MobileIron®

MobileIron® AppConnect for iOS Cordova Plugin Developers Guide

AppConnect for iOS SDK Version 2.1
AppConnect for iOS Cordova Plugin Version 2.1

March 23, 2015

Contents

Chapter 1

Introducing the MobileIron AppConnect for iOS Cordova Plugin

AppConnect for iOS overview

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 the following ways:

- Wrapping the app

  The MobileIron AppConnect wrapping technology creates a secure app without any further app development. You can wrap Cordova (or PhoneGap) apps.

  See the *AppConnect for iOS App Wrapping Developers Guide*.
- 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. The AppConnect for iOS SDK is for iOS native development.

  See the *AppConnect for iOS SDK Developers Guide*.
- Using the AppConnect for iOS Cordova Plugin (also called the AppConnect Cordova Plugin)

  A Cordova (or PhoneGap) app developer uses the plugin to create a secure Cordova app, or turn an existing Cordova app into a secure app.

  The AppConnect Cordova Plugin is described in this document.

Important: Before using the AppConnect Cordova Plugin, determine whether wrapping the app meets your needs. See *Choosing Wrapping or SDK Development to Create AppConnect for iOS Apps*.

Note:

- If your app uses an older version of the AppConnect Cordova Plugin, MobileIron recommends that you always rebuild your app with the current version of the plugin. Using the current version ensures the app contains all new features, improvements, and resolved issues.
- The referenced documents and updates to this document are on https://support.mobileiron.com/docs/current/appconnect/.
- Legal notices are on:

  https://support.mobileiron.com/copyrights/ACe

Secure app features

Secure enterprise apps that are built using the AppConnect Cordova Plugin can:

- Receive app-specific configuration information from MobileIron Core, which is the MobileIron server.

  This capability 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, and the app is secured from misuse due to configuration. This feature is also useful for apps which do not want to allow the device users to provide certain configuration settings for security reasons.
- 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 Core administrator decides whether an app can copy content to the iOS pasteboard, use the document interaction feature (Open In), or print. The app uses this information to limit its functionality to prevent data loss through these features.

AppConnect for iOS Cordova Plugin advantages

With the AppConnect for iOS Cordova Plugin:

- You can focus on application logic.

  The plugin handles low-level, complex work such as authentication to access AppConnect apps, certificate authentication to enterprise resources, tunneling, AppConnect passcode handling, data encryption, and getting app-specific settings and configuration from MobileIron Core.
- You use a set of simple APIs to develop a secure enterprise app.

  The app does not have to interact directly with web service interfaces to get the information it needs to behave as a secure enterprise app. Using the APIs, the app gets notified of any changes that the administrator makes on Core to controls and configuration.
- You can create one app, with one code base, that can behave as a secure app or a regular app. This behavior is required for secure apps that are distributed from the Apple App Store.

  For more information, see “Developing third-party dual-mode apps” on page 53.

32-bit and 64-bit app support

Using the AppConnect for iOS Cordova Plugin, you can build an app as a 32-bit app or as a 64-bit app. A 64-bit app takes advantage of the 64-bit processors available on some iOS devices.

MobileIron AppConnect components

The apps that you build with AppConnect for iOS Cordova Plugin work with the following MobileIron components:

MobileIron component	Description
MobileIron Core	The MobileIron 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.
Standalone Sentry	The MobileIron server which provides secure network traffic tunneling from your app to enterprise servers.
The Mobile@Work for IOS app	The MobileIron app that runs on an iOS device. It interacts with Core to get current security and management information for the device. It interacts with the AppConnect library to communicate necessary information to your app.
The AppConnect library	The MobileIron library that the plugin uses to get AppConnect information for your app. The AppConnect library is part of the AppConnect framework that the plugin includes.

Using a secure app

A device user can use a secure enterprise app only if:

- The device user has been authenticated through MobileIron Core.

  The user must use the Mobile@Work for iOS app to register the device with Core. Registration authenticates the device user.
- The Core administrator has authorized the device user to use the app.
- The device user has entered a secure apps passcode or Touch ID.

  The Core 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, 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.

App responsibilities

Your app is responsible for:

- enforcing the authorization settings
- enforcing the data loss prevention settings

- using the app-specific configuration

Mobile@Work and AppConnect library responsibilities

The Mobile@Work app and the AppConnect library are responsible for:

- authenticating the user to MobileIron Core
- authenticating to enterprise services using certificates
- tunneling network connections
- AppConnect passcode and Touch ID handling
- protecting AppConnect-related data, such as configurations and certificates

AppConnect Cordova Plugin contents

The AppConnect Cordova Plugin is included in the AppConnect for iOS SDK ZIP file called AppConnectiOSSDK_*<version number_xxx>*.zip, where xxx is the build number.

The AppConnect for iOS SDK ZIP file contains a directory called plugins/cordova. The plugins/cordova directory contains:

- AppConnectCordovaPlugin_*<version number_xxx>*.zip

  This ZIP file contains all the plugin files, which include:

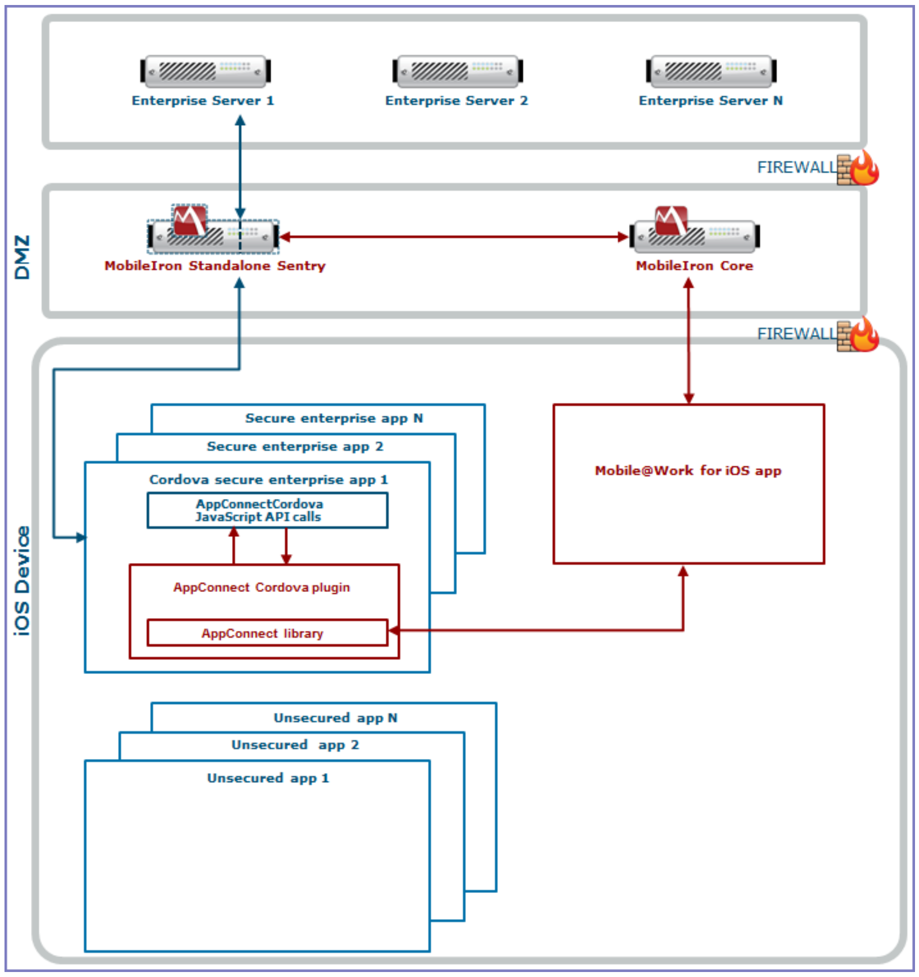
  - the AppConnect library files. The AppConnect library provides your app management and security capabilities. The AppConnect library facilitates communication between your app and the Mobile@Work for iOS app, which communicates with MobileIron Core.
  - AppConnectCordova.js which contains the JavaScript interfaces that your app uses to interact with the AppConnect library
- install_ac_cordova_plugin.sh, which is the script you use to install the AppConnect Cordova Plugin.
- TestAppConnect example in the Samples folder

  This sample Cordova app demonstrates how an app uses the AppConnect Cordova Plugin. It displays its authorization status, its app configuration, and its data loss prevention policies. Note that it only displays the data loss prevention policies and authorization status, but does not enforce them.
- Documentation folder, which contains this document

  Note: Check for updates to this document on https://support.mobileiron.com/docs/current/appconnect/.

AppConnect for iOS architecture

The following diagram illustrates how your app, using the AppConnect Cordova Plugin, interacts with the Mobile@Work for iOS app, MobileIron Core, and the Standalone Sentry.



Note the following:

- Each secure enterprise app communicates with an instance of the AppConnect library.
- The AppConnect library communicates with the Mobile@Work for iOS app.
- The app uses the AppConnect Cordova JavaScript API to get management and security-related information, such as whether the administrator has authorized the app to run on the device.
- The Mobile@Work app communicates with MobileIron Core to get management and security-related information.

  Core is the MobileIron 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.
- The AppConnect library interacts with a Standalone Sentry if it is tunneling network connections to an enterprise server behind the firewall.

Mobile@Work and AppConnect apps

Mobile@Work supports AppConnect apps, including the following tasks:

- It communicates with MobileIron Core to get management and security-related information and passes the information to the AppConnect apps.

  Mobile@Work periodically does an *app checkin* with Core to get this information. The administrator configures the app checkin interval in the AppConnect global policy. It is the maximum time between app checkins while an AppConnect app is running.
- It enforces the AppConnect passcode or Touch ID.

  Mobile@Work prompts the device user to create an AppConnect passcode or Touch ID when first launching any AppConnect app. You configure an auto-lock timeout in the AppConnect global policy. After this period of inactivity, Mobile@Work prompts the device user to reenter his AppConnect passcode or Touch ID.

When you run your AppConnect app, Mobile@Work sometimes automatically launches to support app checkin and the AppConnect passcode or Touch ID. Understanding Mobile@Work's expected behavior can help you when you test your AppConnect app.

App checkin and Mobile@Work

On each app checkin, Mobile@Work gets AppConnect policy updates for all the AppConnect apps that have already run on the device. These updates include changes to:

- the AppConnect global policy for the device.
- AppConnect container policies for each of the AppConnect apps that have run on the device.
- AppConnect app configurations for each of the AppConnect apps that have run on the device.
- the current authorization status for each of the AppConnect apps that have run on the device.

Mobile@Work does an app checkin in the following situations:

- The device user launches an AppConnect app for the first time.

  In this situation, Mobile@Work finds out about the app for the first time, and adds it to the set of AppConnect apps for which it gets updates.
- The app checkin interval expires while an AppConnect app is running.
- The app checkin interval expired while no AppConnect apps were running and then the device user launches an AppConnect app.

In each of these situations, Mobile@Work launches, and the device user sees the Mobile@Work app momentarily. Once Mobile@Work has completed the app checkin, the device user automatically returns to the AppConnect app.

The AppConnect passcode auto-lock time and Mobile@Work

Mobile@Work launches to prompt the device user for the AppConnect passcode or Touch ID in the following situations:

- The auto-lock (inactivity) timeout expires while the device is running an AppConnect app and the AppConnect passcode, not Touch ID, is the login mechanism.

  Note: If the device user is *interacting with* the app, the auto-lock time does not expire. This case occurs only when the device user has not touched the device for the duration of the timeout interval.
- The device user used Mobile@Work to log out of AppConnect apps, and then launches an AppConnect app.
- The Core administrator has changed the complexity rules of the AppConnect passcode, and an app checkin occurs.

In each of these situations, Mobile@Work launches, and presents the device user with a screen for entering his AppConnect passcode or Touch ID. After the device user enters the passcode or Touch ID, the device user automatically returns to the AppConnect app.

Product versions required

To develop and deploy an app that uses the AppConnect for iOS Cordova Plugin, you need certain products. MobileIron *supports* a set of product versions, and a larger set of product versions are *compatible* with apps built with this version of the AppConnect Cordova Plugin.

- 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:

Product	Supported versions	Compatible versions
iOS	6 through 8.2	
Cordova	4.0	3.5 through 4.3
MobileIron Core and Connected Cloud	7.1.1 through 7.5.1	5.7 through 7.1
Standalone Sentry	6.1	4.7 through 6.0
Mobile@Work for iOS	6.3	5.7.4 through 6.2

Note:

- Some features depend on the MobileIron Core, Standalone Sentry, and Mobile@Work versions. Only the newest versions support all features.
- MobileIron Cloud also supports AppConnect for iOS apps, but some AppConnect features are not supported.

Securing and managing the app using the AppConnect library

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

- "Authorization" on page 16
- "AppConnect passcode and Touch ID policy" on page 17
- "Configuration specific to the app" on page 18
- "AppTunnel" on page 18
- "Certificate authentication to enterprise services" on page 19
- "Data loss prevention policies" on page 19

Additionally, the AppConnect library uses encryption to protect AppConnect-related data, but this protection requires no additional Core configuration besides requiring an AppConnect passcode or device passcode.

The following steps show the flow of information from Core to your app:

1. The Core administrator decides which app-related settings to apply to a device or set of devices.
2. Core sends the information to the Mobile@Work app.
3. The Mobile@Work app passes the information to the AppConnect library, which is part of the AppConnect Cordova Plugin. The Mobile@Work app and the AppConnect library enforce the AppConnect passcode policy. The AppConnect library enforces tunneling.
4. Using the AppConnect for iOS Cordova Plugin APIs, your app can find out the current settings and receive events about changes.

Your app is responsible for:

- *enforcing authorization*
- *enforcing the data loss prevention policies*
- *using the configuration specific to the app.*

Authorization

The MobileIron Core administrator determines:

- whether or not each device user is authorized to use each secure enterprise app.

  If the user is not authorized, the app should not allow the user to access any secure data or functionality. If the app handles only secure data and functionality, then the app does nothing more than display a message that the user is not authorized to use the app.
- the situations that cause an authorized device user to become unauthorized.

These situations include, for example, when the device OS is compromised. Mobile@Work reports device information to Core. Core then determines whether to change the user to unauthorized based on security policies on Core.

When a user becomes unauthorized, the app should stop allowing the user access to any secure data or functionality.

- the situations that retire the app.

  Retiring an app means that the user is not authorized to use it, and the app removes all secure data associated with the app.

  Note: When an app is retired, you remove all its secure data. When a user is unauthorized but the app is not retired, you do not allow the user to access the data, but you do not have to remove it. The reason is that an unauthorized user can become authorized again, and therefore the secure data should become available again.

Your app uses the AppConnect for iOS Cordova Plugin to get the user's authorization status for using the app and to be notified of changes. For more information, see "Authorization API details" on page 36.

AppConnect passcode and Touch ID policy

The MobileIron Core administrator determines:

- whether the AppConnect passcode or Touch ID is required, which requires the device user to enter a passcode or Touch ID to access any secure enterprise apps.
- the complexity of the AppConnect passcode.
- the auto-lock time for the AppConnect passcode or Touch ID.

The AppConnect library and Mobile@Work enforce an AppConnect passcode as follows:

- Core notifies Mobile@Work when the Core administrator has enabled an AppConnect passcode or Touch ID. Mobile@Work prompts the user to set the passcode the next time that the device user launches or switches to a secure enterprise app.
- Mobile@Work prompts the user to enter the passcode or Touch ID when the user subsequently launches or switches to a secure enterprise app but the inactivity timeout has expired.
- Mobile@Work prompts the user to enter the passcode or Touch ID when the auto-lock time expires *while* the user is running a secure enterprise app.
- Mobile@Work prompts the user to set the passcode the next time the device user launches or switches to a secure enterprise app after Core has notified Mobile@Work that the passcode's complexity rules have changed.

Your app does not handle the AppConnect passcode or Touch ID at all. The AppConnect library and Mobile@Work enforce the passcode or Touch ID, and auto-lock time.

Configuration specific to the app

Sometimes an app requires app-specific configuration. Some 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

You determine the app-specific configuration that your app requires. Each configurable item is a key-value pair. Each key and value is a string. A MobileIron Core administrator specifies the key-value pairs for each app on Core. 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 Core 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, starting with Core 5.6, Core 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.

Your app uses the AppConnect for iOS Cordova Plugin to get the configuration and to be notified of changes. For more information, see "App-specific configuration API details" on page 40.

AppTunnel

Using MobileIron's AppTunnel feature, a secure enterprise app can securely tunnel HTTP and HTTPS network connections from the app to servers behind an organization's firewall. A Standalone Sentry is necessary to support AppTunnel. The MobileIron Core administrator handles all AppTunnel configuration on Core. Once the administrator has configured tunneling for the app on Core, the AppConnect library, Mobile@Work, and a Standalone Sentry handle tunneling for the app.

Your app typically does not take any special actions related to tunneling. Although your app uses a server address that results in tunneling, your app does not know that tunneling is occurring. Typically, the Core administrator uses AppConnect's app-specific configuration to specify the enterprise server URL that the app uses. See "Configuration specific to the app" on page 18.

If your app requires locally cached URL responses, it must take a special action. See "Caching tunneled URL responses" on page 52.

Supported APIs

AppTunnel supports typical JavaScript network APIs, such as XMLHttpRequest or jQuery calls. More generally, AppTunnel supports any network API that Cordova binds to the Objective-C APIs NSURLConnection or NSURLSession. If you use a Cordova plugin that uses other Objective-C APIs, including WKWebView, or accesses sockets directly, AppTunnel is not supported.

