and cannot be read from or written to while the device is locked or booting.

AppConnect wrapper callback methods

Typically, you can wrap apps without any changes to the app. However, the following AppConnect wrapper callback methods are available to wrapped apps:

- App-specific configuration callback methods
  These methods are necessary for apps that want to receive app-specific configuration from the MobileIron server.
- Callback method involving network requests with AppTunnel
  This method is a convenient way to make sure your app does not make network requests that depend on AppTunnel until the AppConnect library in the app has received the AppTunnel rules.

App-specific configuration callback methods

Overview of app-specific configuration from the MobileIron server

Handling app-specific configuration from the MobileIron server requires some application development before wrapping the app. If you do not use this feature, the app continues to set up its configuration as it always has.

Typically, wrapped apps do not use this feature. However, if you have application developer resources, you can take advantage of this feature.

You determine the app-specific configuration that your app requires from the MobileIron server. Examples are:

- the address of a server that the app interacts with
- whether particular features of the app are enabled for the user
- user-related information from LDAP, such as the user's ID and password
- certificates for authenticating the user to the server that the app interacts with

Each configurable item is a key-value pair. Each key and value is a string. A MobileIron server administrator specifies on the server the key-value pairs for each app. The administrator applies the appropriate set of key-value pairs to a set of devices. Sometimes more than one set of key-value pairs exists on the server for an app if different users require different configurations. For example, the administrator can assign a different server address to users in Europe than to users in the United States.

NOTE: When the value is a certificate, the value contains the base64-encoded contents of the certificate, which is a SCEP or PKCS-12 certificate. If the certificate is password encoded, the MobileIron server automatically sends another key-value pair. The key's name is the string *<name of key for certificate>*_MI_CERT_PW. The value is the certificate's password.

Methods for receiving app-specific configuration from the MobileIron server

To receive app-specific configuration from the MobileIron server, the developer implements one or both of the following callback methods on the class that implements the UIApplicationDelegate protocol:

In Objective-C:

```
-(NSString *)appConnectConfigIs:(NSDictionary *)config;
-(NSString *)appConnectConfigChangedTo:(NSDictionary *)config;
```

In Swift:

```
@objc func appConnectConfigIs(_ config: [String : Any]) -> String?
@objc func appConnectConfigChangedTo(_ config: [String : Any]) -> String?
```

Both methods:

- Have a `config` parameter.
  The parameter is an `NSDictionary` object which contains the current key-value pairs for the app-specific configuration. The app applies the values according to its requirements and logic.
- Return the value `nil` if the configuration was successfully applied. Otherwise, return a string that describes the error that occurred in applying the configuration.

When to use -appConnectConfigIs: versus -appConnectConfigChangedTo:

When to use -appConnectConfigIs:

Use `-appConnectConfigIs:` to find out what the app-specific configuration is. The AppConnect library calls `-appConnectConfigIs:` when:

- the app is launched or relaunched.
- the configuration changes on the MobileIron server.

When using `-appConnectConfigIs:`, the app can depend on the configuration values being available in memory at any time.

Some examples for using `-appConnectConfigIs:` to get app-specific configuration from the MobileIron server:

- User permissions that are accessed throughout the life of the app.
- A server address that the app uses to connect to a server, or reconnect after losing network connectivity.

When to use -appConnectConfigChangedTo:

Use `-appConnectConfigChangedTo:` when the app needs to take some action when the app-specific configuration changes. The AppConnect library calls `-appConnectConfigChangedTo:` when the configuration changes on the MobileIron server.

When using `-appConnectConfigChangedTo:`, the app knows when configuration values change, and should take the appropriate action immediately. The configuration values are not available in memory after an app relaunch, but because the app already took the necessary actions, the unavailability does not matter.

Some examples for using `-appConnectConfigChangedTo:` to receive changes to the MobileIron server app-specific configuration are:

- A setting indicating whether the user can access certain data.
  If access is denied, the app removes that data immediately. Once the app has removed the data, the app will not use the setting again. Therefore, the setting's availability in memory does not matter.
- User registration information
  If the app requires that the user registers to a server one time, the app registers the user when it receives the registration information. Once registered, the app will not use the registration information again. Therefore, the information's availability in memory does not matter.

Details about when each method is called

The following table provides the details about when each method is called, with the differences in behavior shown in **bold**.

Action	-appConnectConfigChangedTo:	-appConnectConfigIs:
The app is launched for the first time and the MobileIron server has app-specific configuration. NOTE: In this case, the app-specific configuration changes from nil to some values.	Called	Called
The app is launched for the first time and the MobileIron server has no app-specific configuration.	Not called	Not called
The app is re-launched after being terminated and the configuration **has not** changed since the app last ran. (Termination is due to, for example, a force-close by the user, an iOS termination, or an app crash).	Not called	Called
The app is re-launched after being terminated and the configuration **has changed** since the app last ran.	Called	Called
A change has occurred to the app-specific configuration on the MobileIron server while the app is running.	Called	Called

Handling app-specific configuration in Xamarin apps

Wrapped Xamarin apps can handle app-specific configuration from the MobileIron server.

The developer implements one or both of the following methods in the UIApplicationDelegate class or subclass:

```
[Export ("appConnectConfigIs:")]

public string AppConnectConfigIs (NSDictionary config) {

       // The config parameter contains the current key-value pairs for the
       // app-specific configuration.
       // Apply the configuration according to the application's requirements and logic.

       return null; // Return null on success. If a error occurs, return a string
                    // describing the error.
}

[Export ("appConnectConfigChangedTo:")]

public string AppConnectConfigChangedTo (NSDictionary config) {

       // The config parameter contains the current key-value pairs for the
       // app-specific configuration.
       // Apply the configuration according to the application's requirements and logic.

       return null; // Return null on success. If a error occurs, return a string
                    // describing the error.
}
```

Related topics

When to use -appConnectConfigIs: versus -appConnectConfigChangedTo:.

Callback method involving network requests with AppTunnel

Overview of network requests when using AppTunnel

A wrapped app can use MobileIron AppTunnel, as described in Tunneling, to securely tunnel HTTP and HTTPS network connections from the app to servers behind a company's firewall. An administrator configures the MobileIron server with the AppTunnel rules for the app. The AppTunnel rules specify which URL requests to tunnel.

When an app first launches:

1. Control switches to the MobileIron client app (Mobile@Work for MobileIron Core, MobileIron Go for MobileIron Cloud).
2. The MobileIron client app gets the configuration and policy settings for the app, including the AppTunnel rules, from the MobileIron server.
3. The MobileIron client app delivers the settings to the AppConnect library in the app.
4. Control switches back to the app.

When the app makes a network request, such as when loading a web view, the AppConnect library determines if the URL matches one of the AppTunnel rules. If a match is found, the AppConnect library tunnels the request.

Therefore, if an app makes network requests before the AppConnect library has received the AppTunnel rules, the network requests will fail for URLs behind the enterprise's firewall.

When this occurs, an app should try the request again. For example, the app can try the request again after some time has elapsed, or the next time it becomes active.

Alternatively, an app can wait to make a network request until after the AppConnect library has received the AppTunnel rules. An AppConnect wrapper callback method is available for the app to know when the rules have been received.

Wrapper callback method for when to send network requests

You can use the following callback method to make sure your wrapped app does not send a network request until after the AppConnect library in the app has received the AppTunnel rules. Implement the method on the class that implements the UIApplicationDelegate protocol.

In Objective-C:

```
-(void)appConnectStateChangedTo:(NSInteger)newState;
```

In Swift:

```
@objc func appConnectStateChangedTo(_ newState: Int)
```

When in the app life cycle the AppConnect library calls -appConnectStateChangedTo:

The AppConnect library calls the `-appConnectStateChangedTo:` method when:

- the app is first launched.
- the app is launched after being terminated.
- the app is launched after the device is restarted.