Advanced AppTunnel

Advanced AppTunnel provides support for a secure tunnel for TCP traffic. Both AppConnect apps and standard apps can use Advanced AppTunnel. The MobileIron Core administrator configures Advanced AppTunnel, including installing MobileIron Tunnel (an iOS app) on the device. *Your app takes no actions related to using Advanced AppTunnel.*

Certificate authentication to enterprise services

Without any development, 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 Core administrator configures on Core 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.

Your app takes no action at all.

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 Core administrator specifies on Core whether each app is allowed to use each of these features.

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. This policy does not impact whether your app is allowed to paste content *from* the pasteboard into your app.
- The Open In policy specifies the apps, if any, 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.

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

Your app uses the AppConnect for iOS Cordova Plugin to get the data loss prevention policies and to be notified of changes. For more information, see:

- "Pasteboard policy API details" on page 43
- "Open In policy API details" on page 46
- "Print policy API details" on page 49

Data protection

Mobile@Work and the AppConnect 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 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.

Your app does not handle data protection for AppConnect-related data. Mobile@Work and the AppConnect library provide this data protection.

Chapter 2

Getting started with the AppConnect for iOS Cordova Plugin

Getting started tasks

Before you begin

Be sure you have the required product versions. See “Product versions required” on page 15.

Getting started task list

Do the following tasks to add the AppConnect for iOS Cordova Plugin to your app:

1. “Run the AppConnect Cordova Plugin installation script” on page 22
2. “Initialize the AppConnect library” on page 22
3. “Wait for the AppConnect library to be ready” on page 23

Optionally, you can create an AppConnect.plist file. See “Specify app permissions and configuration in a plist file” on page 23.

Run the AppConnect Cordova Plugin installation script

The AppConnect Cordova Plugin installation script is called install_ac_cordova_plugin.sh. The script does the following:

- Installs the AppConnect Cordova Plugin into your Cordova app.
- Creates the iOS platform directory for your app if it was not already created.
- Modifies main.m in the iOS platform directory to include code that the AppConnect Cordova Plugin requires.

  Note: If you delete the iOS platform directory and re-create it without using the script, follow the instructions in “Code changes if you manually recreate the iOS platform directory” on page 26.

To run install_ac_cordova_plugin.sh:

1. Put the AppConnectCordovaPlugin_*<version number>*.zip and install_ac_cordova_plugin_sh files in a convenient directory.
2. Change to the top-level directory of your app’s Cordova project.

   This directory contains the subdirectories plugins, www, hooks, and platforms, and contains the project’s config.xml file.
3. Run the script.

   For example, if the plugin zip file and the script are also in the top-level directory:
   ```
   $./install_ac_cordova_plugin.sh -p AppConnectCordovaPlugin_V2_0_0_0.zip
   ```
   The -p option is the relative or absolute path of the plugin zip file.

Initialize the AppConnect library

To initialize the AppConnect library for your app to use, call the following method when your app receives the Cordova `'deviceready'` event:

```
AppConnectCordova.initialize();
```

After this step, the AppConnect library is initializing. However, the app cannot yet use the other AppConnect Cordova Plugin interfaces.

Wait for the AppConnect library to be ready

The AppConnect Cordova Plugin generates the `'appconnect.isReady'` event when the AppConnect library initialization has completed.

Do the following:

1. Add an event handler for the `'appconnect.isReady'` event as part of your app's initialization. For example:

   ```
   document.addEventListener('appconnect.isReady', this.onAppConnectIsReady, false);
   ```

2. Indicate in the user interface that the app is initializing if the app requires the AppConnectCordova JavaScript interfaces to determine what to do. For example, use an activity indicator (spinner).

   One reason this indication is important involves when to display sensitive data. Do not show any sensitive data until the AppConnect library is ready, because until that time, the app cannot determine whether it is authorized. Only an authorized app should show sensitive data.

3. In the event handler for the `'appconnect.isReady'` event:
   - Remove the activity indicator after the app receives the `'appconnect.isReady'` event.
   - Access other AppConnectCordova JavaScript interfaces such as `authState()`, `authMessage()`, and `config()` and take actions appropriate for your app.

   Note: Before accessing AppConnectCordova JavaScript interfaces other than `initialize()`, always check the `isReady()` method. Doing so allows you to use the same methods when the app first launches and throughout execution.

4. Handle events that the AppConnect Cordova Plugin generates.

   These events indicate changes to the authorization status, data loss prevention policies, and app-specific configuration. For details, see "AppConnect for iOS Cordova Plugin API" on page 29.

Specify app permissions and configuration in a plist file

If your app is an in-house app, you can specify default values for:

- the data loss protection policies, such as the Open In policy
- the key-value pairs for your app-specific configuration

Specifically, you can provide a special plist file called AppConnect.plist as part of your in-house app that:

- specifies whether your app should be allowed by default to copy to the iOS pasteboard, use document interaction (Open In), and print.
- specifies app-specific configuration keys and default values.

These default values are used by MobileIron Core to make it easier for the Core administrator to set up your app with the correct data loss protection policies and app-specific configurations. *Your app never reads the AppConnect.plist*.

When you include the AppConnect.plist in your app:

1. When an administrator uploads your in-house app to Core, Core uses this plist file to automatically create Core policies that contain your specified data loss protection policies and app-specific configuration.
2. The administrator can then edit these policies.

   For example:

   - If one of your app-specific configuration keys requires a URL of an enterprise server, the administrator provides that value.
   - If the administrator requires stricter data loss protection policies than your app's default values, the administrator changes the values.
3. The administrator then applies these policies to the appropriate set of devices.
4. When your app runs, it receives the data loss protection policies and app-specific configuration by using the AppConnect for iOS Cordova Plugin APIs, described in "AppConnect for iOS Cordova Plugin API" on page 29.

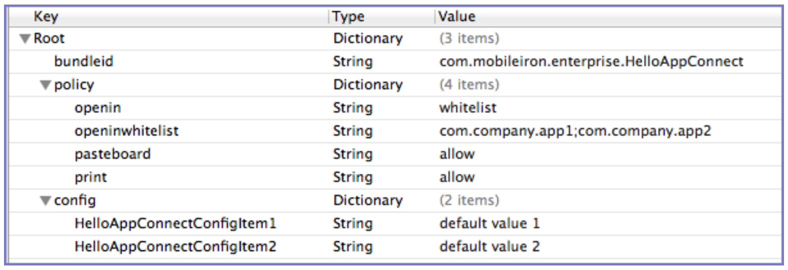
   For example, to handle app-specific configurations, you use the `AppConnectCordova.config()` method to get the key-value pairs.

If the administrator later changes the data loss protection policies or app-specific configuration, your app receives the updates by using the AppConnect for iOS Cordova Plugin APIs.

You can create an AppConnect.plist file using the Xcode project in your app's ios platform directory. For example:

```
$HOME/Hello/platforms/ios/HelloWorld/HelloWorld.xcodeproj
```

An example of an AppConnect.plist file as viewed in Xcode looks like the following:



Key	Type	Value
▼Root	Dictionary	(3 items)
bundleid	String	com.mobileiron.enterprise.HelloAppConnect
▼policy	Dictionary	(4 items)
openin	String	whitelist
openinwhitelist	String	com.company.app1;com.company.app2
pasteboard	String	allow
print	String	allow
▼config	Dictionary	(2 items)
HelloAppConnectConfigItem1	String	default value 1
HelloAppConnectConfigItem2	String	default value 2

To set up an AppConnect.plist file using Xcode:

1. Open the .xcodeproj file in Xcode.
2. Right-click on the project name, such as HelloWorld, in the left pane.
3. Select New File.
4. Select Resource.
5. Select Property List.
6. Click Next.

7. Save as AppConnect.plist.
8. In the Root key of AppConnect.plist, place a key called `bundleid` with the type String, and set the value to the bundle ID of your app.
9. In the Root key of AppConnect.plist, create two keys called `policy` and `config`, each with the type Dictionary.
10. In the `policy` dictionary, create keys called `openin`, `openinwhitelist`, `pasteboard`, and `print`, each with the type String.
11. Set these keys' values as given in the following table:

Key	Possible values and meanings
`openin`	• `allow` Document interaction is allowed with all other apps. • `disable` Document interaction is not allowed. • `whitelist` Only documents in the `openinwhitelist` list can open documents from your app. • `appconnect` Document interaction is allowed with all other AppConnect apps. **Note:** This value results in the app receiving a whitelist in the Open In policy API. The whitelist contains the list of all currently authorized AppConnect apps. You do not enter an `openinwhitelist` key in the plist. See "Open In policy API details" on page 46.
`openinwhitelist`	Semicolon separated list of the bundle IDs of the apps with which document interaction is allowed. This key is necessary when the `openin` key has the value `whitelist`.
`pasteboard`	• `allow` Pasteboard interaction is allowed with all other apps. That is, this option allows the device user to be able to copy content from your app to the iOS pasteboard. • `disable` Pasteboard interaction is not allowed.
`print`	• `allow` Printing is allowed. • `disable` Printing is not allowed.

12. In the `config` dictionary, create keys as required for your app.

13. Optionally, add values for the keys. The values must be String types.

    Note: The value `$USERID$` in the example tells Core to substitute the device user's user ID for the value. Other possible variables are `$EMAIL$` and `$PASSWORD$`. Depending on the Core configuration, custom variables called `$USER_CUSTOM1$` through `$USER_CUSTOM4$` are sometimes available.

Code changes if you manually recreate the iOS platform directory

The AppConnect Cordova Plugin installation script creates the iOS platform directory for your app if it was not already created. It also modifies main.m in the iOS platform directory to include code that the AppConnect Cordova Plugin requires.

If you delete the iOS platform directory and re-create it without using the script, edit main.m as follows:

1. Add the following line to the import statements:

   ```
   #import "AppConnect/AppConnect.h"
   ```

2. Change the third argument of the call to `UIApplicationMain()` to `kACUIApplicationClassName`.

   The third argument, the `principalClassName` argument, is the UIApplication class or subclass for the app. The modified statement in the sample app is:

   ```
   int retVal = UIApplicationMain(argc, argv, kACUIApplicationClassName,
                                   @"AppDelegate");
   ```

Troubleshooting

App crashes due to not waiting for AppConnect ready event

Problem

Your app crashes due to the following uncaught exception:

```
Function <function name> called before AppConnect is ready.
```

For example:

```
Function authState() called before AppConnect is ready.
```

The AppConnect library throws this exception if the app calls any of the following AppConnect Cordova Plugin methods before the AppConnect library is ready:

- `AppConnectCordova.managedPolicy()`
- `AppConnectCordova.authState()`
- `AppConnectCordova.authMessage()`
- `AppConnectCordova.pasteboardPolicy()`
- `AppConnectCordova.openInPolicy()`
- `AppConnectCordova.openInWhitelist()`
- `AppConnectCordova.printPolicy()`
- `AppConnectCordova.config()`

Solution

Refactor your code to make sure you check `AppConnectCordova.isReady()` before calling the listed methods. If `AppConnectCordova.isReady()` returns `true`, you can access the methods. If `AppConnectCordova.isReady()` returns `false`, wait for the AppConnect Cordova Plugin to generate the `'appconnect.isReady'` event before calling the methods.

See “AppConnect ready API details” on page 34.

Chapter 3

AppConnect for iOS Cordova Plugin API

AppConnect for iOS Cordova Plugin overview

The AppConnect for iOS Cordova Plugin provides these capabilities to your app:

- Initializing the AppConnect library
- Getting the user's authorization status, and receiving events about changes
- Getting app-specific configuration, and receiving events about changes
- Getting data loss prevention policies, and receiving events about changes
- Getting the version of the AppConnect library
- Allowing cached responses for URL requests that use AppTunnel

The AppConnect Cordova Plugin JavaScript interface is defined in AppConnectCordova.js. It defines enumerations and methods for interacting with the plugin. An app also interacts with the plugin by handling events that the plugin generates.

Dual-mode app capabilities

If your AppConnect app is distributed from the Apple App Store, due to Apple App Store requirements, your app is required to work as either:

- an AppConnect app for enterprise users
- a regular app for general consumers

Such an app is called a *dual-mode* app. Using one code base and APIs in the AppConnect for iOS Cordova Plugin, the app determines which way to behave. Depending on the app, the functionality available as a regular app can differ significantly from the functionality available as an AppConnect app.

AppConnect apps distributed from the Apple App Store must be dual-mode apps. If you are a third-party app developer, you typically build apps for Apple App Store distribution. If you are an in-house app developer, your apps are typically distributed from MobileIron Core.

For more information, see "Developing third-party dual-mode apps" on page 53.

The AppConnectCordova JavaScript interface

The AppConnectCordova.js file defines the AppConnect Cordova Plugin Javascript interface. This interface includes the method that you use to initialize the AppConnect library. For details, see "Initialize the AppConnect library" on page 22.

The AppConnectCordova interface also declares the methods that your app uses to interact with the AppConnect library. However, the app cannot interact with the AppConnect library until the AppConnect library has completed its initialization. For details about checking when the AppConnect library is ready, see:

- "Wait for the AppConnect library to be ready" on page 23
- "AppConnect ready API details" on page 34

For details of each of the AppConnectCordova interface’s methods, see:

- “AppConnect ready API details” on page 34
- “Authorization API details” on page 36
- “App-specific configuration API details” on page 40
- “Pasteboard policy API details” on page 43
- “Open In policy API details” on page 46
- “Print policy API details” on page 49
- “Getting the AppConnect library version” on page 51
- “Caching tunneled URL responses” on page 52

Note: The AppConnectCordova interface also provides methods specifically for dual-mode apps. These methods are described in “Developing third-party dual-mode apps” on page 53.

Event handling overview

The AppConnect Cordova Plugin generates events when it has new information to report to your app. Your app adds event listeners to its document object for these events, and provides event handlers.

Your app handles events about changes to:

- the ready status of the AppConnect library
- the user’s authorization status
- app-specific configuration
- data loss prevention policies

In the event handlers, your app:

1. Makes appropriate changes to its logic, display, and data.
2. In most cases, calls a method of the AppConnectCordova interface to inform the AppConnect library about its success or failure in making the changes.

AppConnect Cordova Plugin events

Add event listeners to the document object for these events:

Event name	Description
`'appconnect.isReady'`	The AppConnect library has finished initializing. Your app can now access the properties and methods of the AppConnect Cordova Plugin. Handling this event is required.
`'appconnect.authStateChangedTo'`	The authentication state has changed. Handling this event is required.

Event name	Description
`'appconnect.configChangedTo'`	The app-specific configuration settings have changed. Handling this event is optional. Handle it only if your app uses app-specific configuration.
`'appconnect.openInPolicyChangedTo'`	The Open In policy has changed. Handling this event is optional. Handle it only if your app uses the Open In feature.
`'appconnect.pasteboardPolicyChangedTo'`	The pasteboard policy has changed. Handling this event is optional. Handle it only if your app copies content *to* the pasteboard.
`'appconnect.printPolicyChangedTo'`	The print policy has changed. Handling this event is optional. Handle it only if your app is able to print.

For details about handling each event, see:

- "AppConnect ready API details" on page 34
- "Authorization API details" on page 36
- "App-specific configuration API details" on page 40
- "Pasteboard policy API details" on page 43
- "Open In policy API details" on page 46
- "Print policy API details" on page 49

Event handling acknowledgments

Your app must inform the AppConnect library of your app's success or failure in applying changes it receives in events. Depending on the type of event, your app calls one of the following methods of the AppConnectCordova interface:

```
AppConnectCordova.authStateApplied()

AppConnectCordova.configApplied()

AppConnectCordova.openInPolicyApplied()

AppConnectCordova.pasteboardPolicyApplied()

AppConnectCordova.printPolicyApplied()
```

Note: No event acknowledgment method exists for `'appconnect.isReady'`.

Each event acknowledgment method takes two parameters:

- an `AppConnectCordova.ACPolicyState` enumeration value:

```
AppConnectCordova.prototype.ACPolicyState = {
    UNSUPPORTED: 0, // The policy is not supported by this application
    APPLIED: 1,     // The policy was applied successfully
    ERROR: 2        // An error occurred applying the policy
}
```

Typically, you pass either `APPLIED` or `ERROR`. If you do not call the acknowledgment method for one of the optional events, the AppConnect Cordova Plugin behaves as if your app had called the method with `UNSUPPORTED`.

- a string value, which is a message explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the update. The string is reported in the MobileIron Core log files.

Note: Two other optional parameters are also in the function definition, but you should not pass these parameters.

AppConnect ready API details

The 'appConnect.isReady' event

The AppConnect library begins its initialization when your app calls `AppConnectCordova.initialize()`. The AppConnect Cordova Plugin generates the `'appconnect.isReady'` event when the AppConnect library has completed its initialization.

The isReady() method

The AppConnectCordova JavaScript interface provides an `isReady()` method:

```
AppConnectCordova.isReady()
```

This method indicates whether the AppConnect Cordova Plugin is ready for the app to access its other methods.

Return value: `true` or `false`

The app can access the other methods only if `AppConnectCordova.isReady()` returns `true`. If `AppConnectCordova.isReady()` returns `false`, an attempt to access another method causes the AppConnect library to throw an exception, and the app will crash.

When your app calls `AppConnectCordova.initialize()`, the AppConnect library begins its initialization, which concludes with the AppConnect Cordova Plugin generating the `'appconnect.isReady'` event. The return value of `AppConnectCordova.isReady()` is `false` until that time. Then it will return `true`. The value remains `true` for the life of the app.

Event handler for 'appConnect.isReady' event

You are required to add an event listener to your document object for the `'appconnect.isReady'` event. For example:

```
document.addEventListener('appconnect.isReady', this.onAppConnectIsReady, false);
```

The AppConnect Cordova Plugin generates the `'appconnect.isReady'` event one time when the AppConnect library initialization is complete. The app starts the initialization by calling `AppConnectCordova.initialize()`.

In your event handler:

- Update the app with the current authorization status, data loss protection policies, and configuration key-value pairs.

  The information is available by calling these AppConnectCordova methods:

  - `AppConnectCordova.authState()`
  - `AppConnectCordova.authMessage()`
  - `AppConnectCordova.pasteboardPolicy()`
  - `AppConnectCordova.openInPolicy()`
  - `AppConnectCordova.openInWhitelist()`

    - `AppConnectCordova.printPolicy()`
    - `AppConnectCordova.config()`
- Remove the user interface indication that informed the user that the app was initializing.

Note: Always update the app’s policies and configuration status in the event handler for the `'appconnect.isReady'` event. The AppConnect Cordova Plugin generates this event when the app is launched, after the AppConnect library finishes its initialization. The AppConnect Cordova Plugin generates other events, such as the events for authorization, data loss protection policies, and configuration, *only if* the status has changed. Therefore, you can always expect all these events on the first launch of the app. However, subsequent launches often result in the AppConnect Cordova Plugin generating only the `'appconnect.isReady'` event.

Authorization API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to handle the device user's authorization status for using the app. For an overview of this feature, see "Authorization" on page 16.

The ACAuthState enumeration

The `ACAuthState` enumeration provides the possible authorization statuses for the device user to use the application:

```
AppConnectCordova.prototype.ACAuthState = {
    UNAUTHORIZED: 0, // The user is not authorized to access sensitive
                     // data or views in this app.

    AUTHORIZED: 1,   // This is the only state in which the user is
                     // authorized to access sensitive data or views.

    RETIRED: 2       // The app must erase all sensitive data,
                     // including any stored authentication
                     // credential.
}
```

The authState() and authMessage() methods

The authState() method

The following method returns the current authorization status of the device user for using the app:

```
AppConnectCordova.authState()
```

Return value: An `AppConnectCordova.ACAuthState` enumeration value.

The authMessage() method

The following method returns a string value that indicates the reason for the current authorization status:

```
AppConnectCordova.authMessage()
```

Return value: A string explaining the current authorization status

Calling authState() and authMessage() when your app launches

When your app launches:

1. Upon receiving the Cordova `'deviceready'` event, call the method `AppConnectCordova.initialize()`.
2. Wait for the AppConnectCordova event `'appconnect.isReady'` before calling the `AppConnectCordova.authState()` or the `AppConnectCordova.authMessage()` methods.

3. While waiting, indicate in the user interface that the app is initializing if the app requires AppConnectCordova methods such as `AppConnectCordova.authState()` to determine what to do. For example, use an activity indicator (spinner).

   One reason this indication is important involves when to display sensitive data. Do not show any sensitive data until the AppConnect Cordova Plugin is ready, because until that time, the app cannot determine whether it is authorized. Only an authorized app should show sensitive data.

In the event handler for the `'appconnect.isReady'` event, check the return value of the `AppConnectCordova.authState()` method. Do the following:

1. Remove the indication that the app is initializing.
2. If the status is not the `AppConnectCordova.ACAuthState` value `AUTHORIZED`:
   - Do not allow the user to see or access sensitive data.
   - Display the string returned by the `AppConnectCordova.authMessage()` method.
3. If the status is the `AppConnectCordova.ACAuthState` value `AUTHORIZED`, allow the user to see and access sensitive data. Typically, the app does not display the `AppConnectCordova.authMessage()` string when the status is `AUTHORIZED`.

Method return values after updates to authorization status

On any updates to authorization status or message while the app is running, the AppConnect Cordova Plugin generates the `'appconnect.authStateChangedTo'` event. Subsequent calls to the `AppConnectCordova.authState()` and `AppConnectCordova.authMessage()` methods return the updated authorization status and message.

The 'appconnect.authStateChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.authStateChangedTo'` event when the authorization state or authorization message changes after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description
`newAuthState`	An `AppConnectCordova.ACAuthState` enumeration value
`authMessage`	A string explaining the new authorization status

Event handler for 'appConnect.authStateChangedTo' event

When a change has occurred to the user’s authorization status, the AppConnect Cordova Plugin:

1. Stores the new `AppConnectCordova.ACAuthState` value so that subsequent calls to the `AppConnectCordova.authState()` method returns the updated authorization status.
2. Stores the new string value explaining the new authorization status so that subsequent calls to the `AppConnectCordova.authMessage()` method return the updated authorization message.

3. Generates the `'appconnect.authStateChangedTo'` event.

You are required to add an event listener to your document object for the `'appconnect.authStateChangedTo'` event. For example:

```
document.addEventListener('appconnect.authStateChangedTo',
                          this.onAppConnectAuthStateChangedTo, false);
```

Your app handles the new status as follows:

New status	App actions
`UNAUTHORIZED`	• Exits any sensitive part of the application. • Stops allowing the user to access sensitive data and views. • Displays the message received in the event that explains the authorization status change. • Calls the `AppConnectCordova.authStateApplied()` method.
`AUTHORIZED`	• Allows the user to access sensitive data and views. • Calls the `AppConnectCordova.authStateApplied()` method.
`RETIRED`	• Exits any sensitive part of the application. • Deletes all sensitive data, including any stored authentication credentials, data in files, keychain items, pasteboard data, and any other persistent storage. • Displays the message received in the event that explains the authorization status change. • Calls the `AppConnectCordova.authStateApplied()` method.

Note: The AppConnect Cordova Plugin can generate the event when *only* the explanatory string, but not the authorization status, has changed. When the status is `UNAUTHORIZED` or `RETIRED`, the message typically contains a new reason for the status. Display the new message.

The authStateApplied() method

After your event handler processes the information provided in the `'appconnect.authStateChangedTo'` event, it must call this acknowledgment method:

```
AppConnectCordova.authStateApplied(policyState, message)
```

Your app passes the following parameters to this method:

- the `AppConnectCordova.ACPolicyState` value that represents the success or failure of handling the new authorization status.

  Pass the value `APPLIED` if the app successfully handled the new status. Otherwise, pass the value `ERROR`. Passing the value `UNSUPPORTED` is not allowed, because every app must handle authorization status changes.

- a string explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the new authorization status. The string is reported in the Core log files.

The displayMessage() method

The AppConnect Cordova Plugin provides a method that causes the Mobile@Work app to display the current authorization status message:

```
AppConnectCordova.displayMessage(message)
```

In most cases, your production app does not use this method. Your production app is responsible for displaying the message that it receives in the event handler for an authorization status change. Your app controls exactly when and how to display the string.

However, you can temporarily use this method when your app is under development. For example, when the status changes to `UNAUTHORIZED`, your app must exit all sensitive views. This requirement can make displaying the message difficult, depending on the application. In this case, use the `AppConnectCordova.displayMessage()` method until you are able to fully develop your app.

App-specific configuration API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to receive app-specific configuration from MobileIron Core. For an overview of this feature, see "Configuration specific to the app" on page 18.

The config() method

The following method returns the current key-value pairs for the app-specific configuration:

```
AppConnectCordova.config()
```

Return value: An array of key-value pairs. Each key and value is a string corresponding to the key and value configured on MobileIron Core. For example:

```
[
    {"serverURL", "enterpriseServer.finance.com"},
    {"userID", "jdoe@myenterprise.com"},
    {"appAdvancedFeaturesEnabled", "true"}
]
```

If no key-value pairs are configured on MobileIron Core, the return value is an empty array:

```
[]
```

Calling config() when your app launches

When your app launches:

1. Upon receiving the Cordova `'deviceready'` event, call the method `AppConnectCordova.initialize()`.
2. Wait for the AppConnectCordova event `'appconnect.isReady'` before calling the `AppConnectCordova.config()` method.
3. In the event handler for the `'appconnect.isReady'` event, call the `AppConnectCordova.config()` method. It returns the key-value pairs, if any, that are configured on MobileIron Core for the app. Apply the configuration according to your application's requirements and logic.

config() return value after updates to app-specific configuration

On any updates to the app-specific configuration while the app is running, the AppConnect Cordova Plugin generates the `'appconnect.configChangedTo'` event. Subsequent calls to the `AppConnectCordova.config()` return the updated key-value pairs.

The 'appconnect.configChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.configChangedTo'` event when the app-specific configuration changes after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description

`newConfig`	An array of key-value pairs. Each key and value is a string corresponding to the key and value configured on MobileIron Core.
	For example:
	`[`
	`    {"serverURL", "enterpriseServer.finance.com"},`
	`    {"userID", "jdoe@myenterprise.com"},`
	`    {"appAdvancedFeaturesEnabled", "true"}`
	`]`
	If no key-value pairs are configured on MobileIron Core, the return value is an empty array:
	`[]`

Event handler for 'appConnect.configChangedTo' event

When a change has occurred to the app-specific configuration on MobileIron Core, the AppConnect Cordova Plugin:

1. Stores the updated array of key-value pairs so that subsequent calls to the `AppConnectCordova.config()` method return the updated key-value pairs.
2. Generates the `'appconnect.configChangedTo'` event.

You can optionally add an event listener to your document object for the `'appconnect.configChangedTo'` event. For example:

```
document.addEventListener('appconnect.configChangedTo',
                          this.onConfigChangedTo, false);
```

Add this event listener only if your app uses app-specific configuration key-value pairs that the MobileIron Core administrator configures on the Admin Portal.

In the event handler, your app:

- Applies the configuration according to your application’s requirements and logic.
- Calls the `AppConnectCordova.configApplied()` method.

The configApplied() method

After your event handler processes the information provided in the `'appconnect.configChangedTo'` event, it must call this acknowledgment method:

```
AppConnectCordova.configApplied(policyState, message)
```

Your app passes the following parameters to this method:

- the `AppConnectCordova.ACPolicyState` value that represents the success or failure of handling the new configuration.

Pass the value `APPLIED` if the app successfully handled the new configuration. Otherwise, pass the value `ERROR`. Pass the value `UNSUPPORTED` if your app does not support configuration from MobileIron Core. If you do not implement an event handler for the `'appconnect.configChangedTo'` event, the AppConnect Cordova Plugin behaves as if you passed it `UNSUPPORTED`.

- a string explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the app-specific configuration updates. The string is reported in the Core log files.

Pasteboard policy API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to handle its pasteboard policy as determined by MobileIron Core. For an overview of this feature, see "Data loss prevention policies" on page 19.

This policy determines whether your app is allowed to copy content *to* the pasteboard. This policy does not impact whether your app is allowed to paste content *from* the pasteboard into your app.

The ACPasteboardPolicy enumeration

The `AppConnectCordova.ACPasteboardPolicy` enumeration provides the possible pasteboard statuses for the app:

```
AppConnectCordova.prototype.ACPasteboardPolicy = {
    UNAUTHORIZED:  0, // The application may not write data to the pasteboard.
    AUTHORIZED:    1  // The application may write data to the pasteboard.
}
```

The pasteboardPolicy() method

The following method returns the current status of the pasteboard policy for the app:

```
AppConnectCordova.pasteboardPolicy()
```

Return value: An `AppConnectCordova.ACPasteboardPolicy` enumeration value.

Calling pasteboardPolicy() when your app launches

When your app launches:

1. Upon receiving the Cordova `'deviceready'` event, call the method `AppConnectCordova.initialize()`.
2. Wait for the AppConnectCordova event `'appconnect.isReady'` before calling the `AppConnectCordova.pasteboardPolicy()` method.
3. In the event handler for the `'appconnect.isReady'` event, enable or disable the app's ability to copy content to the pasteboard. Whether to enable or disable copying depends on the `AppConnectCordova.ACPasteboardPolicy` value returned from `AppConnectCordova.pasteboardPolicy()`.

pasteboardPolicy() return value after updates to pasteboard policy

On any updates to the pasteboard policy while the app is running, the AppConnect Cordova Plugin generates the `'appconnect.pasteboardPolicyChangedTo'` event. Subsequent calls to the `AppConnectCordova.pasteboardPolicy()` method returns the updated pasteboard policy.

The 'appconnect.pasteboardPolicyChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.pasteboardPolicyChangedTo'` event when the pasteboard policy changes after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description
`newPasteboardPolicy`	An `AppConnectCordova.ACPasteboardPolicy` enumeration value

Event handler for 'appConnect.pasteboardPolicyChangedTo' event

When a change has occurred to the pasteboard policy on MobileIron Core, the AppConnect Cordova Plugin:

1. Stores the new `AppConnectCordova.ACPasteboardPolicy` value so that subsequent calls to the `AppConnectCordova.pasteboardPolicy()` method return the updated policy.
2. Generates the `'appconnect.pasteboardPolicyChangedTo'` event.

You can optionally add an event listener to your document object for the `'appconnect.pasteboardPolicyChangedTo'` event. For example:

```
document.addEventListener('appconnect.pasteboardPolicyChangedTo',
                          this.onPasteboardPolicyChangedTo, false);
```

Add this event listener only if your app copies content *to* the pasteboard. This policy does not impact whether your app is allowed to paste content *from* the pasteboard into your app.

Your app handles the new status as follows:

New status	App actions
`UNAUTHORIZED`	• Disables its ability to copy content to the pasteboard • Calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
`AUTHORIZED`	• Enables its ability to copy content to the pasteboard • Calls the `AppConnectCordova.pasteboardPolicyApplied()` method.

The pasteboardPolicyApplied() method

After your event handler processes the information provided in the `'appconnect.pasteboardPolicyChangedTo'` event, it must call this acknowledgment method:

```
AppConnectCordova.prototype.pasteboardPolicyApplied(policyState, message)
```

Your app passes the following parameters to this method:

- the `AppConnectCordova.ACPolicyState` value that represents the success or failure of handling the new pasteboard policy.

Pass the value `APPLIED` if the app successfully handled the new policy. Otherwise, pass the value `ERROR`. Pass the value `UNSUPPORTED` if your app does not support copying content to the pasteboard. If you do not implement an event handler for the `'appconnect.pasteboardPolicyChangedTo'` event, the AppConnect Cordova Plugin behaves as if you passed it `UNSUPPORTED`.

- a string explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the new pasteboard policy. The string is reported in the Core log files.

Open In policy API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to handle its Open In policy as determined by MobileIron Core. For an overview of this feature, see "Data loss prevention policies" on page 19.

Specifically, when an app is allowed to use Open In, it can share a document with another app on the device. This capability:

- is usually presented to the user as an Open In menu item.
- includes sending documents or document portions by encoding them in custom URLs handled by other applications.
- includes any future iOS feature or 3rd party library functionality that has the effect of sharing a document or portion of a document with another app.

The ACOpenInPolicy enumeration

The `AppConnectCordova.ACOpenInPolicy` enumeration provides the possible Open In statuses for the app:

```
AppConnectCordova.prototype.ACOpenInPolicy = {
    UNAUTHORIZED:  0, // The application may not use Open In.
    AUTHORIZED:    1  // The application may use Open In.
    WHITELIST:     2  // The application may only use Open In to send
                      // documents to applications in the whitelist.
}
```

The openInPolicy() and openInWhitelist() methods

OpenInPolicy() method

The following method returns the current status of the Open In policy for the app:

```
AppConnectCordova.openInPolicy()
```

Return value: An `AppConnectCordova.ACPOpenInPolicy` enumeration value.

OpenInWhitelist() method

The following method returns the current Open In whitelist for the app. The whitelist is the set of apps to which your app is allowed to send documents.

```
AppConnectCordova.openInWhitelist()
```

Return value: An array. Each array element is a string which is the bundle ID of an app in the whitelist.

For example:

```
["com.somecompany.someapp","com.anothercompany.anotherapp","com.thatcompany.thatapp"]
```

Calling OpenInPolicy() and OpenInWhitelist() when your app launches

When your app launches:

1. Upon receiving the Cordova `'deviceready'` event, call the method `AppConnectCordova.initialize()`.
2. Wait for the AppConnectCordova event `'appconnect.isReady'` before calling the `OpenInPolicy()` method.
3. In the event handler for the `'appconnect.isReady'` event:
   - enable or disable the app's ability to use the Open In feature. Whether to enable or disable Open In depends on the `AppConnectCordova.ACPOpenInPolicy` value returned from `AppConnectCordova.OpenInPolicy()`.
   - If the value returned from `AppConnectCordova.OpenInPolicy()` is `WHITELIST`, make sure that only the apps in the whitelist can open documents.

Method return values after updates to Open In policy

On any updates to the Open In policy while the app is running, the AppConnect Cordova Plugin generates the `'appconnect.OpenInPolicyChangedTo'` event. Subsequent calls to the `AppConnectCordova.openInPolicy()` and `AppConnectCordova.openInWhitelist()` methods return the updated Open In policy and whitelist.

The 'appconnect.openInPolicyChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.openInPolicyChangedTo'` event when the Open In policy or whitelist changes after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description
`newOpenInPolicy`	An `AppConnectCordova.ACOpenInPolicy` enumeration value
`newWhitelist`	An array. Each array element is a string which is the bundle ID of an app in the whitelist. For example: `["com.somecompany.someapp","com.anothercompany.anotherapp"]` **Note:** When the Open In policy on MobileIron Core specifies "All AppConnect apps", the `newOpenInPolicy` value is `WHITELIST`. The `newWhitelist` value lists all the currently authorized AppConnect apps. Therefore, your app handles the "All AppConnect apps" Core setting the same way it handles the "whitelist" Core setting.