In these situations, the AppConnect library calls the method when the library has either:

- Received the AppTunnel rules.
- Determined that it will not receive the AppTunnel rules because the MobileIron client app is not installed on the device.
  Without the MobileIron client app, the wrapped app does not run as an AppConnect app. No AppConnect features are available to it.

NOTE: When an app is unauthorized, the MobileIron client app displays a message and the app exits. In that case, the AppConnect library does not call the -appConnectStateChangedTo: method.

The -appConnectStateChangedTo: method parameter

The following table shows the possible values of the `newState` parameter in the `-appConnectStateChangeTo:` method:

Value of newState	Meaning of value	Action your app takes
0	The AppConnect library in the app will not receive AppTunnel rules because the MobileIron client app is not installed on the device. Without the MobileIron client app, the wrapped app does not run as an AppConnect app. No AppConnect features are available, including AppTunnel. Network requests to URLs behind a firewall will fail.	Behave as a standard, non-AppConnect app. Typically, this requires no changes to your app.
1	The AppConnect library in the app has received the AppTunnel rules.	The app can now make network requests that will be tunneled.

AppConnectStateChangedTo callback method in Xamarin apps

Wrapped Xamarin apps can use a callback method to determine when to send network requests that use AppTunnel. Specifically, the developer implements the following method in the UIApplicationDelegate class or subclass:

```
[Export ("appConnectStateChangedTo:")]

public AppConnectStateChangedTo (System.nint newState) {

	// newState parameter:
	//
	// 0 - The AppConnect library in the app will not receive AppTunnel rules,
	//       or any other AppConnect configurations and policies, because
	//      the MobileIron client app is not installed on the device.
	//      Network requests to URLs behind a firewall will fail.
	//      The app should behave   as a standard, non-AppConnect app.
	//
	// 1 - The AppConnect library in the app has received the AppTunnel rules. The app can
	//      now make network requests that will be tunneled.

}
```

How to wrap an iOS app

You can wrap an iOS app if:

- It is an app developed in-house.
- It is an app developed by a third-party organization for you to distribute.

IMPORTANT: You **cannot** wrap an app if you got the app (IPA file) from the Apple App Store.

You can wrap an iOS app one of these ways:

- Submit the app to AppConnect Wrapping Portal for immediate turnaround.
  You re-sign the returned app using a script that MobileIron provides to you.
  See Using the AppConnect Wrapping Portal.
- Use the AppConnect App Wrapper, also known as the wrapping tool.
  Provided by MobileIron, this OS X app wraps your app. Its output is a wrapped, signed app. It also can output a wrapped, unsigned app, which you can give to another party to sign. Use the wrapping tool if you cannot submit the app to MobileIron due to your security policies.
  See Using the AppConnect App Wrapper (wrapping tool).

Be sure the unwrapped app installs and runs according to your requirements before you wrap the app.

Related topics

- Re-signing an app before wrapping it
- Using the AppConnect Wrapping Portal
- Re-signing the wrapped app
- Using the AppConnect App Wrapper (wrapping tool)

Re-signing an app before wrapping it

Before you wrap an app developed by a third-party developer, re-sign it with your own enterprise's signing identity using the sign_wrapped_app.sh script. Then wrap the app, and sign it again as described in Using the AppConnect Wrapping Portal or Using the AppConnect App Wrapper (wrapping tool)

For in-house apps, re-signing the app before wrapping it is typically not necessary because it is already signed with your enterprise's signing identity.

Before you begin

1. Login to help.mobileiron.com.
2. Click the Software tab.
3. Download the sign_wrapped_app.sh script.
4. Make sure the signing certificate that you created for the app is in the MacOS computer's login keychain.
5. Put the IPA file of the unwrapped app and the sign_wrapped_app.sh script in the same directory for convenient access.

The signing script is supported only with the versions MacOS and Xcode listed in Product versions required.

IMPORTANT: You must download the signing script sign_wrapped_app.sh for each new release of AppConnect for iOS. Previous versions of the script will not work.

Procedure

1. Open the Terminal application on the MacOS computer.
2. Change to the directory containing the IPA file of the unwrapped app and the sign_wrapped_app.sh script.
3. Make sure that the sign_wrapped_app.sh script is executable. For example:
   `$chmod 755 sign_wrapped_app.sh`
4. Run the script, specifying two parameters: the app's signing certificate and the IPA file of the unwrapped app. For example:
   `$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" myApp.ipa`

Specify the name of the signing certificate in double quotes. The name has the format `"iPhone Distribution: <certificate name>"` where `<certificate name>` is typically the name of your company.

5. When prompted, enter the password to unlock your keychain.
   The script continues to run, displaying the following output when successful:
   /var/folders/6g/z1_193_x0lj6jkzmysxl5wz80000gq/T//resign-QJ4wZrPR/Payload/myApp.app/MISandbox.framework/Versions/A: replacing existing signature
   /var/folders/6g/z1_193_x0lj6jkzmysxl5wz80000gq/T//resign-QJ4wZrPR/Payload/myApp.app: replacing invalid existing signature
   $
   The script replaces the IPA file with a signed IPA file. The signed IPA file is the file you will wrap.