Event handler for 'appConnect.openInPolicyChangedTo' event

When a change has occurred to the Open In policy or whitelist on MobileIron Core, the AppConnect Cordova Plugin:

1. Stores the new `ACOpenInPolicy` value so that subsequent calls to the `AppConnectCordova.openInPolicy()` method return the updated policy.
2. Stores the new value of the whitelist so that subsequent calls to the `AppConnectCordova.openInWhitelist()` method return the updated whitelist.
3. Generates the `'appconnect.openInPolicyChangedTo'` event.

You can optionally add an event listener to your document object for the `'appconnect.openInPolicyChangedTo'` event. For example:

```
document.addEventListener('appconnect.openInPolicyChangedTo',
                          this.onOpenInPolicyChangedTo, false);
```

Add this event listener only if your app uses the Open In feature.

Your app handles the new status as follows:

New status	App actions
`UNAUTHORIZED`	• Disables its ability to use the Open In feature. • Calls the `AppConnectCordova.openInPolicyApplied()` method.
`AUTHORIZED`	• Enables its ability to use the Open In feature. • Calls the `AppConnectCordova.openInPolicyApplied()` method.
`WHITELIST`	• Enables its ability to use the Open In feature, but limits the set of apps that can open documents to the apps in the whitelist. • Calls the `AppConnectCordova.openInPolicyApplied()` method.

The openInPolicyApplied() method

After your event handler processes the information provided in the `'appconnect.openInPolicyChangedTo'` event, it must call this acknowledgment method:

```
AppConnectCordova.openInPolicyApplied(policyState, message)
```

Your app passes the following parameters to this method:

- the `AppConnectCordova.ACPolicyState` value that represents the success or failure of handling the new Open In policy.

  Pass the value `APPLIED` if the app successfully handled the new policy. Otherwise, pass the value `ERROR`. Pass the value `UNSUPPORTED` if your app does not support the Open In feature. If you do not implement an event handler for the `'appconnect.openInPolicyChangedTo'` event, the AppConnect Cordova Plugin behaves as if you passed it `UNSUPPORTED`.
- a string explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the new Open In policy. The string is reported in the Core log files.

Print policy API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to handle its print policy as determined by MobileIron Core. For an overview of this feature, see "Data loss prevention policies" on page 19.

The ACPrintPolicy enumeration

The `AppConnectCordova.ACPrintPolicy` enumeration provides the possible print statuses for the app:

```
AppConnectCordova.prototype.ACPrintPolicy = {
    UNAUTHORIZED:  0, // The application may not use Print.
    AUTHORIZED:    1  // The application may use Print.
}
```

The printPolicy() method

The following method returns the current status of the print policy for the app:

```
AppConnectCordova.printPolicy()
```

Return value: An `AppConnectCordova.ACPrintPolicy` enumeration value.

Calling printPolicy() when your app launches

When your app launches:

1. Upon receiving the Cordova `'deviceready'` event, call the method `AppConnectCordova.initialize()`.
2. Wait for the AppConnectCordova event `'appconnect.isReady'` before calling the `printPolicy()` method.
3. In the event handler for the `'appconnect.isReady'` event, enable or disable the app's ability to print. Whether to enable or disable printing depends on the `AppConnectCordova.ACPrintPolicy` value returned from `AppConnectCordova.printPolicy()`.

printPolicy() return value after updates to print policy

On any updates to the print policy while the app is running, the AppConnect Cordova Plugin generates the `'appconnect.printPolicyChangedTo'` event. Subsequent calls to the `AppConnectCordova.printPolicy()` method returns the updated print policy.

The 'appconnect.printPolicyChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.printPolicyChangedTo'` event when the print policy changes after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description
`newPrintPolicy`	An `AppConnectCordova.ACPrintPolicy` enumeration value

Event handler for 'appConnect.printPolicyChangedTo' event

When a change has occurred to the print policy on MobileIron Core, the AppConnect Cordova Plugin:

1. Stores the new `ACPrintPolicy` value so that subsequent calls to the `AppConnectCordova.printPolicy()` method return the updated policy.
2. Generates the `'appconnect.printPolicyChangedTo'` event.

You can optionally add an event listener to your document object for the `'appconnect.printPolicyChangedTo'` event. For example:

```
document.addEventListener('appconnect.printPolicyChangedTo',
                          this.onPrintPolicyChangedTo, false);
```

Add this event listener only if your app is able to print.

Your app handles the new status as follows:

New status	App actions
`UNAUTHORIZED`	• Disables its ability to print. • Calls the `AppConnectCordova.printPolicyApplied()` method.
`AUTHORIZED`	• Enables its ability to print. • Calls the `AppConnectCordova.printPolicyApplied()` method.

The AppConnectCordova.printPolicyApplied() method

After your event handler processes the information provided in the `'appconnect.printPolicyChangedTo'` event, it must call this acknowledgment method:

```
AppConnectCordova.prototype.printPolicyApplied (policyState, message)
```

Your app passes the following parameters to this method:

- the `AppConnectCordova.ACPolicyState` value that represents the success or failure of handling the new print policy.

  Pass the value `APPLIED` if the app successfully handled the new policy. Otherwise, pass the value `ERROR`. Pass the value `UNSUPPORTED` if your app does not support printing. If you do not implement an event handler for the `'appconnect.printPolicyChangedTo'` event, the AppConnect Cordova Plugin behaves as if you passed it `UNSUPPORTED`.
- a string explaining the `AppConnectCordova.ACPolicyState` value.

  Typically, you use this string to report the reason the app failed to apply the new print policy. The string is reported in the Core log files.

Getting the AppConnect library version

The AppConnect Cordova JavaScript interface provides a method that returns the version of the AppConnect library.

```
AppConnectCordova.version ()
```

This method returns a string value. The value reflects the version of the AppConnect library that the AppConnect Cordova Plugin is using.

A best practice is to report the AppConnect library version number on your app's About page. This information is useful to support organizations if a device user has any issues with the app.

For example, use the following statement to get the AppConnect library version:

```
var AppConnectVersion = AppConnectCordova.version();
```

Caching tunneled URL responses

Apps that access enterprise servers can use the AppTunnel feature, as described in “Data loss prevention policies” on page 19. By default, for a tunneled URL request:

- The data for the URL is reloaded from the originating source. Any existing locally cached response is ignored.
- The data in the response is not stored in the local cache.

The reason that the AppTunnel feature does not use locally cached responses is to avoid caching sensitive enterprise server data on the device.

However, some apps have requirements to use locally cached responses. Some examples are:

- The app requires a response even when the device has no network connectivity.
- The app requires a customized response.

If your app requires locally cached responses for URL requests that use AppTunnel, use the following AppConnect Cordova Plugin method:

```
AppConnectCordova.allowLocalCachingForTunneledRequests(flag)
```

The value of `flag` has the following impact:

- `true`

  Allows caching for requests and responses that use AppTunnel. However, whether caching actually occurs depends on the cache policy for the URL request.
- `false`

  Clears all cached responses, including responses for URL requests not using AppTunnel.

Important: Do not cache sensitive data.

Chapter 4

Developing third-party dual-mode apps

What is a dual-mode app?

If your AppConnect app is distributed from the Apple App Store, due to Apple App Store requirements, your app is required to work as either:

- an AppConnect app for enterprise users
- a regular app for general consumers

Such an app is called a *dual-mode* app. Using one code base and APIs in the AppConnect for iOS Cordova Plugin, the app automatically decides which way to behave the first time it launches. Running as an AppConnect app, the app supports the AppConnect features, such as authorization, data loss prevention, and secure file I/O. Running as a regular app, the app supports none of the AppConnect features. Furthermore, depending on the app, the functionality available as a regular app can differ significantly from the functionality available as an AppConnect app. For example, as a regular app, the app does not allow the user to access any sensitive enterprise data.

AppConnect apps distributed from the Apple App Store must be dual-mode apps. If you are a third-party app developer, you typically build apps for Apple App Store distribution. If you are an in-house app developer, your apps are typically distributed from MobileIron Core.

When running as an AppConnect app, the app is in *AppConnect Mode*, because MobileIron, through MobileIron Core, Mobile@Work, and the AppConnect library, provides AppConnect management. When running as a regular app, the app is in *Non-AppConnect Mode*.

Important: If your app is not distributed from the Apple App Store and works only as an AppConnect app, ignore the dual-mode capability and associated APIs.

Dual-mode app states

An app must maintain a dual-mode state that indicates whether it is in AppConnect Mode. It stores this state persistently, so that when it next launches, it knows how to behave. The possible states are:

- Undecided

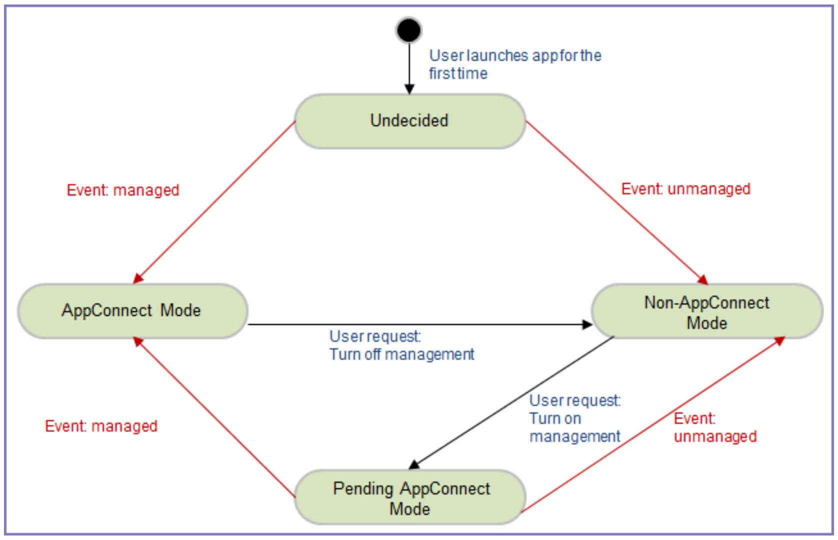
  The app has initialized for the first time and has not yet decided whether to run in AppConnect Mode or Non-AppConnect Mode.
- AppConnect Mode

  The app is running as an AppConnect app. It supports the AppConnect features, such as authorization and data loss prevention policies.
- Non-AppConnect Mode

  The app is running as a regular app.
- Pending AppConnect Mode

  The app changes to this state if the device user explicitly requests a change to AppConnect Mode using the app’s user interface. For example, device users in an enterprise sometimes have installed and used an app before the enterprise requires it as an AppConnect app. In this state, the app is waiting for an event from the AppConnect Cordova Plugin to find out whether MobileIron AppConnect components are managing the app.

The following diagram summarizes the state transitions that a dual-mode app implements. See “High-level dual-mode app behavior” on page 56 for more information about these state transitions.



High-level dual-mode app behavior

When the app launches for the first time

When a dual-mode app launches for the first time, it does not know whether it is managed by MobileIron. It does the following high-level steps:

1. Sets its initial dual-mode state to Undecided.
2. Starts the AppConnect library.
3. Waits for an event from the AppConnect Cordova Plugin indicating whether MobileIron is managing the app. For example, one typical reason that MobileIron is not managing the app is that Mobile@Work is not installed on the device.
4. Changes its state to AppConnect Mode or Non-AppConnect Mode according to the event.
    - When changing to Non-AppConnect Mode, the app notifies the AppConnect library that it is retiring. Normally, MobileIron Core decides when to retire an app. In this case, the app is retiring itself. Then the app stops the AppConnect library. It behaves as a regular app.
    - When changing to AppConnect Mode, the app behaves as an AppConnect app.
5. Stores the dual-mode state persistently for the next time it launches.

Note: For more details, including specific API calls for these steps, see “API call sequence when the app launches for the first time” on page 60.

When an app subsequently launches

On subsequent launches, the app does the following high-level steps:

1. Gets the dual-mode state that it stored.
2. Checks the dual-mode state.
3. If the state is AppConnect Mode, starts the AppConnect library.
   The app continues as an AppConnect app.
4. If the state is Non-AppConnect Mode, continues as a regular app.
   The app does not start the AppConnect library.

Note: For more details, including specific API calls for these steps, see “API call sequence when the app subsequently launches” on page 61.

User requests to switch to Non-AppConnect Mode

A dual-mode app provides a user interface that allows the device user to explicitly request that MobileIron no longer manage the app. That is, the user requests a change to Non-AppConnect Mode. This user interface can be useful if a device user leaves an enterprise, but still wants to use the app as a regular app.

Users are typically not aware of the term “AppConnect”. Therefore, the user interface should use other terminology. For example, an app can use “Managed by MobileIron” in its user interface. Another possibility is “Secure enterprise mode”.

When switching from AppConnect Mode to Non-AppConnect Mode, the app does the following high-level steps:

1. Removes all its secure data, since regular apps do not have secure data.
2. Notifies the AppConnect library that it is retiring.

   Normally, MobileIron Core decides when to retire an app. In this case, the app is retiring itself.
3. Stops the AppConnect library.
4. Stores its dual-mode state, Non-AppConnect Mode, persistently for the next time it launches.
5. Continues running as a regular app.

   For example, the app no longer enforces AppConnect policies.

Note: For more details, including specific API calls for these steps, see "API call sequence when user requests Non-AppConnect Mode" on page 62.

User requests to switch to AppConnect Mode

A dual-mode app provides a user interface that allows the device user to explicitly request that MobileIron manage the app. That is, the user requests a change to AppConnect Mode. For example, device users in an enterprise sometimes have installed and used an app before the enterprise requires it as an AppConnect app.

Users are typically not aware of the term "AppConnect". Therefore, the user interface should use other terminology. For example, an app can use "Managed by MobileIron" in its user interface. Another possibility is "Secure enterprise mode".

When switching from Non-AppConnect Mode to AppConnect Mode, the app does the following high-level steps:

1. Starts the AppConnect library.
2. Changes to the Pending AppConnect Mode state.
3. Waits for an event from the AppConnect library indicating that MobileIron is managing the app.
4. If the app receives the event that MobileIron is managing the app, the app changes state to AppConnect Mode, and persistently stores the new state. It begins behaving as an AppConnect app. For example, it enforces DLP policies.

Note: For more details, including specific API calls for these steps, see "API call sequence when user requests AppConnect Mode" on page 62.

Data loss prevention policy handling

When a dual-mode app changes from Non-AppConnect Mode to AppConnect Mode, it starts enforcing the AppConnect data loss prevention policies that it supports. For example, if the app supports the Open In policy, it allows document sharing only as directed by the policy it receives from the AppConnect library. When changing to Non-AppConnect Mode, the app stops enforcing the AppConnect DLP policies.

Dual-mode API details

The AppConnect Cordova Plugin provides an enumeration, an event, and methods that allow an app to behave as a dual-mode app.

The ACManagedPolicy enumeration

The `ACManagedPolicy` enumeration provides the possible managed policy values for the app:

```
AppConnectCordova.prototype.ACManagedPolicy = {

    UNKNOWN:   0, // The AppConnect library has not yet determined
                  // whether the app is managed by MobileIron.

    UNMANAGED: 1, // The application is not currently managed by MobileIron.

    MANAGED:   2  // The application is currently managed by MobileIron.
  }
```

The managedPolicy() method

The following method returns the current status of the managed policy for app. The managed policy indicates whether MobileIron is managing the app.

```
AppConnectCordova.managedPolicy()
```

Return value: An `AppConnectCordova.ACManagedPolicy` enumeration value.

Note: Currently, apps have no need to use the `managedPolicy()` method. Dual-mode apps depend on events to instigate changes to the app’s dual-mode state.

The 'appconnect.managedPolicyChangedTo' event

The AppConnect Cordova Plugin generates the `'appconnect.managedPolicyChangedTo'` event to provide managed policy status updates after the plugin initialization is completed.

The event object passed to the event handler contains:

Event object properties	Description
`newManagedPolicy`	An `AppConnectCordova.ACManagedPolicy` enumeration value

Event handler for 'appConnect.managedPolicyChangedTo' event

When a change has occurred to the managed policy, the AppConnect Cordova Plugin:

1. Stores the new `ACManagedPolicy` value so that subsequent calls to the `AppConnectCordova.managedPolicy()` method return the updated policy.
2. Generates the `'appconnect.managedPolicyChangedTo'` event.

In a dual-mode app, add an event listener to your document object for the `'appconnect.managedPolicyChangedTo'` event. For example:

```
document.addEventListener('appconnect.managedPolicyChangedTo',
                          this.onManagedPolicyChangedTo, false);
```

Your app handles the new managed policy status by changing app's state to AppConnect Mode or Non-AppConnect Mode as described in:

- "API call sequence when the app launches for the first time" on page 60
- "API call sequence when user requests Non-AppConnect Mode" on page 62
- "API call sequence when user requests Non-AppConnect Mode" on page 62

The stop method

The following method shuts down the AppConnect library for the app.

```
AppConnectCordova.stop()
```

The app calls the `AppConnectCordova.stop()` method when it changes state to Non-AppConnect Mode.

If at a later time, the user requests to change to AppConnect Mode, the app restarts the AppConnect library.

For an example of when to call the `AppConnectCordova,stop()` method, see "API call sequence when user requests AppConnect Mode" on page 62.

For an example of restarting the AppConnect library, see "API call sequence when user requests AppConnect Mode" on page 62.

The retire method

The following method informs the AppConnect library that the app is retiring. Normally, MobileIron Core decides when to retire an app. In this case, the app is retiring itself.

```
AppConnectCordova.retire()
```

Calling `retire()` causes the AppConnect library to:

- clean up information it keeps about the app, including secure data.
- set its managedPolicy status for the app to `UNKNOWN`.

Important: An app must call `AppConnectCordova.retire()` and then immediately call `AppConnectCordova.stop()` when it is changing to Non-AppConnect Mode.

API call sequence when the app launches for the first time

When a dual-mode app launches for the first time, it does not know whether it is managed by MobileIron. It waits for an `'appConnect.managedPolicyChangedTo'` event to determine whether to continue in AppConnect Mode or Non-AppConnect Mode.

Therefore, when launching for the first time, the app does the following:

1. Sets the dual-mode state to Undecided, and persistently stores it.
2. Starts the AppConnect library after receiving the the Cordova `'deviceready'` event: `AppConnectCordova.initialize();`
3. Waits for the `'appConnect.managedPolicyChangedTo'` event.
4. In the event handler for the `'appConnect.managedPolicyChangedTo'` event, the app changes its dual-mode state. The state change depends on the value of the `newManagedPolicy` property in the event object:
   - If the value is `MANAGED`, the app changes to AppConnect Mode.
   - If the value is `UNMANAGED`, the app changes to Non-AppConnect Mode.
5. If the app changes to AppConnect Mode, it persistently stores its new dual-mode state. It begins behaving as an AppConnect app. For example, it enforces DLP policies.
6. If the app changes to Non-AppConnect Mode, it does the following:
   - Calls `AppConnectCordova.retire()` and then stops the AppConnect library:

     ```
     AppConnectCordova.retire();
     AppConnectCordova.stop();
     ```

   - Persistently stores its new state.
   - Begins behaving as a regular, non-AppConnect app.

API call sequence when the app subsequently launches

When a dual-mode app subsequently launches, it relies on its persisted dual-mode state to determine whether the behave as an AppConnect app.

When launching for a subsequent time, the app does the following:

1. Gets its persisted dual-mode state.
2. Continues as a regular, non-AppConnect app if the dual-mode state is Non-AppConnect Mode.

   Note: The app does not call `AppConnectCordova.initialize()` to start the AppConnect library in this case.

   The remaining steps do not apply.
3. Starts the AppConnect library if the dual-mode state is anything except Non-AppConnect Mode.

   ```
   AppConnectCordova.initialize();
   ```

4. Waits for the `'appconnect.isReady'` event before accessing other AppConnectCordova JavaScript interfaces such as `AppConnectCordova.authState()`, `AppConnectCordova.authMessage()`, and `AppConnectCordova.config()`.
5. Continues as an AppConnect app if the dual-mode state is AppConnect Mode.

API call sequence when user requests Non-AppConnect Mode

If the device user, through the app's user interface, requests to change to Non-AppConnect Mode, the app makes the change.

The app does the following:

1. Performs its usual retire actions, such as removing all its sensitive data, since regular apps do not have sensitive data.
2. Persistently saves its dual-mode state as Non-AppConnect Mode.
3. Calls the `AppConnectCordova.retire()` method, and then stops the AppConnect library.

   ```
   AppConnectCordova.retire();
   AppConnectCordova.stop();
   ```

4. Continues as a regular, non-AppConnect app.

When the app next launches, it checks its dual-mode state. Because the state is Non-AppConnect Mode, the app does not start the AppConnect library.

API call sequence when user requests AppConnect Mode

If the device user, through the app's user interface, requests to change to AppConnect Mode, the app attempts to make the change.

The app does the following:

1. Changes to the Pending AppConnect Mode state.
2. Starts the AppConnect library by calling `AppConnectCordova.initialize()`.
3. Waits for the `'appConnect.managedPolicyChangedTo'` event.

In the event handler for the `'appConnect.managedPolicyChangedTo'` event:

- If the `newManagedPolicy` property of the event object has the value `UNMANAGED`:

  The app changes its dual-mode state back to Non-AppConnect Mode. The app persistently stores the state.

  The app calls `AppConnectCordova.retire()`, and then stops the AppConnect library:

  ```
  AppConnectCordova.retire();
  AppConnectCordova.stop();
  ```

  The app notifies the user of the failure to change to AppConnect Mode. It continues behaving as a regular, non-AppConnect app.
- If the `newManagedPolicy` property in the even object has the value `MANAGED`:

  The app changes its dual-mode state to AppConnect Mode. The app persistently stores the state.

  The app checks if the authorization status is retired. If it is, the app performs its usual retire actions, such as removing all its sensitive data.

  Finally, the app notifies the user of the successful change to AppConnect Mode. It continues behaving as an AppConnect app.

Chapter 5

Best practices using the AppConnect for iOS Cordova Plugin

The following are best practices for developing secure enterprise apps:

- "Display authorization status in the home screen" on page 65
- "Allow the user to enter credentials manually" on page 66
- "Limit the size of configuration data from MobileIron Core" on page 66
- "Consider limitations when using the iOS simulator" on page 66
- "Remove secure data from views before moving to the background" on page 67
- "Do not put secure data in the app bundle" on page 67
- "Indicate to the user that the app is initializing" on page 67
- "Provide documentation about your app to the MobileIron Core administrator" on page 68

Display authorization status in the home screen

When an app becomes unauthorized or retires, the AppConnect Cordova Plugin `AppConnectCordova.authState()` method returns `UNAUTHORIZED` or `RETIRED`. Additionally, the `AppConnectCordova.authMessage()` method returns a string that explains to the device user why the app is unauthorized or retired. The string sometimes also explains what the device user can do to make the app authorized again.

The app should display the `AppConnectCordova.authMessage()` string. However, consider that since the app is now unauthorized or retired, the app must exit its secure functionality. Therefore, the best user experience is to display the string in a home view that never contains secure information.

The following alternatives for displaying the `AppConnectCordova.authMessage()` string are not recommended:

- Do not display the string on top of the current view. Beneath the message, the current view can still have secure information visible.
- Do not use the `AppConnectCordova.displayMessage()` method. This method does not match the look of your app.
- Do not exit the app without displaying the string.

Allow the user to enter credentials manually

Always provide a way for a user to enter login credentials manually in your app. Provide this user interface even if you are receiving login credentials in app-specific configuration information from the AppConnect library.

As described in "Configuration specific to the app" on page 18, a MobileIron Core administrator can set up configuration information for your app on Core. Your app receives the information using the AppConnect Cordova interfaces. This information can include authentication credentials, such as username, password and certificates, for a corporate service. Because the app receives the information, the device user does not have to enter the information.

However, if the credentials change, the amount of time for the change to reach your application can vary. Some variables that impact this notification include:

- the app checkin interval that the administrator configured on Core. This value is the maximum number of minutes until devices running AppConnect apps receive updates of their AppConnect policies and app-specific configurations.
- whether the device has network coverage.

Therefore, providing changes to devices is not a real-time process and can take up to several hours. Therefore, if the corporate service rejects the credentials, provide a way for the user to enter the credentials manually.

Limit the size of configuration data from MobileIron Core

Do not design your app to use large amounts of configuration data from MobileIron Core.

As described in "Configuration specific to the app" on page 18, a Core administrator can set up configuration information for your app on Core. Your app receives the information using the AppConnect Cordova Plugin. Use this capability only for short strings and options, such as server addresses, authentication credentials, and certificates.

Do not use it for larger data items, such as documents, large blocks of HTML, or images. For large data items, use a web service to deliver the items. Use AppConnect configuration only to provide the URL for the web service.

Although no precise upper limit is defined for an item configured on Core, a large item can impact Core performance. It can also slow connectivity between Core and the Mobile@Work for iOS app. A very large item can possibly cause the communication protocol between Core and Mobile@Work to fail entirely.

Consider limitations when using the iOS simulator

To fully test an AppConnect app, debug on a device. On a device, your testing includes the Mobile@Work app, which is necessary for the complete flow of data from MobileIron Core to your app.

You can do initial functionality testing in the iOS simulator in Xcode. However, when using the AppConnect Cordova Plugin in the iOS simulator, the plugin interfaces behae as follows:

- `AppConnectCordova.authState()` returns `AUTHORIZED`
- `AppConnectCordova.config()` returns no entries
- `AppConnectCordova.pasteboard()` returns `AUTHORIZED`
- `AppConnectCordova.openInPolicy()` returns `AUTHORIZED`
- `AppConnectCordova.printPolicy()` returns `AUTHORIZED`

This behavior is necessary because no simulator version of Mobile@Work is available, and Mobile@Work is necessary for your app to receive AppConnect Cordova Plugin events. Without the events, the app's authorization status cannot change to `AUTHORIZED`, and your app cannot execute its logic that accesses its secure data and functionality. The AppConnect Cordova Plugin's special simulator behavior solves this problem, allowing you to use the iOS simulator to test your app's functionality. You cannot, however, use the simulator to test handling events from the AppConnect Cordova Plugin.

Remove secure data from views before moving to the background

A best practice for all apps, not just AppConnect apps, is to remove or hide sensitive data from views when the app moves from the active to inactive state. Some examples of when an app enters the inactive state are:

- When an incoming phone call is received.
- When the Touch ID prompt displays for the user to log back into AppConnect apps.
- When the user quits the app, which transitions the app through the inactive state to the background state. Note that iOS saves a snapshot of your app's screen when it transitions your app to the background.

The app receives the Cordova `'resign'` event before it goes into the background. Therefore, in the event handler for the `'resign'` event, make sure secure information is not visible.

Do not put secure data in the app bundle

Files that you package in your app bundle are not encrypted files. Also, files packaged with an app cannot be modified at runtime. Therefore, these files are not secure. Therefore, include only non-sensitive data in the app bundle.

Indicate to the user that the app is initializing

Indicate in the user interface that the app is initializing if the app requires the AppConnectCordova JavaScript interfaces to determine what to do. For example, use an activity indicator (spinner). Remove the activity indicator after the app receives the `'appconnect.isReady'` event.

One reason this indication is important involves when to display sensitive data. Do not show any sensitive data until the AppConnect library is ready, because until that time, the app cannot determine whether it is authorized. Only an authorized app should show sensitive data.

Provide documentation about your app to the MobileIron Core administrator

Whether your app is an in-house app or is available from the Apple App Store, a MobileIron Core administrator configures Core with information about your app. Provide the Core administrator documentation that specifies:

- whether your app enforces the pasteboard policy.

  The Core administrator needs to know whether allowing or not allowing your app to copy content to the pasteboard has impact on your app’s behavior.
- whether your app enforces the print policy.

  The Core administrator needs to know whether allowing or not allowing your app to use print capabilities has impact on your app’s behavior.
- whether your app enforces the Open In policy.

  The Core administrator needs to know whether allowing or not allowing your app to share documents with other apps has impact on your app’s behavior. The administrator also needs to know whether your app supports a whitelist of apps that can open documents from your app. If your app does support a whitelist, and you have a recommended list of whitelisted apps, document their bundle ids.
- the app-specific configuration key-value pairs.

  Provide a list of the key-value pairs that your app expects to receive through the AppConnect API. Provide each key’s default value if it has one. Specify if the value should default to the device’s user’s LDAP user ID or password.
- AppTunnel information

  If your app expects to interact with internal servers using AppTunnel, provide information about those internal servers.

  For example:
  - Explain the type of servers your app interacts with, such as, for example, SharePoint servers.
  - Specify if your app expects to receive internal servers’ host names using the app-specific configuration API.
  - Specify if your app expects to be able to interact with all internal servers.
  - If you are an in-house app developer, provide the host names of the internal servers that your app interacts with. Also, provide the port number on each internal server that the app connects to.
- HTTPS connections that your app makes that use certificate authentication to an enterprise service.

  For in-house app developers, provide the URLs of the enterprise services that use certificate authentication.

  If your app receives these URLs through app-specific configuration, make sure you listed the URLs in the app-specific configuration key-value pair documentation.

- whether your app is a dual-mode app.

  Also, if your app is a dual-mode app, and it allows the device user to switch between AppConnect mode and non-AppConnect mode, document what the device user must do.

Chapter 6

Testing for third-party app developers

Third-party AppConnect app testing overview

If you are an in-house app developer with access to your organization’s MobileIron Core, see “Testing for in-house app developers” on page 93.

Testing with Core is necessary to verify the AppConnect-related functionality of your AppConnect app. If your app accesses servers behind a firewall using AppTunnel, a Standalone Sentry is necessary to verify the AppTunnel feature. All AppConnect apps require Mobile@Work to interact with Core.

As a third-party AppConnect app developer, MobileIron provides you access to a Core and Standalone Sentry for testing. You then use a web portal called the Admin Portal to make configuration changes necessary for testing your app.

Mobile@Work is available from the Apple App Store.

Note: Use an enterprise build of your app for testing. When your app is completely tested, build a distribution build for distributing the app through the Apple App Store. These procedures are for testing only.

Before you begin: If your app accesses servers behind a firewall using AppTunnel, contact MobileIron to provide you with a Standalone Sentry.

Set up MobileIron Core

To set up Core for testing your AppConnect app, do the following high-level steps:

1. "Login to the Admin Portal" on page 73.
2. "Enable AppConnect on MobileIron Core" on page 73.
3. "Configure the AppConnect global policy" on page 73.
4. "Create an AppConnect container policy" on page 74.

Note: These instructions are for Core 7.5.

Login to the Admin Portal

MobileIron provides you with the following information about your test MobileIron Core:

- the URL for accessing the Core's Admin Portal
  The Admin Portal is a web portal for configuring Core. The URL has the format:
  https://m.mobileiron.net/<*app partner name*>
- a user ID and password for accessing the Admin Portal
  You also use this user ID to register a device with Core.
- a port number for Core, used when you register a device with Core.
  The port number is typically four or five digits.

To login to Core:

1. Open a browser to the URL for accessing the Core's Admin Portal.
   Use the URL of your test Core, appended with /mifs. For example:
   https://m.mobileiron.net/myCompany/mifs
2. Enter your Username and Password.
3. Click Sign In.
   You are now in the Admin Portal.
   Change your password when prompted.

Enable AppConnect on MobileIron Core

To enable AppConnect on Core:

1. In the Admin Portal, go to Settings > Preferences.
2. Scroll down to Additional Products.
3. Select Enable AppConnect For Third-party And In-house Apps.
4. Click Save.

Configure the AppConnect global policy

An AppConnect global policy is necessary for your AppConnect app to work properly.

To configure an AppConnect global policy:

1. In the Admin Portal, select Policies & Configs > Policies.
2. Select the row that says Default AppConnect Global Policy for the Policy Name.
3. Click Edit in the right-hand pane.
4. For AppConnect, select Enabled.
   The display now shows all the AppConnect global policy fields.
5. In the AppConnect Passcode section, for Passcode Type, select Numeric.
6. In the AppConnect Passcode section, select Passcode Is Required For iOS Devices.
7. Click Save.

Note: Do not select Authorize in the field Apps Without An AppConnect Container Policy in the section Data Loss Prevention Policies in the AppConnect global policy. You will authorize the app with an AppConnect container policy instead.

Create an AppConnect container policy

An app is authorized only if an AppConnect container policy for the app is present on the device.

To create an AppConnect container policy:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Container Policy.
3. Enter a name for the AppConnect container policy.
   For example: My App’s Container Policy
4. In the Application field, enter the bundle ID of your app.
   For example: com.MyCompany.MySecureApp
5. Click Save.
   The dialog box closes and the new AppConnect container policy appears in the list.
6. Select the AppConnect container policy you just created.
7. Select More Actions > Apply To Label.
8. Select iOS.
9. Click Apply.
10. Click OK.

Set up your end-user device

To set up your end-user device, do the following high-level steps:

1. "Set up Mobile@Work on an iOS device" on page 75.
2. "Install your app on the device" on page 75.
3. "Set up the AppConnect passcode on the device" on page 75.

Set up Mobile@Work on an iOS device

To set up Mobile@Work for iOS on your device:

1. Download and install Mobile@Work from the Apple App Store.
2. Tap the MobileIron app icon to launch Mobile@Work.
3. Enter the user name that MobileIron gave you.

   You use the same user name that you use to log into the Admin Portal.
4. Enter the server as follows:

   m.mobileiron.net:<*port number*>

   where <*port number*> is the port number you received from MobileIron along with your user name and password.

   For example:

   m.mobileiron.net:27643
5. Enter the password.

   Enter the password that you created when you first logged into the Admin Portal.
6. Follow the prompts from Mobile@Work to complete its setup.

   Allow Mobile@Work to use the current location.

   Install new profiles and certificates when prompted.

Install your app on the device

Install your app on the device in the same way you install any app that you are testing.

Set up the AppConnect passcode on the device

When you run your app for the first time, Mobile@Work's Secure App Manager prompts you to create the AppConnect passcode. Follow the steps to create the AppConnect passcode.

Test authorization status handling

You can make changes to Core configuration to test your app's handling of the different authorization statuses: authorized, unauthorized, and retired.

Change the status to authorized or unauthorized

A security policy on Core specifies the requirements for a device. If a device is not compliant with a requirement, the security policy specifies a compliance action. One compliance action is to block AppConnect apps on the device, which means that the apps become unauthorized.

The list of requirements that can impact authorization is long, but for testing your app, you need to work with only one requirement. The requirement involves a list of device models that are not allowed to use AppConnect apps.