Optionally, you can specify a different output file for the signed IPA file. Use the -o option as follows:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -o mySignedApp.ipa myApp.ipa
```

Using the AppConnect Wrapping Portal

Use the AppConnect Wrapping Portal to receive the wrapped app within minutes. The AppConnect Wrapping Portal wraps the iOS app with the latest version of the AppConnect for iOS Wrapper Library. The AppConnect Wrapping Portal does not keep either the unwrapped or wrapped version of your app. You can upload apps that are up to 200 MB.

The AppConnect Wrapping Portal is available at help.mobileiron.com in the **Developer > Wrapped Apps** tab.

Subscribe to https://trust.mobileiron.com for AppConnect Wrapping Portal system status and updates.

Do the following high-level steps:

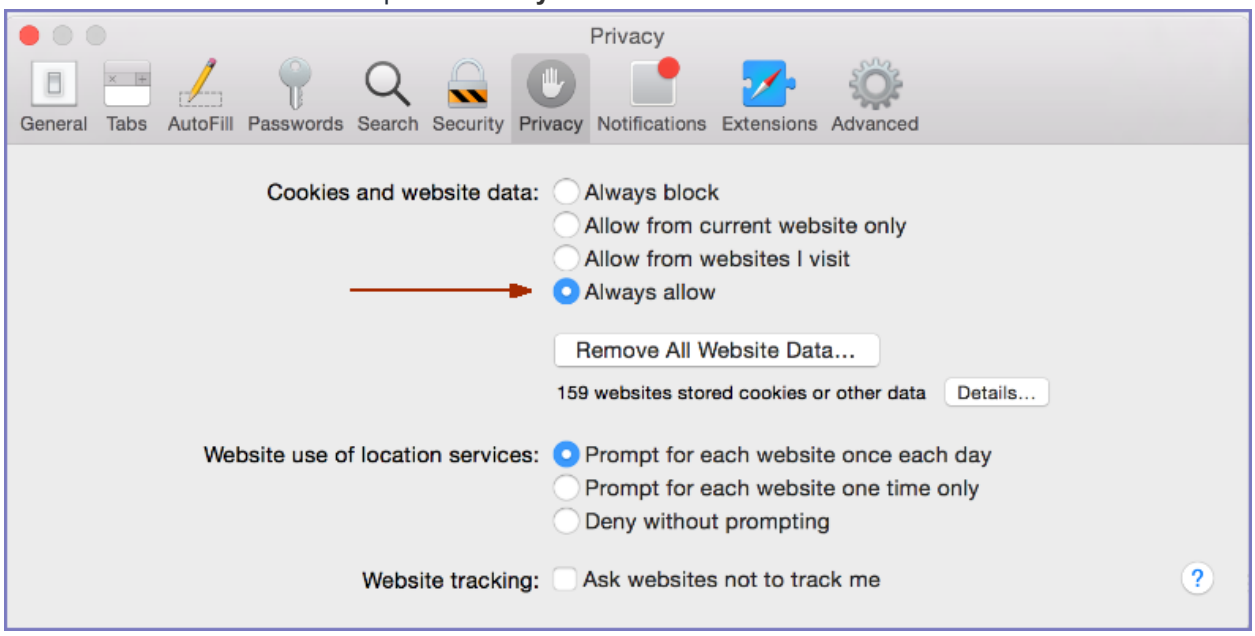
1. Login to help.mobileiron.com
2. Upload an unwrapped app and wrap it
3. Download the wrapped app and signing script
4. Re-sign the wrapped app as described in Re-signing the wrapped app.

Important: Do **not** submit an app for wrapping if you got the app (IPA file) from the Apple App Store.

Before you begin

Before you use the AppConnect Wrapping Portal:

- Be sure the unwrapped app installs and runs according to your requirements before you wrap the app.
- Sign the app according to Apple's requirements.
- Set Safari browser privacy option
  If you are using Safari to access the AppConnect Wrapping Portal, in **Safari > Preferences > Privacy,** set the **Cookies and website data** option to **Always allow**.



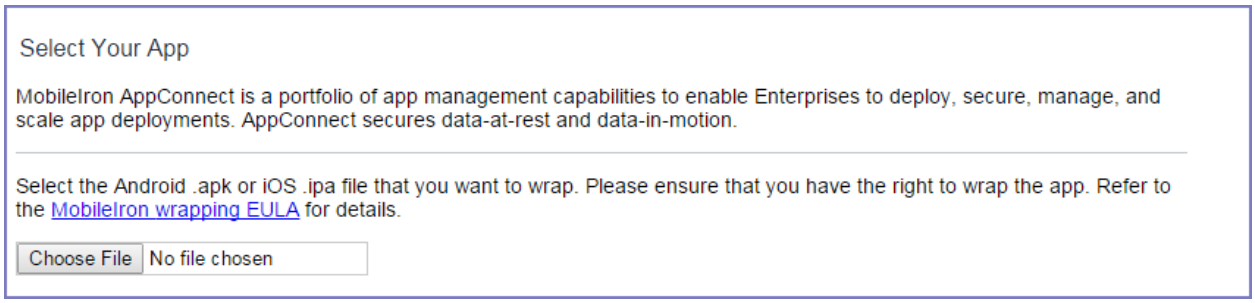
Login to help.mobileiron.com

Enter your login ID and password at https://help.mobileiron.com.

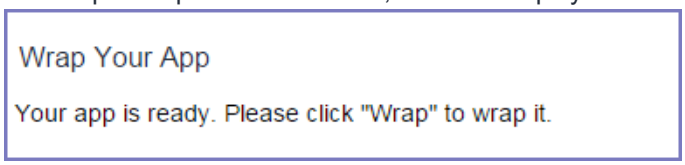
The home page displays.

Upload an unwrapped app and wrap it

1. Click **Developer > Wrapped Apps** in the tab bar.
   The **Wrapped Apps** page displays.
2. Click **Create New Wrapped App**.
3. Read and accept the license agreement, if presented.
   The license agreement is presented the first time that you click **Create New Wrapped App**.
   After accepting the license agreement, the **Select Your App** page displays.



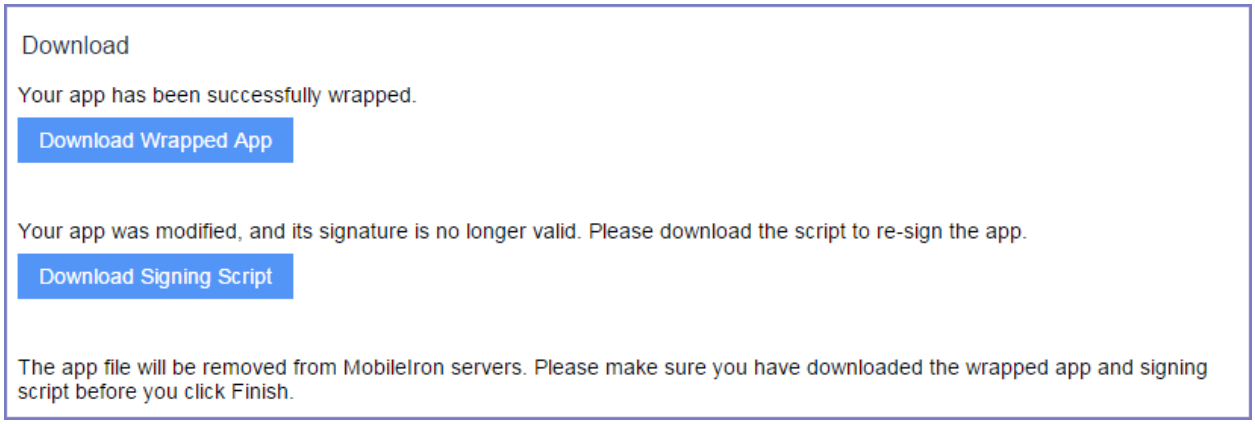
4. Click **Choose File**.
   A dialog box for choosing the file opens.
5. Select the IPA file of an unwrapped app in your computer's folders.
6. Click **Next** on the **Select Your App** page.
   The portal uploads the IPA file, and then displays the **Wrap Your App** page.



7. Click **Wrap**.

Download the wrapped app and signing script

When you click **Wrap**, after a few moments, depending on the size of the app, the **Download** page displays.



1. Click **Download Wrapped App**.
   The portal downloads the wrapped app to your computer.
2. Click **Download Signing Script**.
   The portal downloads the signing script to your computer.

IMPORTANT: You must download the signing script sign_wrapped_app.sh for each new release of AppConnect for iOS. Previous versions of the script will not work.

3. Click **Finish**.
   The portal removes both the wrapped and unwrapped version of the app.
4. Re-sign the wrapped app with the signing script as described in Re-signing the wrapped app.

NOTE: if wrapping fails, the portal displays the reason. You can click **Open Support Ticket** if you need help.

Re-signing the wrapped app

When you receive your wrapped app from the AppConnect Wrapping Portal, re-sign the app using the script that MobileIron returns with the app. The script is called sign_wrapped_app.sh.

The signing script is supported only with the versions MacOS and Xcode listed in Product versions required.

IMPORTANT: You must download the signing script sign_wrapped_app.sh for each new release of AppConnect for iOS. Previous versions of the script will not work.

Before you run the script

- Make sure the signing certificate that you created for the app is in the MacOS computer's login keychain.
- Put the IPA file of the wrapped app and the sign_wrapped_app.sh script in the same directory for convenient access.

Running the sign_wrapped_app.sh script

1. Open the Terminal application on the MacOS computer.
2. Change to the directory containing the IPA file of the wrapped app and the sign_wrapped_app.sh script. For example:
   ```
   $cd ~/wrapping
   ```
3. Make sure that the sign_wrapped_app.sh script is executable. For example:
   ```
   $chmod 755 sign_wrapped_app.sh
   ```
4. Run the script, specifying two parameters: the original app's signing certificate and the IPA file of the wrapped app. For example:
   ```
   $./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" myApp-wrapped.ipa
   ```

   NOTE: Specify the name of the signing certificate of the original, unwrapped app in double quotes. This name is also the original app's signing identity. The name has the format `"iPhone Distribution: <certificate name>"` where `<certificate name>` is typically the name of your company.
5. When prompted, enter the password to unlock your keychain.
   The script continues to run, displaying the following output when successful:
   ```
   /var/folders/6g/z1_193_x0lj6jkzmysxl5wz80000gq/T//resign-QJ4wZrPR/Payload/myApp.app/MISandbox.framework/Versions/A: replacing existing signature
   /var/folders/6g/z1_193_x0lj6jkzmysxl5wz80000gq/T//resign-QJ4wZrPR/Payload/myApp.app: replacing invalid existing signature
   $
   ```
   The script replaces the IPA file with a signed IPA file. The signed IPA file is the file you distribute to device users.

Optionally, you can specify a different output file for the signed IPA file. Use the -o option as follows:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -o mySignedWrappedApp.ipa myApp-wrapped.ipa
```

Specifying custom entitlements

By default, the sign_wrapped_app.sh script takes the app entitlements (for example, enabling iCloud, push notifications, and App Sandbox) from the app binary. You can override this behavior by specifying an optional parameter when running the sign_wrapped_app.sh script. The parameter names an entitlements plist file.