Therefore, to unauthorize the app on the device:

1. In the Admin Portal, select Policies & Configs > Policies.
2. Select the Default Security Policy.
3. Click Edit in the right-hand pane.
4. Scroll down to the section called Access Control, under For iOS Devices.
5. Select Block Email, AppConnect Apps, And Send Alert For The Following Disallowed Devices.
6. Move the model of your test device to the Disallowed area.
7. Click Save.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is unauthorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the unauthorized state. Specifically, verify that your app:

- exits any sensitive part of the application.
- stops allowing the user to access sensitive data and views.
- displays the message received in the callback method that explains the authorization status change.
- calls the `AppConnectCordova.authStateApplied()` method.

To re-authorize the app on the device:

1. In the Admin Portal, select Policies & Configs > Policies.

2. Select the Default Security Policy.
3. Click Edit in the right-hand pane.
4. In the section called Access Control, under For iOS Devices, *uncheck* Block Email, AppConnect Apps, And Send Alert For The Following Disallowed Devices.
5. Click Save.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is authorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the authorized state. Specifically, verify that your app:

- allows the user to access sensitive data and views.
- calls the `AppConnectCordova.authStateApplied()` method.

Change the status to retired

An app is authorized only if an AppConnect container policy for the app is present on the device. If you remove the AppConnect container policy from the device, the app becomes retired.

To retire the app on the device:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Select More Actions > Remove From Label.
4. Select iOS.
5. Click Remove.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is retired. Otherwise, it receives the event the next time it runs. The message string in the event object is the default unauthorized message:

   “Your administrator has not authorized this app.”

Verify that your app correctly handles the change to the retired state. Specifically, verify that your app:

- exits any sensitive part of the application.
- deletes all sensitive data, including any stored authentication credentials, data in files, keychain items, pasteboard data, and any other persistent storage.
- displays the message received in the event that explains the authorization status change.
- calls the `AppConnectCordova.authStateApplied()` method.

Reauthorize a retired app

A retired app is sometimes re-authorized at a later time.

To reauthorize the retired app on the device:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Select More Actions > Apply To Label.
4. Select iOS.
5. Click Apply.
6. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is authorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the authorized state. Specifically, verify that your app:

- dismisses any user interface that displays that the user is not authorized to use the app.
- allows the user to access sensitive data and views.
- calls the `AppConnectCordova.authStateApplied()` method.

Test data loss prevention policy handling

The AppConnect container policy for your app specifies its data loss prevention (DLP) policies. In this policy, you specify whether your app is allowed to:

- copy content to the iOS pasteboard.
- print by using AirPrint, any future iOS printing feature, any current or future third-party libraries or apps that provide printing capabilities.
- share documents with other apps.

By changing the AppConnect container policy, you can test:

- your app's behavior for each data loss prevention policy.
- how your app handles changes to the policies in its event handlers.

To change the DLP policies:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Click Edit in the right-hand pane.
4. Allow or prohibit features relating to data loss prevention policies as follows:

DLP policy	Description
Allow Print	Select Allow Print if you want the app to use the device's print capabilities.

DLP policy	Description
Allow Copy/Paste To	Select Allow Copy/Paste To if you want the device user to be able to copy content from the AppConnect app to other apps. When you select this option, then select either: • All Apps Select All Apps if you want the device user to be able to copy content from the AppConnect app and paste it into any other app. • AppConnect Apps This feature is not available.
Allow Open In	Select Allow Open In if you want the app to be allowed to use the device's Open In (document interaction) feature. When you select this option, then select either: • All Apps Select All Apps if you want the app to be able to send documents to any other app. • AppConnect Apps Select AppConnect Apps to allow an AppConnect app to send documents to *only* other AppConnect apps. **Note:** This option results in the `AppConnectCordova.openInPolicy()` method returning the value `WHITELIST`. Also, the `AppConnectCordova.openInWhitelist()` method contains the list of currently authorized AppConnect apps. • Whitelist Select Whitelist if you want the app to be able to send documents only to the apps that you specify. Enter the bundle ID of each app, one per line, or in a semicolon delimited list. For example: com.myAppCo.myApp1 com.myAppCo.myApp2;com.myAppCo.myApp3 The bundle IDs that you enter are case sensitive.

5. Click Save.
6. Click Yes to confirm.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the events for the updated DLP policies. Otherwise, it receives the events the next time it runs.

Verify that your app correctly handles the data loss prevention policy changes, as shown in the following table:

Policy change	What to verify
Allow copy/paste to	For each view in your app that displays secure information: • verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any user interface that offers the ability to cut or copy data is available and enabled. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Do not allow copy/ paste to	For each view in your app that displays secure information: • verify that the user cannot to cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any user interface that offers the ability to cut or copy data is removed or disabled. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Allow open in for all apps	For each part of your app that allows the user to send documents to other apps, verify that it functions normally. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for AppConnect apps	For each part of your app that allows the user to send documents to other apps, verify that the functionality works only with other AppConnect apps. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.

Policy change	What to verify
Allow open in for whitelisted apps	For each part of your app that allows the user to send secure documents to other apps: • verify it allows the user to send documents to apps that are in the whitelist • verify it does not allow the user to send documents to apps that are not in the whitelist. Also, verify that your app calls the `openInPolicyApplied()` method.
Do not allow open in	For each part of your app that normally allows the user to send secure documents to other apps, verify that the functionality is not available. Also, verify that your app calls the - `openInPolicyApplied()` method.
Allow print	For each part of your app that allows the user to print secure data, verify the capability is enabled. Also, verify that your app calls the `printPolicyApplied()` method.
Do not allow print	For each part of your app that allows the user to print secure data, verify the capability is removed or disabled. Also, verify that your app calls the `printPolicyApplied()` method.

Test AppConnect configuration change handling

AppConnect app configuration on MobileIron Core specifies key-value pairs for configuring your app. You add, and edit, key-value pairs using the Admin Portal.

By changing the AppConnect app configuration, you can test your app's event handler for the `'appconnect.configChangedTo'`event.

Create an AppConnect app configuration

To create an AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Configuration.
3. Enter a name for the AppConnect app configuration.
   For example: My App's App Configuration
4. In the Application field, enter the bundle ID of your app.
   For example: com.MyCompany.MySecureApp
5. In the App-specific Configurations section, click Add+ to add a key-value pair.

6. Enter the key-value pairs.

Key	The key is any string that the app recognizes as a configurable item. For example: userid, appURL
Value	Enter the value. The value is either: • a string The string can have any value that is meaningful to the app. It can also include one or more of these MobileIron Core variables: $USERID$, $EMAIL$, $USER_CUSTOM1$, $USER_CUSTOM2$, $USER_CUSTOM3$, $USER_CUSTOM4$. If you do not want to provide a value, enter $NULL$. The $NULL$ value tells the app that the app user will need to provide the value. Examples: $USERID$ https://someEnterpriseURL.com • a SCEP or Certificate app setting SCEP and Certificate app settings that are configured in Policies & Configs > Configurations appear in the dropdown list. When you choose a SCEP or Certificate app setting, Core sends the contents of the certificate as the value. The contents are base64-encoded. If the certificate is password-encoded, Core 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.

7. Click Save.
8. Click Yes to confirm.
9. Select the new AppConnect app configuration.
10. Select More Actions > Apply To Label.
11. Select iOS.
12. Click Apply.
13. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

If your app is running, it receives the event for the new configuration. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the new configuration, correctly applying and using the configured options according to your app's requirements and design.

Update the AppConnect app configuration

To update the AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the your app's AppConnect app configuration.
3. Click Edit in the right-hand pane.
4. In the App-specific Configurations section, click Add+ to add a key-value pair. To delete a key-value pair, click the X on the row.
5. Update the key-value pairs as described in "Create an AppConnect app configuration" on page 83.
6. Click Save.
7. Click Yes to confirm.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event for the updated configuration. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the updated configuration, correctly applying and using the configured options according to your app's requirements and design.

Test using AppTunnel

Using MobileIron's AppTunnel feature, your app can securely tunnel network connections from the app to servers behind an organization's firewall. Your app does not take any special actions related to tunneling; the AppConnect library, Mobile@Work, and a Standalone Sentry handle tunneling for the app.

You can test the tunneling capability using the provided MobileIron Core and Sentry. Using the Admin Portal, you configure app-specific AppTunnel settings for Core and Sentry.

Before you begin: Contact MobileIron to provide you with a Standalone Sentry.

To test your app's use of AppTunnel, do these high-level steps:

1. "Enable AppTunnel on MobileIron Core" on page 86.
2. Use an existing certificate or generate a new one.
   If you have an existing certificate, see "Use an existing certificate" on page 86.
   Otherwise, see "Generate a certificate" on page 87.
3. "Configure the Sentry with an AppTunnel service" on page 89.
4. "Configure the AppTunnel service in the AppConnect app configuration" on page 90.

Enable AppTunnel on MobileIron Core

To enable AppTunnel on MobileIron Core:

1. In the Admin Portal, go to Settings > Preferences.
2. Scroll down to Additional Products.
3. Select Enable AppConnect For Third-party And In-house Apps.
4. Select Enable AppTunnel For Third-party And In-house Apps.
5. Click Save.

Use an existing certificate

For your app to use AppTunnel, the Sentry needs a server certificate and the user's device needs a client certificate. If you already have an existing certificate, typically a .p12 file, you can use it.

To upload the certificate to MobileIron Core:

1. In the Admin Portal, go to Policies & Configs > Configurations.
2. Select Add New > Certificates.
3. For Name, enter any name.
   For example: Tunneling Certificate
4. For File Name, select your certificate file, typically a .p12 file.
5. For Password and Confirm Password, specify your certificate's private key, if applicable.
6. Click Save.

Generate a certificate

For your app to use AppTunnel, the Sentry needs a server certificate and the user's device needs a client certificate. You can generate these certificates using the Admin Portal.

This process involves the following high-level steps:

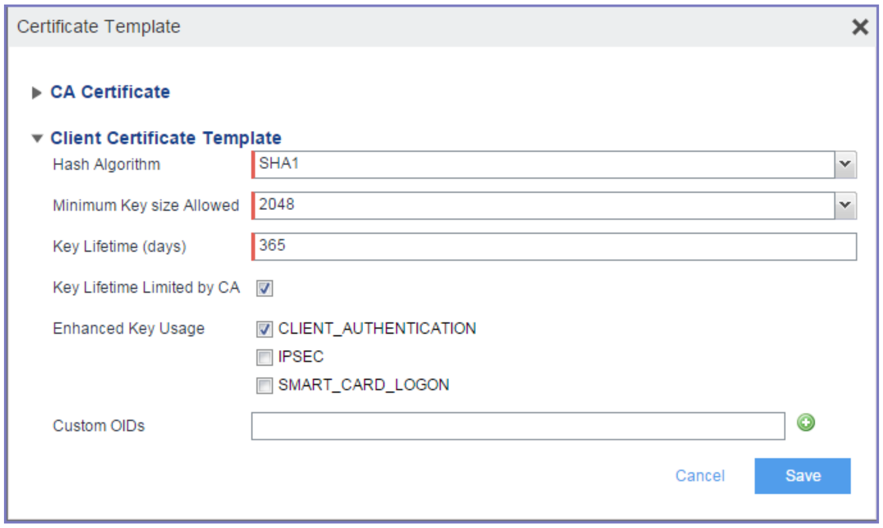
1. "Create a certificate authority for using an AppTunnel" on page 87
2. "Create a SCEP server" on page 88

Create a certificate authority for using an AppTunnel

To create a local certificate authority to be used in generating certificates:

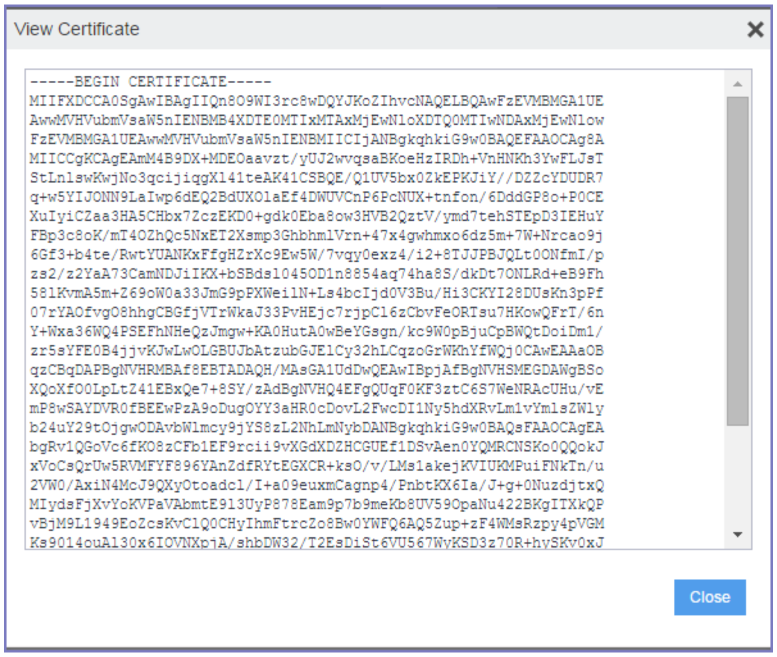
1. In the Admin Portal, select Settings > Local CA.
2. Select Add > Generate Self-Signed Cert
3. Enter a name for Local CA Name.

   For example: CA for AppTunnel
4. Set Key Length to 2048.
5. Set the Issuer Name to "CN=Tunneling CA".
6. Click Generate.



Hash Algorithm: SHA1
Minimum Key size Allowed: 2048
Key Lifetime (days): 365
Key Lifetime Limited by CA: ✓
Enhanced Key Usage: ✓ CLIENT_AUTHENTICATION; IPSEC; SMART_CARD_LOGON
Custom OIDs:

7. Click Save.
8. Click View Certificate next to your new local certificate authority.



View Certificate

```
-----BEGIN CERTIFICATE-----
MIIFXDCCA0SgAwIBAgIIQn8O9WI3rc8wDQYJKoZIhvcNAQELBQAwFzEVMBMGA1UE
AwwMVHVubmVsaW5nIENBMB4XDTE0MTIxMTAxMjEwNloXDTQ0MTIwNDAxMjEwNlow
FzEVMBMGA1UEAwwMVHVubmVsaW5nIENBMIICIjANBgkqhkiG9w0BAQEFAAOCAg8A
MIICCgKCAgEAmM4B9DX+MDEOaavzt/yUJ2wvqsaBKoeHzIRDh+VnHNKh3YwFLJsT
StLnlswKwjNo3qcijiggXl41teAK41CSBQE/Q1UV5bx0ZkEPKJiY//DZZcYDUDR7
q+w5YIJONN9LaIwp6dEQ2BdUXOlaEf4DWUVCnP6PcNUX+tnfon/6DddGP8o+P0CE
XuIyiCZaa3HA5CHbx7ZczEKD0+gdk0Eba8ow3HVB2QztV/ymd7tehSTEpD3IEHuY
FBp3c8oK/mT4OZhQc5NxET2Xsmp3GhbhmlVrn+47x4gwhmxo6dz5m+7W+Nrcao9j
6Gf3+b4te/RwtYUANKxFfgHZrXc9Ew5W/7vqy0exz4/i2+8TJJPBJQLt0ONfmI/p
zs2/z2YaA73CamNDJiIKX+bSBdsl045OD1n8854aq74ha8S/dkDt7ONLRd+eB9Fh
58lKvmA5m+Z69oW0a33JmG9pPXWeilN+Ls4bcIjd0V3Bu/Hi3CKYI28DUsKn3pPf
07rYAOfvgO8hhgCBGfjVTrWkaJ33PvHEjc7rjpCl6zCbvFeORTsu7HKowQFrT/6n
Y+Wxa36WQ4PSEFhNHeQzJmgw+KA0HutA0wBeYGsgn/kc9W0pBjuCpBWQtDoiDm1/
zr5sYFE0B4jjvKJwLwOLGBUJbAtzubGJElCy32hLCqzoGrWKhYfWQj0CAwEAAaOB
qzCBqDAPBgNVHRMBAf8EBTADAQH/MAsGA1UdDwQEAwIBpjAfBgNVHSMEGDAWgBSo
XQoXfO0LpLtZ41EBxQe7+8SY/zAdBgNVHQ4EFgQUqF0KF3ztC6S7WeNRAcUHu/vE
mP8wSAYDVR0fBEEwPzA9oDugOYY3aHR0cDovL2FwcDI1Ny5hdXRvLm1vYmlsZWly
b24uY29tOjgwODAvbWlmcy9jYS8zL2NhLmNybDANBgkqhkiG9w0BAQsFAAOCAgEA
bgRv1QGoVc6fKO8zCFb1EF9rcii9vXGdXDZHCGUEf1DSvAen0YQMRCNSKo0QQokJ
xVoCsQrUw5RVMFYF896YAnZdfRYtEGXCR+ksO/v/LMs1akejKVIUKMPuiFNkTn/u
2VW0/AxiN4McJ9QXyOtoadcl/I+a09euxmCagnp4/PnbtKX6Ia/J+g+0NuzdjtxQ
MIydsFjXvYoKVPaVAbmtE913UyP878Eam9p7b9meKb8UV59OpaNu422BKgITXkQP
vBjM9L1949EoZcsKvClQ0CHyIhmFtrcZo8Bw0YWFQ6AQ5Zup+zF4WMsRzpy4pVGM
Ks9014ouAl30x6IOVNXpjA/shbDW32/T2EsDiSt6VU567WyKSD3z70R+hySKv0xJ
```

Close

9. Copy all the text in into a text file.
10. Save the text file.

    You will upload this text file later.

Create a SCEP server

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel. One way to authenticate a device to Standalone Sentry involves using a SCEP server. The SCEP server generates certificates for the devices to present to Standalone Sentry.

To create a SCEP server:

1. In the Admin Portal, select Policies & Configs > Configurations
2. Select Add New > SCEP.
3. For the Name, enter any name for your SCEP server.

   For example: Tunneling SCEP
4. Ensure Enable Proxy is checked.
5. For Setting Type, select Local.
6. For Local CA, select the certificate authority you created for AppTunnel.
7. For Subject, enter "cn=tunneling".

   The value can be any string.
8. For Key Size, select 2048.
9. Click Issue Test Certificate.

   The issued test certificate displays.
10. Click Close to close the displayed certificate.
11. Click Save to save the SCEP setting.

12. Click OK.

Configure the Sentry with an AppTunnel service

To support AppTunnel, configure the Sentry with the internal servers that your app uses.

Do the following:

1. In the Admin Portal, go to Settings > Sentry.
2. Select Add New > Standalone Sentry.
3. Enter the host name of the Sentry that MobileIron provides you.
4. Select Enable AppTunnel.
5. For Device Authentication Configuration:

   If you already had a certificate, select Group Certificate.

   If you created a certificate authority and a SCEP server, select Identity Certificate.
6. Click Upload Certificate.

   If you already had a certificate, upload it.

   If you created a certificate authority and a SCEP server, upload the certificate text file that you created in "Create a certificate authority for using an AppTunnel" on page 87.
7. In the AppTunnel Configuration section, click + to add a new service.
8. Enter a Service Name.

   The service name is any unique identifier for the internal server or servers that your AppConnect app tunnels to. Entering <ANY> means that the app can reach any of your internal servers.

   Service Name examples:

   SharePoint

   HumanResources
9. For Server Auth, select Pass Through.

   This field selects the authentication scheme for the Standalone Sentry to use to authenticate the user to the internal server. Pass Through means that the Sentry passes through the authentication credentials, such as the user ID and password (basic authentication) or NTLM, to the internal server.

   Note: The other option is Kerberos. Kerberos means that the Sentry uses Kerberos Constrained Delegation (KCD). The corporate environment must be set up for Kerberos Constrained Delegation.
10. Enter a Server List.

    Enter a semicolon-separated list of internal server host names or IP addresses and the port that the Sentry can access.

    For example:

    sharepoint1.companyname.com:443;sharepoint2.companyname.com:443.

    When you enter multiple servers, the Sentry uses a round-robin distribution to load balance the servers. That is, it sets up the first tunnel with the first internal server, the next with the next internal server, and so on.

Note: If you selected <ANY> for the Service Name, the Server List is not applicable.

11. Select TLS Enabled if the internal servers require SSL.

    Although port 443 is typically used for https and requires SSL, the internal server can use other port numbers requiring SSL.

    Note: If you selected <ANY> for the Service Name, do not select TLS Enabled.

12. Do not fill in Server SPN List. It applies only when the Server Auth field is Kerberos.
13. Select Proxy/ATC only if your testing requires that you direct the AppTunnel service traffic through a proxy server. The proxy server is located behind the firewall and sits between the Sentry and corporate resources. This deployment allows you to access corporate resources without having to open the ports that Sentry would otherwise require.

    If selected, also configure the Server-side Proxy fields: Proxy Host Name / IP and Proxy Port.

14. Click Save.
15. Click View Certificate on the row with your new Sentry.

    This action copies the Sentry's self-signed certificate that you created to MobileIron Core.

Configure the AppTunnel service in the AppConnect app configuration

The AppConnect app configuration specifies the AppTunnel services that your app uses. You configured these services on the Sentry.

To configure AppTunnel on an AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Configuration.

   Note: If you already have created an AppConnect app configuration for your app, select it and click Edit in the right-hand pane.

3. Enter a name for the AppConnect app configuration if this is a new one.

   For example: My App's App Configuration

4. In the Application field, enter the bundle ID of your app if this is a new app configuration.

   For example: com.MyCompany.MySecureApp

5. In the AppTunnel Rules section, click Add+ to add a new AppTunnel configuration.
6. For Sentry, select the Sentry from the drop-down list.
7. For Service, select the service name from the drop-down list.

   You created this service name in "Create a certificate authority for using an AppTunnel" on page 87.

8. For the URL Wildcard, enter the host name or URL of the app server with which the app communicates. If the Service specified for this server in "Configure the Sentry with an AppTunnel service" on page 89 is <ANY>, the host name can use the wildcard character *.

If a URL request in your app matches the value you enter here, the request uses an AppTunnel.

Examples:

sharepoint1.yourcompany.com

*.yourcompanyname.com

9. For Port, enter the port number that the app connects to.
10. For Identity Certificate:

    If you already had a certificate, select the certificate app setting that you created.

    If you created a certificate authority and a SCEP server, select the SCEP setting that you created.
11. Click Save.

If you are creating a new AppConnect app configuration:

1. Select the new AppConnect app configuration.
2. Select More Actions > Apply To Label.
3. Select iOS.
4. Click Apply.
5. Click OK.

Push the changes to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

If your app is running, Mobile@Work launches and updates the AppConnect app configuration. If your app is not running, Mobile@Work launches and updates the configuration the next time that you run your app. When Mobile@Work has updated the configuration, your app will use AppTunnel for the URLs you specified.

Verify that your app’s networking capabilities work as expected.

Test the app documentation

A MobileIron Core administrator configures Core with information about your app. You provide this information in documentation about your app. The documentation includes:

- whether your app enforces the pasteboard, the print policy, and the Open In policy.
- your app’s app-specific configuration key-value pairs.
- information about internal servers that your app expects to interact with using AppTunnel.

Test whether your app correctly handles what your documentation specifies.

For more information, see “Provide documentation about your app to the MobileIron Core administrator” on page 68.

Chapter 7

Testing for in-house app developers

In-house AppConnect app testing overview

If you are a third-party app developer, see “Testing for third-party app developers” on page 71.

If you are an in-house app developer using the AppConnect for iOS Cordova Plugin to create an AppConnect app for your organization, your testing includes testing with MobileIron Core. Testing with Core is necessary to verify the AppConnect-related functionality of your AppConnect app. If your app accesses servers behind your organization’s firewall using AppTunnel, a Standalone Sentry is necessary to verify the AppTunnel feature. All AppConnect apps require Mobile@Work to interact with MobileIron Core.

As an in-house AppConnect app developer, contact your organization’s Core administrator to get access to a Core and Standalone Sentry for testing. You then use a web portal called the Admin Portal to make configuration changes necessary for testing your app.

Mobile@Work is available from the Apple App Store.

Before you begin: If your app accesses servers behind your organization’s firewall using AppTunnel, contact your Core administrator to find out the host name or IP address of the Sentry to use.

Set up MobileIron Core

To set up MobileIron Core for testing your AppConnect app, do the following high-level steps:

1. "Login to the Admin Portal" on page 95.
2. "Enable AppConnect on MobileIron Core" on page 95.
3. "Create a label for testing your app" on page 96.
4. "Upload your app to MobileIron Core if you use AppConnect.plist" on page 96.
5. "Verify your AppConnect.plist settings" on page 97.
6. "Configure the AppConnect global policy" on page 97.
7. "Create an AppConnect container policy, if necessary" on page 98.

Note: These instructions are for Core 7.5.

Login to the Admin Portal

Contact your organization's MobileIron Core administrator to get the following information about the Core to test with:

- the URL for accessing the Core's Admin Portal

  The Admin Portal is a web portal for configuring Core. It has the format:

  https://<Core domain name>/mifs
- a username and password for accessing the Admin Portal
- a username and password for registering a device with Core

  Depending on your Core administrator, this username and password can be the same as the username and password for accessing the Admin Portal.

To login to Core:

1. Open a browser to the URL for accessing the Core's Admin Portal.

   For example:

   https://myCore.mycompany.com/mifs
2. Enter your Username and Password for accessing the Admin Portal.
3. Click Sign In.

   You are now in the Admin Portal.

Enable AppConnect on MobileIron Core

To test your AppConnect app, ensure that AppConnect is enabled on MobileIron Core.

1. In the Admin Portal, go to Settings > Preferences.
2. Scroll down to Additional Products.
3. Select Enable AppConnect For Third-party And In-house Apps if it is not already selected.
4. Click Save.

Create a label for testing your app

MobileIron Core uses labels to associate policies and apps with devices. For testing your app, create a new label so that your testing impacts only your test device.

1. In the Admin Portal, go to Users & Devices > Labels.
2. Click Add Label.
3. Enter a name for the label.
   For example: AppConnect Test
4. Enter a description.
   For example: Use only for devices testing new AppConnect apps.
5. Click Save.

Upload your app to MobileIron Core if you use AppConnect.plist

If your app uses an AppConnect.plist, upload your app to MobileIron Core. Uploading your app causes Core to create and populate an AppConnect container policy and AppConnect app configuration with the values you entered in the AppConnect.plist.

To upload your app:

1. In the Admin Portal, select Apps > App Distribution Library.
2. Select iOS from the Select Platform list.
3. Click Add App.
   The iOS Add App Wizard starts.
4. Click Next.
5. Click Browse to select your app’s .ipa file.
6. Click Next.
7. Click Next.
8. Upload an icon for the app.
9. Click Next.
10. Click Finish.
    The app is now in Core’s app distribution library. Core has created an AppConnect container policy and AppConnect app configuration based on your AppConnect.plist.
11. Select the row listing your app.
12. Select Actions > Apply To Label.
13. Select the label that your created in “Create a label for testing your app” on page 96.
14. Click Apply.
15. Click OK.
    Core applies the label to your app. It also applies it to the AppConnect container policy and AppConnect app configuration.

Verify your AppConnect.plist settings

Once you have uploaded your app to MobileIron Core, verify that the AppConnect.plist settings are correctly reflected in the AppConnect container policy and AppConnect app configuration.

To verify the AppConnect.plist settings:

1. On the Admin Portal, go to Policies & Configs > Configurations.
2. Select the row with the name of your app and the Setting Type APPCONFIG.
3. Click Edit in the right-hand pane.
4. In the App-specific Configurations section, verify the keys and values are what you entered in the AppConnect.plist.
5. Click Cancel.
6. Select the row with the name of your app and the Setting Type APPPOLICY.
7. Click Edit in the right-hand pane.
8. Verify the data loss protection settings are what you entered in the AppConnect.plist.
9. Click Cancel.

If any of the key-value pairs or data loss prevention policies are not what you expected, review the contents of your AppConnect.plist.

Configure the AppConnect global policy

An AppConnect global policy is necessary for your AppConnect app to work properly.

To configure an AppConnect global policy:

1. In the Admin Portal, select Policies & Configs > Policies.
2. Select Add New > AppConnect.
3. Enter a name for the AppConnect Global Policy.
   For example: Test AppConnect Global Policy.
4. For AppConnect, select Enabled.
   The display now shows all the AppConnect global policy fields.
5. In the AppConnect Passcode section, for Passcode Type, select Numeric.
6. In the AppConnect Passcode section, select Passcode Is Required For iOS Devices.
7. Click Save.
   The dialog box closes and the new AppConnect global policy appears in the list.
8. Select the AppConnect global policy that you just created.
9. Select More Actions > Apply To Label.
10. Select the test label that you created in “Create a label for testing your app” on page 96.
11. Click Apply.
12. Click OK.

Note: Do not select Authorize in the field Apps Without An AppConnect Container Policy in the section Data Loss Prevention Policies in the AppConnect global policy. You will authorize the app with an AppConnect container policy instead.

Create an AppConnect container policy, if necessary

An app is authorized only if an AppConnect container policy for the app is present on the device. If you have an AppConnect.plist in your app, and uploaded the app to MobileIron Core, Core creates an AppConnect container policy automatically. If you do not have an AppConnect.plist in your app, manually create an AppConnect container policy.

To create an AppConnect container policy:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Container Policy.
3. Enter a name for the AppConnect container policy.

   For example: My App's Container Policy
4. In the Application field, enter the bundle ID of your app.

   For example: com.MyCompany.MySecureApp
5. Click Save.

   The dialog box closes and the new AppConnect container policy appears in the list.
6. Select the AppConnect container policy you just created.
7. Select More Actions > Apply To Label.
8. Select the test label that you created in "Create a label for testing your app" on page 96.
9. Click Apply.
10. Click OK.

Set up your end-user device

To set up your end-user device, do the following high-level steps:

1. "Set up Mobile@Work on an iOS device" on page 99.
2. "Install your app on the device" on page 99.
3. "Set up the AppConnect passcode on the device" on page 99.

Set up Mobile@Work on an iOS device

To set up Mobile@Work for iOS on your device:

1. Download and install Mobile@Work from the Apple App Store.
2. Tap the MobileIron app icon to launch Mobile@Work.
3. Enter the user name that the Core administrator gave you for registering your test device.
4. Enter the server name that the Core administrator gave you.

   For example: myCore.mycompany.com
5. Enter the password.

   Enter the password that the Core administrator gave you for registering your test device.
6. Follow the prompts from Mobile@Work to complete its setup.

   Allow Mobile@Work to use the current location.

   Install new profiles and certificates when prompted.

Install your app on the device

Install your app on the device in the same way you install any app that you are testing.

Set up the AppConnect passcode on the device

When you run your app for the first time, Mobile@Work's Secure App Manager prompts you to create the AppConnect passcode. Follow the steps to create the AppConnect passcode.

Test authorization status handling

You can make changes to the MobileIron Core configuration to test your app's handling of the different authorization statuses: authorized, unauthorized, and retired.

Change the status to authorized or unauthorized

A security policy on MobileIron Core specifies the requirements for a device. If a device is not compliant with a requirement, the security policy specifies a compliance action. One compliance action is to block AppConnect apps on the device, which means that the apps become unauthorized.

The list of requirements that can impact authorization is long, but for testing your app, you need to work with only one requirement. The requirement involves a list of device models that are not allowed to use AppConnect apps.

Therefore, to unauthorize the app on the device:

1. In the Admin Portal, select Policies & Configs > Policies.
2. Select Add New > Security.
3. Enter a name.
   For example: AppConnect test security policy
4. Scroll down to the section called Access Control, under For iOS Devices.
5. Select Block Email, AppConnect Apps, And Send Alert For The Following Disallowed Devices.
6. Move the model of your test device to the Disallowed area.
7. Click Save.
   Core creates the new security policy.
8. Select the row listing the new security policy.
9. Select More Actions > Apply To Label.
10. Select the test label that you created in "Create a label for testing your app" on page 96.
11. Click Apply.
12. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.
   If your app is running, it receives the event that it is unauthorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the unauthorized state. Specifically, verify that your app:

- exits any sensitive part of the application.
- stops allowing the user to access sensitive data and views.
- displays the message received in the callback method that explains the authorization status change.
- calls the `AppConnectCordova.authStateApplied()` method.

To re-authorize the app on the device:

1. In the Admin Portal, select Policies & Configs > Policies.
2. Select the security policy that you created.
3. Click Edit in the right-hand pane.
4. In the section called Access Control, under For iOS Devices, *uncheck* Block Email, AppConnect Apps, And Send Alert For The Following Disallowed Devices.
5. Click Save.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is authorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the authorized state. Specifically, verify that your app:

- allows the user to access sensitive data and views.
- calls the `AppConnectCordova.authStateApplied()` method.

Change the status to retired

An app is authorized only if an AppConnect container policy for the app is present on the device. If you remove the AppConnect container policy from the device, the app becomes retired.

To retire the app on the device:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Select More Actions > Remove From Label.
4. Select the label that you created in “Create a label for testing your app” on page 96.
5. Click Remove.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.

2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is retired. Otherwise, it receives the event the next time it runs. The message string in the notification is the default unauthorized message:

   “Your administrator has not authorized this app.”

Verify that your app correctly handles the change to the retired state. Specifically, verify that your app:

- exits any sensitive part of the application.
- deletes all sensitive data, including any stored authentication credentials, data in files, keychain items, pasteboard data, and any other persistent storage.
- displays the message received in the callback method that explains the authorization status change.
- calls the `AppConnectCordova.authStateApplied()` method.

Reauthorize a retired app

A retired app is sometimes re-authorized at a later time.

To reauthorize the retired app on the device:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Select More Actions > Apply To Label.
4. Select the label that you created in “Create a label for testing your app” on page 96.
5. Click Apply.
6. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event that it is authorized. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the change to the authorized state. Specifically, verify that your app:

- dismisses any user interface that displays that the user is not authorized to use the app.
- allows the user to access sensitive data and views.
- calls the `AppConnectCordova.authStateApplied()` method.

Test data loss prevention policy handling

The AppConnect container policy for your app specifies its data loss prevention (DLP) policies. In this policy, you specify whether your app is allowed to:

- copy content to the iOS pasteboard.
- print by using AirPrint, any future iOS printing feature, any current or future third-party libraries or apps that provide printing capabilities.
- share documents with other apps.

By changing the AppConnect container policy, you can test:

- your app's behavior for each data loss prevention policy.
- how your app handles changes to the policies in its event handlers.

To change the DLP policies:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the AppConnect container policy for your app.
3. Click Edit in the right-hand pane.
4. Allow or prohibit features relating to data loss prevention policies as follows:

DLP policy	Description
Allow Print	Select Allow Print if you want the app to use the device's print capabilities.