-e <*entitlements plist file name*>

For example:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -e entitlements.plist myApp-
wrapped.ipa
```

Specifying a new provisioning profile

By default, the sign_wrapped_app.sh script uses the distribution provisioning profile that is embedded in the app's IPA file. You can override this behavior by specifying an optional parameter when running the sign_wrapped_ app.sh script. The parameter names another distribution provisioning profile.

-p *<path to distribution provisioning profile>*

For example, when the provisioning profile is in the current directory:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName"
       -p MyProvisioningProfile.mobileprovision myApp-wrapped.ipa
```

Specifying new version numbers

By default, the sign_wrapped_app.sh script does not change the version numbers in the IPA file. These version numbers are:

- the release version number (CFBundleShortVersionString key's value in the Info.plist of the app)
- the build version number (CFBundleVersion key's value in the Info.plist of the app)

Typically, you do not need to specify new version numbers for the signing script. The new version numbers in an updated app are fine. However, if you are using the signing script to re-provision and re-sign an existing version of a wrapped app that is already in the app distribution library on MobileIron Core, you must increase the version numbers. MobileIron Core does not allow you to upload an app with the same version numbers as the version you already uploaded.

The signing script provides parameters for specifying new version numbers.

```
-s <CFBundleShortVersionString>
-d <CFBundleVersion>
```

For example:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -s "3.1.2" -d "3.1.2" myApp-
wrapped.ipa
```

Specifying a different bundle ID

You can use the -b option on a wrapped app to change its bundle ID. Because the -i option is required, you must also specify the signing certificate.

For example:

```
sign_wrapped_app.sh -i "iPhone Distribution:myCompanyName"
          -b "com.new.bundleID" myWrappedApp.ipa
```

Troubleshooting the signed wrapped app

If your signed wrapped app exits unexpectedly when you launch it, the issue sometimes involves the original app's bundle ID. In some cases, the sign_wrapped_app.sh script cannot infer the original app's bundle ID, which results in problems running the app.

To correct this issue, run the script using the -b option to specify the original app's bundle ID. For example:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -b com.myCompanyName.myApp
myApp-wrapped.ipa
```

Using the AppConnect App Wrapper (wrapping tool)

The AppConnect App Wrapper, also known as the wrapping tool, is an MacOS app that MobileIron provides. Its output is a wrapped, signed app. It also can output a wrapped, unsigned app, which you can give to another party to sign. You can run the wrapping tool using either its graphical user interface or using a command-line interface (CLI). The CLI is useful when you want to wrap an app from an automated script.

Important: You **cannot** wrap an app if you got the app (IPA file) from the Apple App Store.

Before you begin

Before you use the wrapping tool, be sure the unwrapped app installs and runs according to your requirements.

System requirements

The wrapping tool is supported only with the following versions of MacOS and Xcode listed in Product versions required.

Install Xcode

Install Xcode from the Apple App Store. After the installation completes, open Xcode and accept the license agreement. Then enter your administrator password for your MacOS system, if prompted for it.

Before you run the wrapping tool

Before you run the wrapping tool, make sure you have the following:

- Permission on your MacOS computer to allow apps that are not from the Mac App Store.
  On your computer, go to System Preferences > Security & Privacy > General. For the setting “Allow apps downloaded from”, select “Mac App Store and identified developers”.
- The signed, unwrapped IPA file for the app that you want to wrap.
  Make sure that you or another party signed the app according to Apple’s requirements.
- The signing identity, if you plan to re-sign the wrapped app now.
  You can distribute an app only if the app is signed. The wrapping tool gives you the option to sign the wrapped app. To use that option, make sure you have the signing identity that you want to use in the keychain. You can use the same signing identity as the unwrapped app or a different signing identity.
  If you are using the CLI, make sure that the keychain containing the signing identity is unlocked. See Unlock the keychain containing the signing identity.
  If you do not sign the app with the wrapping tool, you can sign the app later.
  MobileIron provides a signing script called sign_wrapped_app.sh for this purpose. Typically, you sign the app with the wrapping tool unless you do not have access to the signing identity. For example, only another person in your enterprise or at a third-party enterprise has the signing identity.
- The provisioning profile, if different from the unwrapped app’s provisioning profile.
  Typically, you re-use the provisioning profile from the unwrapped app. However, some reasons to replace it are that it has expired, or the signing identity is different from the unwrapped app. If necessary, install the new provisioning profile on the MacOS computer using Xcode.

To learn about signing identities and provisioning profiles, see Apple documentation at http://developer.apple.com.

Get the wrapping tool and signing script

The wrapping tool and signing script are located at:

https://support.mobileiron.com/mi/appconnect_app_wrapper/iOS/current

The wrapping tool is in AppConnectAppWrapper-<*version number*>.zip. Download the zip file to an MacOS computer and unzip the file.

The signing script is called sign_wrapped_app.sh.

Unlock the keychain containing the signing identity

If you plan to sign the wrapped app using the wrapping tool CLI, first make sure the keychain containing the signing identity is unlocked.

MacOS computers store signing identities and other security related items in keychains. A keychain is a password-protected file with the .keychain extension. It must be added to the keychain system on the MacOS computer before a program can use it. A keychain is either locked or unlocked. When locked, you cannot sign an app with a signing identity in the keychain unless you provide the keychain’s password.

Therefore, before running the wrapping tool CLI, unlock the keychain if the signing identity is in a locked keychain. If you do not, MacOS prompts you for the password in a dialog box. This graphical prompt occurs when running the wrapping tool with either the graphical user interface or CLI. **When using the CLI from an automated script, avoiding this prompt is critical to the script's success**.

The MacOS KeyChain Access application and the command-line program `security` allow you to view and manipulate keychains and their contents.

Use the following command-line command to unlock the keychain:

```
security unlock-keychain -p <password> <path to .keychain file>
```

Important: Signing identities are security sensitive because they can be used to sign malicious apps that appear to be your legitimate app. Take appropriate measures to secure the keychain that contains the signing identity and to secure the keychain's password.

Run the wrapping tool using its graphical user interface

Run the wrapping tool, following the instructions in the tool's graphical user interface.

Alternatively, you can run the wrapping tool using a command-line interface from a Terminal window or a script. For details, see Run the wrapping tool using a command-line interface.

Run the separate signing script, if necessary

The wrapping tool gives the option to sign the wrapped app. If you do not sign the wrapped app with the wrapping tool, sign it using the MobileIron-provided script called sign_wrapped_app.sh. This step is necessary if the person using the wrapping tool does not have access to the signing identity for the app.

Instructions for using sign_wrapped_app.sh are in Re-signing the wrapped app.

Run the wrapping tool using a command-line interface

You can run the wrapping tool using a command-line interface (CLI) from a Terminal window or a script. You provide command-line arguments instead of following a graphical user interface. Running the wrapping tool using the command-line interface is useful, for example, to automate wrapping as part of your regular app build process.

Since MacOS apps comprise a directory of resources and executables, the wrapping tool app contains its command-line executable in the following directory:

```
<wrapping tool installation directory>/AppConnect\ App\ Wrapper.app/Contents/MacOS/
```

The executable is `AppConnect\ App\ Wrapper.`

Therefore, to run the wrapping tool using the CLI, use the following command:

```
<wrapping tool installation directory>/AppConnect\ App\ Wrapper.app/ \
 Contents/MacOS/AppConnect\ App\ Wrapper <command-line arguments>
```

Command-line argument usage

Running the wrapping tool from the command-line requires that you enter command-line arguments. Some of the arguments are the options that determine the action that the wrapping tool takes. Include exactly one of the following options:
`-w, -i, -f, -v, -h`.

The following table shows the command-line argument usage.

TABLE 2. COMMAND-LINE ARGUMENT USAGE

Command-line usage	Purpose
`./AppConnect\ App\ Wrapper --nogui -w [-s identity] \ [-p profile] [-o outputPath] [--quiet] <app to wrap>`	Wraps an app.
`./AppConnect\ App\ Wrapper --nogui -i [--verbose|--quiet]`	Lists the names of available signing identities.
`./AppConnect\ App\ Wrapper --nogui -f [--verbose|--quiet]`	Lists the names of available provisioning profiles.
`./AppConnect\ App\ Wrapper --nogui -v`	Displays the version number of the wrapping tool.
`./AppConnect\ App\ Wrapper --nogui -h`	Displays the command-line usage.

NOTE: The order of the options does not matter. For example, `--nogui` can appear before or after `-w`.

Command-line argument descriptions

The following table describes the command-line arguments.

Note The Following:

- For the options `-w, -i, -f, -v,` and `-h`, you can also use the corresponding long name: `--wrap, --listidentities, --listprofiles, --verbose,` and `--help`.
- All long names must begin with a double dash whereas the one character options begin with a single dash.

TABLE 3. COMMAND-LINE ARGUMENTS

Argument	Description
`<app to wrap>`	Specifies the absolute or relative path to the IPA file to wrap. Use this argument only when you specify the `--wrap` argument.
`--nogui`	Suppresses the graphical user interface. If you do not include `--nogui`: • the wrapping tool ignores all other command-line arguments • the wrapping tool graphical user interface launches
`--verbose`	Displays more detailed information as follows: • With `--listidentities`, displays the expiration date of each signing identity. • With `--listprofiles`, displays the expiration date and bundle ID matching rule for each provisioning profile.
`--quiet`	Does not display command-line help text when an error occurs. Without `--quiet`, when an error occurs, the output displays the error information following by the `--help` text.

`--wrap` or `-w`	Wraps the app.
	When you use this argument, you can also specify these arguments:
	`--sign`
	`--provision`
	`--output`
	`<app to wrap>`

`--listidentities` or `-i`	Lists the names of the available signing identities that you can use with the `--sign` option. Use the `--verbose` option to list more detailed information.
`--listprofiles` or `-f`	Lists the names of the available provisioning profiles that you can use with the `--provision` option. Use the `--verbose` option to list more detailed information.
`--version` or `-v`	Displays the version number of the wrapping tool.
`--help` or `-h`	Displays help text describing the command-line arguments.

TABLE 3. COMMAND-LINE ARGUMENTS (CONT.)

Argument	Description

`--sign <identity name>`
or
`-s <identity name>`
	Signs the wrapped app with the specified signing identity. Use this argument only when you specify the `--wrap` argument.
	If you do not include the `--sign` argument, the app is wrapped but not signed.

`--provision <profile name>`
or
`-p <profile name>`
	Provisions the wrapped app with the specified provisioning profile. If multiple provisioning profiles have the specified name, the wrapping tool uses the profile with the most recent creation date.
	Use this argument only when you specify the `--wrap` argument.
	Make sure that the provisioning profile:
	• embeds the signing identity that you chose in the `--sign` argument.
	• contains a bundle ID search string that matches the bundle ID of the app.
	If you do not include the `--provision` argument, the wrapping tool uses the unwrapped app's provisioning profile.

`--output <output file>`
or
`-o <output file>`
	Saves the wrapped app to the specified file. Specify the absolute or relative path to the file. Use this argument only when you specify the `--wrap` argument.
	If you do not include the `--output` argument, the wrapping tool:
	• puts the wrapped app in the same directory as the unwrapped app.
	• appends `" Wrapped"` to the unwrapped app's file name. For example, wrapping `MyApp.ipa` creates `MyApp Wrapped.ipa`.

Command-line usage examples

The following examples demonstrate command-line usage. Each example assumes that the current directory is:

```
<wrapping tool installation directory>/AppConnect\ App\ Wrapper.app/Contents/MacOS
```

TABLE 4. COMMAND-LINE USAGE EXAMPLES

Example	Wrapping tool's actions
`./AppConnect\ App\ Wrapper --nogui -w \` `-o ~/MyApp_Wrapped.ipa ~/MyApp.ipa`	• Wraps the app `~/MyApp.ipa`. • Does not sign the wrapped app. • Uses the unwrapped app's provisioning profile for the wrapped app. • Puts the wrapped app in `~/MyApp_Wrapped.ipa`

Using short option names:
`./AppConnect\ App\ Wrapper --nogui -w \`
`-s "iPhone Distribution: MyCompany" \`
`-p "Wildcard Distribution" \`
`-o ~/MyApp_Wrapped.ipa ~/MyApp.ipa`
 	
Using long option names:
`./AppConnect\ App\ Wrapper --nogui --wrap \`
`--sign "iPhone Distribution: MyCompany" \`
`--provision "Wildcard Distribution" \`
`--output ~/MyApp_Wrapped.ipa ~/MyApp.ipa`
	• Wraps the app `~/MyApp.ipa`.
	• Signs the wrapped app with the specified signing identity.
	• Provisions the wrapped app with the specified provisioning profile.
	• Puts the resulting wrapped app in `~/MyApp_Wrapped.ipa`.

`./AppConnect\ App\ Wrapper --nogui -i --verbose`	Lists the name and expiration date of each available signing identity.
`./AppConnect\ App\ Wrapper --nogui -f --verbose`	Lists the name, expiration date, and bundle ID matching rule of each available provisioning profile.
`./AppConnect\ App\ Wrapper --nogui -v`	Displays the version number of the wrapping tool.
`./AppConnect\ App\ Wrapper --nogui -h`	Displays the command-line usage.

Command-line command exit status

If the command-line command is successful, the command sets the exit status to 0.

If it is unsuccessful, it does the following:

- Sets the exit status to 1.
- Outputs a detailed error message and error code to stderr.
- For appropriate cases, outputs the command-line usage to stderr.

AppConnect for iOS Wrapper Library and Wrapping Tool revision history

AppConnect 4.6.0 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

New features summary

This release includes the following new features and enhancements:

- **Support for UIScene**: Apps using UIScene are supported. As a result, the previous known issue APG-1154 is resolved.

AppConnect 4.5.3 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

- Resolved issues

Resolved issues

This release provides the following new resolved issues in the wrapper:

- APG-1177: Fixed an issue where redirected server requests could fail to connect.

AppConnect 4.5.2 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

- Resolved issues

Resolved issues

This release provides the following new resolved issues in the wrapper:

- APG-1171: Fixed an AppConnect startup issue in Wrapped apps. The issue is seen after updating to new versions of Mobile@Work, MobileIron Go, or MobileIron AppStation.
  For information on the Mobile@Work, MobileIron Go, and MobileIron AppStation versions that are affected, see AppConnect for iOS: Mandatory Updates for Client App Compatibility on the MobileIron Support Community.

AppConnect 4.5.1 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following.

Resolved issues

This release provides the following new resolved issues in the wrapper:

- APG-1162: Resolved an issue where NSURLSession delegate methods in Swift were sometimes not called.

AppConnect 4.5.0 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

Resolved issues

This release provides the following new resolved issues:

- AP-5256: Workaround for a bug in a third-party app security framework, which caused a crash when used with AppConnect.
- AP-5241: Fixed crash in [ACAppInterfaceBus displayMessage:scheme:completion:].
- AP-5199: Sometimes AppConnect apps failed to unlock using biometric authentication if the device passcode was set as the fallback option. Users may have seen this issues if the Check-in interval and the AutoLock interval are small and equivalent. This issue is fixed.
- AP-5245: Fixed a Secure File I/O thread-safety issue which could cause I/O errors when writing to multiple files simultaneously. Note that I/O to individual files should always be done from a single thread.
- AP-5253: Fixed an exception when launching apps in Xcode's Simulator.

Known issues

This release includes the following new known issues:

- APG-1154: UIScene apps, introduced in iOS 13, are not supported. The application lifecycle delegate methods are not called, so AppConnect is never initialized.
- AP-5252: Web@Work 2.9.0.0 for iOS with Chromium does not trust some sites. For more information, see the following Knowledge Base article in the MobileIron Community: Web@Work - Certain sites may not be trusted when using Chromium engine.

Limitations

This release includes the following new limitations:

- APG-1151: AppConnect SDK is not compatible with Xamarin.Forms Events on TextFields.

  Workaround: Add the following call in the AppDelegate's `FinishedLaunching() function: UIApplication.CheckForEventAndDelegateMismatches = false;`

AppConnect 4.4.1 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

- Resolved issues
- Known issues

Resolved issues

This release includes the following new resolved issues:

- AP-5233: Under certain conditions when adding cookies to a network request, the cookies were dropped after receiving an HTTP 302 redirect. This issue is fixed.
- APG-1148: In 4.4.0, if UIDocumentPicker / UIDocumentPickerViewController was initialized for Open From before AppConnect was ready, the AppConnect wrapped app crashed on iOS 12 devices. On iOS 13 devices, the Open From DLP was ignored due to underlying changes in iOS. With 4.4.1, the Open From DLP is ignored on all iOS versions for consistency and to avoid any crashes. Using UIDocumentPicker for Open From now behaves as if the app were unwrapped.

Known issues

This release includes the following new known issues:

- APG-1154: UIScene apps, introduced in iOS 13, are not supported. The application lifecycle delegate methods are not called, so AppConnect is never initialized.

AppConnect 4.4.0 for iOS Wrapper Library and Wrapping Tool revision history

This release provides the following:

New features summary

This release includes the following new features and enhancements:

- **Support for iOS 13**: AppConnect apps work as expected on iOS 13 devices.
- **armv7s architecture**: Support for the armv7s architecture has been dropped.

Resolved issues

This release provides the following new resolved issues:

- AP-5158: iOS 13 changed the identification for iPad devices. If your iPad is upgraded to iOS 13, MobileIron recommends that you also upgrade to MobileIron Core to one of the following patch releases: 10.2.0.2, 10.3.0.2, or 10.4.0.1. These patches contain the fixes for the changes in iOS 13 for iPad identification.
- AP-5179: On devices running iOS 13, openURL does not return the bundle ID of the calling app if the team ID is not the same. This issue is fixed with AppConnect 4.4.0 for iOS. To address the issue, update to AppConnect 4.4.0.
- AP-5201: Previously, the NSProxy instance proxying application delegate did not receive application lifecycle callbacks. This issue is fixed.
- AP-5207: On devices running iOS 13, AppConnect apps can Open files to other apps when Open In is disabled. This issue is fixed with AppConnect 4.4.0 for iOS. To address the issue, update to AppConnect 4.4.0.
- AP-5166: On devices running iOS 13, NSURLSession failed. This issue is fixed with AppConnect 4.4.0 for iOS. To address the issue, update to AppConnect 4.4.0.
- AP-5169: On devices running iOS 13, Email+ for iOS displayed a black background in app switcher. This issue is fixed with AppConnect 4.4.0 for iOS. To address the issue, update to AppConnect 4.4.0.
- AP-5174: Fixed the root cause due to which Email+ for iOS crashed intermittent.
- AP-5206: Previously, the AppConnect for iOS SDK was not calling applicationDidBecomeActive. This issue is fixed.

Limitations

This release includes the following new limitations:

- AP-5186: The openURL API in iOS 13 provides the bundle ID of the calling app only if the calling app has the same team ID. Due to this limitation, the Open From feature does not work on iOS 13 devices.
- AP-5164: Sharing files with the Chrome extension if Open In is restricted may cause the application to freeze.
- AP-5159: On devices running iOS 13, the "Unable to Share Document with selected application" prompt is not shown unless the Share dialog is closed.

AppConnect 4.3.1 for iOS Wrapper Library and Wrapping Tool revision history

This release does not provide any new features.

Support for the armv7s architecture is deprecated.

Resolved issues

This release provides the following new resolved issue:

- APG-1132: Fixed a potential crash in the NSURLSession delegate_task:didCompleteWithError: method.

AppConnect 4.3.0 for iOS Wrapper Library and Wrapping Tool revision history

New features

- **Support for MobileIron AppStation**
  Apps wrapped with the AppConnect 4.3.0 for iOS wrapper can run with MobileIron AppStation as the MobileIron client app instead of MobileIron Go. Administrators can use MobileIron AppStation on devices which are interacting with a MobileIron Cloud tenant that supports Mobile Apps Management (MAM) but not Mobile Device Management (MDM).
- **Support for Open From data loss prevention policy**
  The AppConnect 4.3.0 for iOS AppConnect library adds support for the Open From data loss protection policy.

At the date of this AppConnect release, no MobileIron servers support this policy.

- **iOS 9 no longer supported**
  AppConnect 4.3.0 for iOS is not supported on iOS 9 devices.
  See Product versions required.

Resolved issues

- **APG-1124:**An issue has been fixed when using the -b option with the sign_wrapped_app.sh script. Now you can use the -b option on a wrapped app to change its bundle ID and the app will run successfully.
  Because the -i option is required, you must also specify the signing certificate. For example:

```
sign_wrapped_app.sh -i "iPhone Distribution:myCompanyName"
          -b "com.new.bundleID" myWrappedApp.ipa
```

AppConnect 4.2.1 for iOS Wrapper Library and Wrapping Tool revision history

New features

- **Allow AppConnect apps to send custom cookies in web requests**

Some web pages inject custom cookies into web requests. For example, when an end user taps on a link in a web page, the page's JavaScript injects a custom cookie. If a user makes such a request from a web page displayed in an AppConnect app, by default AppConnect does not include the injected cookies in the web request, which can cause the request to fail. AppConnect now includes the custom cookies in the request if the MobileIron server administrator includes the following key in the app's app-specific configuration on the MobileIron server: `MI_AC_USE_ORIGINAL_COOKIES_FOR_DOMAINS`. The value of the key is a comma-separated string listing the domains for which the custom cookies should be included. Make sure no spaces are included in the value.

For example:

```
www.somewebsite.com,somename.someotherwebsite.com
```

Resolved issues

- APG-1121: Wrapped apps using Firebase no longer fail to launch.

Known issues

- APG-1124: If you change the bundle ID of a wrapped app, re-signing the wrapped app appears to succeed. However, when you launch the re-signed wrapped app, control switches to Mobile@Work but control does not return to the app.

  Workaround: Re-sign the unwrapped app but in addition to the usual -i option for the signing identity, add the -b option to specify the new bundle ID. For example:

```
$./sign_wrapped_app.sh -i "iPhone Distribution: myCompanyName" -b com.myCompanyName.myApp
myApp.ipa
```

  Then wrap the resulting IPA file and sign it.

Limitations

- AP-5026: A Xamarin app crashes if it uses custom code to copy text rather than the native iOS copy functionality.

AppConnect 4.2 for iOS Wrapper Library and Wrapping Tool revision history

This release of the AppConnect for iOS Wrapper library and wrapping tool has no new features.

Resolved issues

- AP-4919: Fixed an issue that caused an AppConnect app to crash when it used the same object as a delegate for multiple UI elements.

Known issues

- AP-4940: The LookUp option in the iOS context menu allows data to be shared to non-AppConnect apps regardless of the **Open In** and **Copy/Paste To** data loss prevention policies.

AppConnect 4.1.1 for iOS Wrapper Library and Wrapping Tool revision history

This AppConnect release has no new features.

Resolved issues

- AP-4920: When an AppConnect's app upload request is redirected, the request failed when using AppTunnel. This issue has been fixed by converting the stream request to a body request when using AppTunnel. Note that you can override the conversion by adding a key-value pair to the app's AppConnect configuration. Add MI_AC_DISABLE_HTTP_STREAM_CONVERSION with the value Yes.
- APG-1118: Fixed an issue where apps subclassing NSProxy could crash on launch with the error `-[NSProxy doesNotRecognizeSelector:_ACDecoratorClass]`.
- APG-1097: Provides a workaround to a known bug in NSURLSession that sometimes causes the form body to be missing in connections in AppConnect apps when using AppTunnel.

Known issues

- AP-4919: If an AppConnect app uses the same object as a delegate for multiple UI elements, the app crashes.

AppConnect 4.1 for iOS Wrapper Library and Wrapping Tool revision history

New features

Certificate pinning support

This AppConnect release supports certificate pinning for AppConnect apps to heighten security for communication between AppConnect apps and enterprise servers or cloud services.

Using certificate pinning requires:

- Configuration on the MobileIron server.
  For MobileIron Core, see "Certificate pinning for AppConnect apps" in the MobileIron Core AppConnect and AppTunnel Guide.
- Mobile@Work 10.0.0.0 for iOS through the most recently released version as supported by MobileIron.

This feature requires no additional development in the app.

Lock AppConnect apps when screen is off

This AppConnect release supports automatically logging out device users from AppConnect apps when the device screen is turned off due to either inactivity or user action.

This feature requires:

- Configuration on the MobileIron server.
  For MobileIron Core, see “Configuring the AppConnect global policy” in the MobileIron Core AppConnect and AppTunnel Guide.
- Mobile@Work 10.0.0.0 for iOS through the most recently released version as supported by MobileIron.

This feature requires no additional development in the app.

Overriding the Open In Policy for openURL: with the mailto: scheme

This AppConnect release allows the MobileIron server administrator to override the Open In policy when the policy blocks the iOS native email app when the app calls `openURL:` with the `mailto:` scheme.

The AppConnect library overrides the Open In policy for native email if the MobileIron server administrator added the key MI_AC_DISABLE_SCHEME_BLOCKING with the value `true` to the app’s app-specific configuration.

This feature requires no additional development in the app.

Resolved issues

- APG-1110: Fixed a "failed to extract entitlements from binary" error when re-signing apps with the sign_wrapped_app.sh script. See https://community.mobileiron.com/docs/DOC-7921 for details.

AppConnect 4.0 for iOS Wrapper Library and Wrapping Tool revision history

New features

- iOS 8 no longer supported
- Swift support for callback methods
- Native email control using the Open In DLP policy
- App extension control using the Open In DLP policy
- Custom keyboard use controlled by MobileIron server
- Dictation with the native keyboard is not allowed
- Support for sending AppConnect logs from Mobile@Work
- Drag and Drop data loss prevention policy support
- Automatic policy status updates sent to MobileIron server
- Support for storing AppConnect library encryption keys in the Secure Enclave

iOS 8 no longer supported

AppConnect 4.0 for iOS is not supported on iOS 8 devices.

See Product versions required.

Swift support for callback methods

Previous versions of AppConnect for iOS included wrapping Swift apps. However, callback methods were supported only when written in Objective-C. Now they are also supported when written in Swift.

See AppConnect wrapper callback methods.

Native email control using the Open In DLP policy

The Open In Data Loss Prevention policy now includes controlling whether an app can share documents with the native iOS mail app. Opening a document with the native iOS mail app is allowed only if one of the following is true:

- Open In is allowed for all apps
- Open In is allowed for only whitelisted apps, and the native iOS mail app is in the whitelist. The whitelist must contain both of these bundle IDs: com.apple.UIKit.activity.Mail and com.apple.mobilemail.

App extension control using the Open In DLP policy

The Open in data loss protection policy now includes restricting access to the iOS extensions that apps provide. Specifically:

Open In DLP for host app (the app using the extension)	Extension behavior
All apps allowed	The host app can use any app's extension for Open In.
Only AppConnect apps allowed	The host app can use only extensions provided by AppConnect apps for Open In.
Whitelist	The host app can use only extensions of apps in the whitelist for Open In.

Custom keyboard use controlled by MobileIron server

Releases prior to the AppConnect 4.0 for iOS Wrapper library blocked the use of custom keyboards in wrapped AppConnect apps. This release changes that behavior. The MobileIron server can now control custom keyboard use by your AppConnect app. If the administrator does not configure this choice, your app can choose to reject custom keyboard use.

See Custom keyboard control.

Dictation with the native keyboard is not allowed

The AppConnect 4.0 for iOS Wrapper library blocks the use of dictation when using the native iOS keyboard. The Wrapper library also adds support for a key-value pair that the MobileIron server administrator can set on the app's configuration. The key is called MI_AC_WR_ALLOW_KEYBOARD_DICTATION. By default, the value is false, and dictation is not allowed. If the administrator sets the value to true, then wrapped AppConnect apps can use dictation with the native keyboard.

Support for sending AppConnect logs from Mobile@Work

AppConnect apps using AppConnect 4.0 for iOS support the feature in Mobile@Work for iOS that sends AppConnect logs to an email address of your choice, such as a company's helpdesk. This feature requires Mobile@Work 9.8 for iOS through the most recently released version as supported by MobileIron.

Mobile@Work displays the option to send logs on the app's status details screen, available in Mobile@Work at **Settings > Secure Apps > *<app name>*.** The option is at the bottom of the screen with this text: **Send *<app name>* Logs.**

The option is displayed only for apps AppConnect apps using AppConnect 4.0 for iOS. However, the displayed option is disabled if the app's AppConnect authorization status is not authorized.

When the option is displayed and enabled, tapping it brings up the list of apps able to share the log files, such as email apps, if you included the following key-value pair for the app in its AppConnect app configuration:

- **MI_AC_ENABLE_LOGGING_TO_FILE** set to **Yes**

For wrapped apps, the server administrator can also include the key **MI_AC_WR_ENABLE_LOG_CAPTURE** set to **Yes.** This key causes the app's logs to be included in the log files along with the logs from the AppConnect wrapper and AppConnect library.

Drag and Drop data loss prevention policy support

MobileIron server administrators can set a drag and drop policy for each AppConnect app. It specifies whether AppConnect apps can drag content to all other apps, to only other AppConnect apps, or not at all. The AppConnect library enforces this policy.

NOTE: This feature is not supported with MobileIron Cloud.

Automatic policy status updates sent to MobileIron server

The AppConnect library now automatically sends a status update to the MobileIron server when it receives the following changes:

Change	Status update that AppConnect library sends to MobileIron server
Open In policy	Informs server that the policy change has been applied.
Pasteboard policy	Informs server that the policy change has been applied.
Print policy	Informs server that the policy change has been passed to the app.
Configuration values	Informs server that the configuration change has been passed to the app.
Authentication status	Informs server that the authentication change has been passed to the app.

Support for storing AppConnect library encryption keys in the Secure Enclave

For heightened security of the encryption keys that the AppConnect library uses, a MobileIron server administrator can now specify that the keys are stored in the Apple hardware known as the Secure Enclave. By using the Secure Enclave, the encryption key's attack surface is reduced, because the keys are stored in the Secure Enclave rather than in memory. The MobileIron server administrator uses the key named MI_AC_CONTAINER_TYPE with the value ENCLAVE in the app's app configuration. The AppConnect library consumes this key. It is not passed to your app in Its configuration key-value pairs.

To benefit from this feature, the device must:

- have Apple's Secure Enclave hardware.

NOTE: Devices that have biometric security have Secure Enclave hardware.

- be running iOS 11 through the most recently released version as supported by MobileIron
- be running Mobile@Work 9.8 for iOS through the most recently released version as supported by MobileIron

NOTE: MobileIron Go does not support this feature.

Resolved issues

- APG-1081: Fixed an issue where the AppConnect library failed to initialize in wrapped apps after changing the iOS **Text Size** setting.
- AP-4202: Custom protocol classes set to NSURLSessionConfiguration were previously ignored in AppConnect apps. This issue has been fixed.
- AP-4133: Added ability to use NSURLConnection with NSURLSession networking with AppTunnel.

Known issues

- AP-4657: The "unauthorized message" screen is blurred. It continues to be blurred until the next time the app switches to the MobileIron client app. After the next AppConnect checkin, the screen is no longer blurred.

Limitations

- AP-4720: On some devices, screen blurring does not occur when going to the Task Switcher.

AppConnect 3.5 for iOS Wrapper Library and Wrapping Tool revision history

New features

iOS 11 compatibility

This version of the AppConnect for iOS Wrapper Library is compatible with devices running iOS 11 Beta 7. At the time of this AppConnect release, the GA version of iOS 11 is not available.

IMPORTANT: Re-wrap your app to use the AppConnect 3.5 wrapper for your app to run on iOS 11 devices. Apps wrapped with wrapper versions prior to 3.1.3 crash on iOS 11 devices. Apps wrapped with wrapper version 3.1.3 do not crash, but the AppConnect library does not handle the pasteboard data loss prevention policy correctly.

For more information, see Product versions required.

Callback method involving network requests with AppTunnel

A callback method `-appConnectStateChangedTo:` is now available. It is a convenient way to make sure your app does not make network requests that depend on AppTunnel until the AppConnect library in the app has received the AppTunnel rules.

For more information, see Callback method involving network requests with AppTunnel.

Open In changes

- The AppConnect for iOS Wrapper Library supports a new key-value pair from the MobileIron server that tells the library not to enforce the Open In policy.
  See “Overriding the Open In Policy for the app” in the *MobileIron Core MobileIron Core AppConnect and AppTunnel Guide*.
- Open In behavior in wrapped app is different in iOS versions prior to iOS 11 than In iOS 11 through the most recently released version as supported by MobileIron. In iOS 11, regardless of the Open In policy, iOS always displays all apps that support the document type as possible target apps. However, if a user taps on an app that is not allowed based on the Open In policy, nothing happens. On iOS versions prior to iOS 11, only allowed apps are displayed. The iOS behavioral change impacts all wrapped apps, regardless what version of the wrapper they are wrapped with.
  See Data loss prevention policies.

Resolved issues

- APG-977: Sometimes AppConnect wrapped apps that used third-party SDKs crashed. The issue has been fixed.
- AP-4145: URL requests made on a background thread were not tunneled if the AppConnect library in the app had not received the AppTunnel rules. The issue has been fixed because the AppConnect library now blocks URL requests until after it has received the AppTunnel rules.

- AP-3917: When a URL request using NTLM authentication was tunneled with AppTunnel, an error occurred when the device user was prompted with the user credentials dialog. The dialog displayed the Standalone Sentry host name instead of the URL request's host name. The issue has been fixed.

Limitations

- AP-4302: Apps that use UIDocumentInteractionController's preview API will not be able to share documents with other apps, because iOS 11 beta 6 and 7 allow sharing only with certain built-in extensions.

AppConnect 3.1.3 for iOS Wrapper Library and Wrapping Tool revision history

This release has no new features.

Resolved issues

- AP-4054: The HTTP error code 403 was not always reported to apps using AppTunnel. This issue has been fixed.
- AP-4149: In some cases, enterprises that used both AppTunnel and a global HTTP proxy policy resulted in AppConnect apps having no access to the network. The issue occurred when an AppTunnel rule caused a tunneling attempt for requests to the URL for the proxy auto-configuration (PAC) file. The issue occurred for all AppTunnel rules that did one of the following:
  - used a wildcard character in the AppTunnel rule's hostname such that the PAC file URL matched the rule
  - explicitly named the PAC file URL in the AppTunnel rule's hostname

  To fix the issue, the AppConnect library now supports a new key-value pair in the AppConnect app configuration for an AppConnect app:
  - key name: `global_http_proxy_url`
  - value: the URL of the PAC file, which the Core administrator also enters into the Proxy PAC URL field of the global HTTP proxy policy.
    Example: http://pac.myproxy.mycompany.com

  The AppConnect library does not attempt to tunnel the specified URL, which results in successful use of both AppTunnel and the global HTTP proxy policy,

NOTE: An AppConnect app does not receive this key-value pair. It is consumed by the AppConnect library.

- AP-4152: This issue fixes a crash of AppConnect apps on iOS 11 Beta 1. However, this release does not support iOS 11.

AppConnect 3.1.2 for iOS Wrapper Library and Wrapping Tool revision history

This release has no new features.

Resolved issues

- AP-4062: Fixed a critical issue that caused an AppConnect app to crash if all of the following are true:
  - The app uses AppTunnel with either HTTP/S tunneling or TCP tunneling.

- The AppConnect log level “Debug” is activated for the app.
- The device is registered with MobileIron Core 9.4.0.0.

AppConnect 3.1.1 for iOS Wrapper Library and Wrapping Tool revision history

This release has no new features.

Resolved issues

- AP-3996: Renamed an AppConnect library internal class (PasteboardManager) to avoid naming conflicts.

AppConnect 3.1 for iOS Wrapper Library and Wrapping Tool revision history

New features

Update to OpenSSL 1.0.2h

The AppConnect library now uses OpenSSL version 1.0.2h.

Resolved issues

- AP-3721: Fixed an AppTunnel issue when using the iOS Social framework's SLRequest class.
- AP-3698: Fixed an issue that caused an AppConnect app to crash if the app used a custom protocol handler with NSURLSession (such as when the Layer SDK uses the SPDY protocol).
  Note that although the app no longer crashes, the custom protocol request might fail if the request is tunneled using AppTunnel.
- AP-3674: Fixed an issue where AppConnect apps inadvertently shared encrypted data with other iOS 10 devices on the same iCloud account.
- AP-3616: Fixed an issue where using the following iOS API caused an AppConnect app to crash.

```
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url
options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options;
```

Known issues

- AP-3958: When you copy content from an AppConnect app, pasting from the Universal Clipboard onto another device sometimes does not work.

Limitations

- AP-3711: A black screen is shown when flipping from the MobileIron client app to an AppConnect app on devices running all versions of iOS 8. This is an Apple issue.

AppConnect 3.0 for iOS Wrapper Library and Wrapping Tool revision history

This release has no new features. It fixes miscellaneous bugs.

Resolved issues

- APG-959: After wrapping an app that had no app entitlements, signing the app failed. The issue has been fixed.

Known issues

- AP-3616: Using the following iOS API causes an AppConnect app to crash:

```
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url
options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options;
```

- AWE-685: As issue occurs if a wrapped app running on a iOS 10 device uses the iOS API QLPreviewController to write to the pasteboard. If the AppConnect pasteboard policy does not allow an AppConnect app to write to the pasteboard, using this iOS API still results in data being written to the pasteboard, and any other app can paste the data.

Releases prior to AppConnect 3.0 for iOS Wrapper library and Wrapping Tool revision history

For the revision history of releases prior to AppConnect 3.0 for iOS Wrapper library and wrapping tool, see the"MobileIron AppConnect 4.2 for iOS App Wrapping Developers Guide", available on https://community.mobileiron.com.