DLP policy	Description
Allow Copy/Paste to	Select Allow Copy/Paste to if you want the device user to be able to copy content from the AppConnect app to other apps. When you select this option, then select either: • All Apps Select All Apps if you want the device user to be able to copy content from the AppConnect app and paste it into any other app. • AppConnect Apps This feature is not available.
Allow Open In	Select Allow Open In if you want the app to be allowed to use the device's Open In (document interaction) feature. When you select this option, then select either: • All Apps Select All Apps if you want the app to be able to send documents to any other app. • AppConnect Apps Select AppConnect Apps to allow an AppConnect app to send documents to *only* other AppConnect apps. **Note:** This option results in the `AppConnectCordova.openInPolicy()` method returning the value `WHITELIST`. Also, the `AppConnectCordova.openInWhitelist()` method contains the list of currently authorized AppConnect apps. • Whitelist Select Whitelist if you want the app to be able to send documents only to the apps that you specify. Enter the bundle ID of each app, one per line, or in a semicolon delimited list. For example: com.myAppCo.myApp1 com.myAppCo.myApp2;com.myAppCo.myApp3 The bundle IDs that you enter are case sensitive.

5. Click Save.
6. Click Yes to confirm.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

If your app is running, it receives the events for the updated DLP policies. Otherwise, it receives the events the next time it runs.

Verify that your app correctly handles the data loss prevention policy changes, as shown in the following table:

Policy change	What to verify
Allow copy/paste to	For each view in your app that displays secure information: • verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any user interface that offers the ability to cut or copy data is available and enabled. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Do not allow copy/ paste to	For each view in your app that displays secure information: • verify that the user cannot to cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any user interface that offers the ability to cut or copy data is removed or disabled. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Allow open in for all apps	For each part of your app that allows the user to send documents to other apps, verify that it functions normally. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for AppConnect apps	For each part of your app that allows the user to send documents to other apps, verify that the functionality works only with other AppConnect apps. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.

Policy change	What to verify
Allow open in for whitelisted apps	For each part of your app that allows the user to send secure documents to other apps: • verify it allows the user to send documents to apps that are in the whitelist • verify it does not allow the user to send documents to apps that are not in the whitelist. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Do not allow open in	For each part of your app that normally allows the user to send secure documents to other apps, verify that the functionality is not available. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow print	For each part of your app that allows the user to print secure data, verify the capability is enabled. Also, verify that your app calls the `AppConnectCordova.printPolicyApplied()` method.
Do not allow print	For each part of your app that allows the user to print secure data, verify the capability is removed or disabled. Also, verify that your app calls the `AppConnectCordova.printPolicyApplied()` method.

Test AppConnect configuration change handling

AppConnect app configuration on MobileIron Core specifies key-value pairs for configuring your app. You add, and edit, key-value pairs using the Admin Portal.

By changing the AppConnect app configuration, you can test your app’s event handler for the `'appconnect.configChangedTo'`event.

If your app includes an AppConnect.plist, and you uploaded your app to Core, Core already has created a default AppConnect app configuration. Go to “Update the AppConnect app configuration” on page 109.

If your app does not include an AppConnect.plist, create an AppConnect app configuration.

Create an AppConnect app configuration

To create an AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Configuration.
3. Enter a name for the AppConnect app configuration.
   For example: My App’s App Configuration
4. In the Application field, enter the bundle ID of your app.
   For example: com.MyCompany.MySecureApp
5. In the App-specific Configurations section, click Add+ to add a key-value pair.

6. Enter the key-value pairs.

Key	The key is any string that the app recognizes as a configurable item. For example: userid, appURL
Value	Enter the value. The value is either: • a string The string can have any value that is meaningful to the app. It can also include one or more of these MobileIron Core variables: $USERID$, $EMAIL$, $USER_CUSTOM1$, $USER_CUSTOM2$, $USER_CUSTOM3$, $USER_CUSTOM4$. If you do not want to provide a value, enter $NULL$. The $NULL$ value tells the app that the app user will need to provide the value. Examples: $USERID$ https://someEnterpriseURL.com • a SCEP or Certificate app setting SCEP and Certificate app settings that are configured in Policies & Configs > Configurations appear in the dropdown list. When you choose a SCEP or Certificate app setting, Core sends the contents of the certificate as the value. The contents are base64-encoded. If the certificate is password-encoded, Core 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.

7. Click Save.
8. Click Yes to confirm.
9. Select the new AppConnect app configuration.
10. Select More Actions > Apply To Label.
11. Select the label that you created in "Create a label for testing your app" on page 96.
12. Click Apply.
13. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

If your app is running, it receives the event for the new configuration. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the new configuration, correctly applying and using the configured options according to your app’s requirements and design.

Update the AppConnect app configuration

To update the AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select the your app’s AppConnect app configuration.
3. Click Edit in the right-hand pane.
4. In the App-specific Configurations section, click Add+ to add a key-value pair. To delete a key-value pair, click the X on the row.
5. Update the key-value pairs as described in “Create an AppConnect app configuration” on page 107.
6. Click Save.
7. Click Yes to confirm.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

   If your app is running, it receives the event for the updated configuration. Otherwise, it receives the event the next time it runs.

Verify that your app correctly handles the updated configuration, correctly applying and using the configured options according to your app’s requirements and design.

Test using AppTunnel

Using MobileIron's AppTunnel feature, your app can securely tunnel network connections from the app to servers behind an organization's firewall. Your app does not take any special actions related to tunneling; the AppConnect library, Mobile@Work, and a Standalone Sentry handle tunneling for the app.

You can test the tunneling capability using the provided MobileIron Core and Sentry. Using the Admin Portal, you configure app-specific AppTunnel settings for Core and Sentry.

Before you begin: Contact your Core administrator to find out the host name or IP address of the Sentry to use for the AppTunnel feature.

To test your app's use of AppTunnel, do these high-level steps:

1. "Enable AppTunnel on MobileIron Core" on page 110.
2. Use an existing certificate or generate a new one.
   If you have an existing certificate, see "Use an existing certificate" on page 110.
   Otherwise, see "Generate a certificate" on page 111.
3. "Configure the Sentry with an AppTunnel service" on page 113.
4. "Configure the AppTunnel service in the AppConnect app configuration" on page 114.

Enable AppTunnel on MobileIron Core

To enable AppTunnel on MobileIron Core if it isn't already enabled:

1. In the Admin Portal, go to Settings > Preferences.
2. Scroll down to Additional Products.
3. Select Enable AppConnect For Third-party And In-house Apps if it isn't already selected.
4. Select Enable AppTunnel For Third-party And In-house Apps if it isn't already selected.
5. Click Save.

Use an existing certificate

For your app to use AppTunnel, the Sentry needs a server certificate and the user's device needs a client certificate. If you already have an existing certificate, typically a .p12 file, you can use it.

To upload the certificate to MobileIron Core:

1. In the Admin Portal, go to Policies & Configs > Configurations.
2. Select Add New > Certificates.
3. For Name, enter any name.
   For example: Tunneling Certificate
4. For File Name, select your certificate file, typically a .p12 file.

5. For Password and Confirm Password, specify your certificate’s private key, if applicable.
6. Click Save.

Generate a certificate

For your app to use AppTunnel, the Sentry needs a server certificate and the user’s device needs a client certificate. You can generate these certificates using the Admin Portal.

This process involves the following high-level steps:

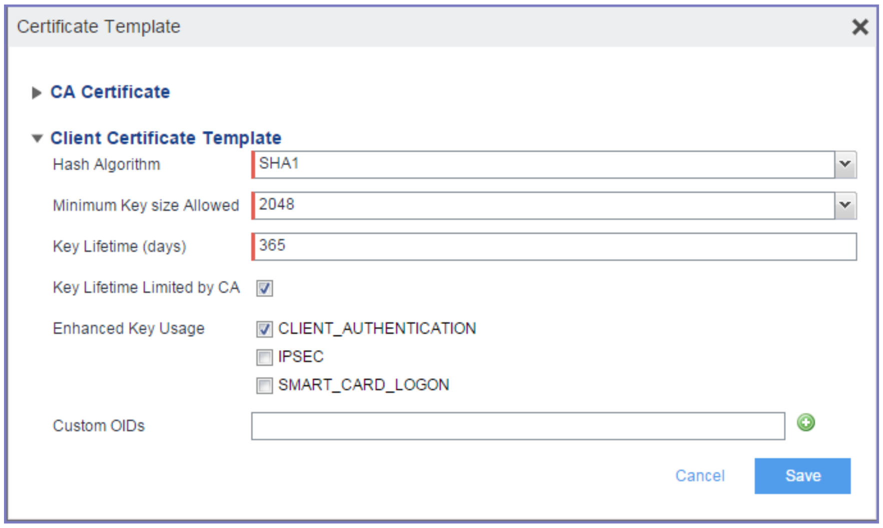
1. “Create a certificate authority for using an AppTunnel” on page 111
2. “Create a SCEP server” on page 112

Create a certificate authority for using an AppTunnel

To create a local certificate authority to be used in generating certificates:

1. In the Admin Portal, select Settings > Local CA.
2. Select Add > Generate Self-Signed Cert
3. Enter a name for Local CA Name.

   For example: CA for AppTunnel
4. Set Key Length to 2048.
5. Set the Issuer Name to “CN=Tunneling CA”.
6. Click Generate.



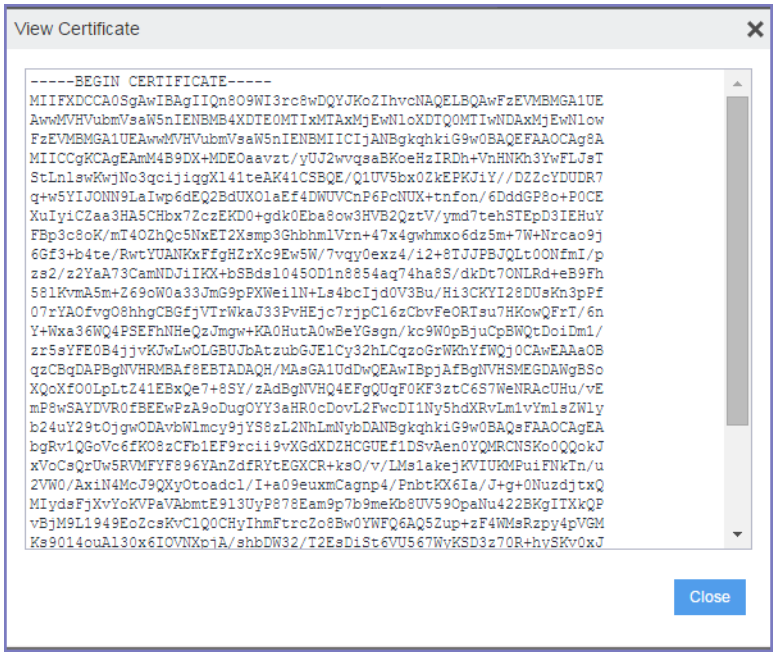
Certificate Template

▶ CA Certificate

▼ Client Certificate Template

Hash Algorithm	SHA1
Minimum Key size Allowed	2048
Key Lifetime (days)	365
Key Lifetime Limited by CA	✓
Enhanced Key Usage	✓ CLIENT_AUTHENTICATION ☐ IPSEC ☐ SMART_CARD_LOGON
Custom OIDs	

7. Click Save.
8. Click View Certificate next to your new local certificate authority.


View Certificate

```
-----BEGIN CERTIFICATE-----
MIIFXDCCA0SgAwIBAgIIQn8O9WI3rc8wDQYJKoZIhvcNAQELBQAwFzEVMBMGA1UE
AwwMVHVubmVsaW5nIENBMB4XDTE0MTIxMTAxMjEwNloXDTQ0MTIwNDAxMjEwNlow
FzEVMBMGA1UEAwwMVHVubmVsaW5nIENBMIICIjANBgkqhkiG9w0BAQEFAAOCAg8A
MIICCgKCAgEAmM4B9DX+MDEOaavzt/yUJ2wvqsaBKoeHzIRDh+VnHNKh3YwFLJsT
StLnlswKwjNo3qcijiqgXl41teAK41CSBQE/Q1UV5bx0ZkEPKJiY//DZZcYDUDR7
q+w5YIJONN9LaIwp6dEQ2BdUXOlaEf4DWUVCnP6PcNUX+tnfon/6DddGP8o+P0CE
XuIyiCZaa3HA5CHbx7ZczEKD0+gdk0Eba8ow3HVB2QztV/ymd7tehSTEpD3IEHuY
FBp3c8oK/mT4OZhQc5NxET2Xsmp3GhbhmlVrn+47x4gwhmxo6dz5m+7W+Nrcao9j
6Gf3+b4te/RwtYUANKxFfgHZrXc9Ew5W/7vqy0exz4/i2+8TJJPBJQLt0ONfmI/p
zs2/z2YaA73CamNDJiIKX+bSBdsl045OD1n8854aq74ha8S/dkDt7ONLRd+eB9Fh
58lKvmA5m+Z69oW0a33JmG9pPXWeilN+Ls4bcIjd0V3Bu/Hi3CKYI28DUsKn3pPf
07rYAOfvgO8hhgCBGfjVTrWkaJ33PvHEjc7rjpCl6zCbvFeORTsu7HKowQFrT/6n
Y+Wxa36WQ4PSEFhNHeQzJmgw+KA0HutA0wBeYGsgn/kc9W0pBjuCpBWQtDoiDm1/
zr5sYFE0B4jjvKJwLwOLGBUJbAtzubGJElCy32hLCqzoGrWKhYfWQj0CAwEAAaOB
qzCBqDAPBgNVHRMBAf8EBTADAQH/MAsGA1UdDwQEAwIBpjAfBgNVHSMEGDAWgBSo
XQoXfO0LpLtZ41EBxQe7+8SY/zAdBgNVHQ4EFgQUqF0KF3ztC6S7WeNRAcUHu/vE
mP8wSAYDVR0fBEEwPzA9oDugOYY3aHR0cDovL2FwcDI1Ny5hdXRvLm1vYmlsZWly
b24uY29tOjgwODAvbWlmcy9jYS8zL2NhLmNybDANBgkqhkiG9w0BAQsFAAOCAgEA
bgRv1QGoVc6fKO8zCFb1EF9rcii9vXGdXDZHCGUEf1DSvAen0YQMRCNSKo0QQokJ
xVoCsQrUw5RVMFYF896YAnZdfRYtEGXCR+ksO/v/LMs1akejKVIUKMPuiFNkTn/u
2VW0/AxiN4McJ9QXyOtoadcl/I+a09euxmCagnp4/PnbtKX6Ia/J+g+0NuzdjtxQ
MIydsFjXvYoKVPaVAbmtE913UyP878Eam9p7b9meKb8UV59OpaNu422BKgITXkQP
vBjM9L1949EoZcsKvClQ0CHyIhmFtrcZo8Bw0YWFQ6AQ5Zup+zF4WMsRzpy4pVGM
Ks9014ouAl30x6IOVNXpjA/shbDW32/T2EsDiSt6VU567WyKSD3z70R+hySKv0xJ
```

9. Copy all the text into a text file.
10. Save the text file.

    You will upload this text file later.

Create a SCEP server

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel. One way to authenticate a device to Standalone Sentry involves using a SCEP server. The SCEP server generates certificates for the devices to present to Standalone Sentry.

To create a SCEP server:

1. In the Admin Portal, select Policies & Configs > Configurations
2. Select Add New > SCEP.
3. For the Name, enter any name for your SCEP server.

   For example: Tunneling SCEP
4. Ensure Enable Proxy is checked.
5. For Setting Type, select Local.
6. For Local CA, select the certificate authority you created for AppTunnel.
7. For Subject, enter "cn=tunneling".

   The value can be any string.
8. For Key Size, select 2048.
9. Click Issue Test Certificate.

   The issued test certificate displays.
10. Click Close to close the displayed certificate.
11. Click Save to save the SCEP setting.

12. Click OK.

Configure the Sentry with an AppTunnel service

To support AppTunnel, configure the Sentry with the internal servers that your app uses.

Do the following:

1. In the Admin Portal, go to Settings > Sentry.
2. Click the edit icon next to the Sentry that your MobileIron Core Administrator has designated for your AppTunnel testing.
3. Select Enable AppTunnel if it is not already selected.
4. For Device Authentication Configuration:

   If you already had a certificate, select Group Certificate.

   If you created a certificate authority and a SCEP server, select Identity Certificate.
5. Click Upload Certificate.

   If you already had a certificate, upload it.

   If you created a certificate authority and a SCEP server, upload the certificate text file that you created in "Create a certificate authority for using an AppTunnel" on page 111.
6. In the AppTunnel Configuration section, click + to add a new service.
7. Enter a Service Name.

   The service name is any unique identifier for the internal server or servers that your AppConnect app tunnels to. Entering <ANY> means that the app can reach any of your internal servers.

   Service Name examples:

   SharePoint

   HumanResources
8. For Server Auth, select Pass Through.

   This field selects the authentication scheme for the Standalone Sentry to use to authenticate the user to the internal server. Pass Through means that the Sentry passes through the authentication credentials, such as the user ID and password (basic authentication) or NTLM, to the internal server.

   Note: The other option is Kerberos. Kerberos means that the Sentry uses Kerberos Constrained Delegation (KCD). The corporate environment must be set up for Kerberos Constrained Delegation.
9. Enter a Server List.

   Enter a semicolon-separated list of internal server host names or IP addresses and the port that the Sentry can access.

   For example:

   sharepoint1.companyname.com:443;sharepoint2.companyname.com:443.

   When you enter multiple servers, the Sentry uses a round-robin distribution to load balance the servers. That is, it sets up the first tunnel with the first internal server, the next with the next internal server, and so on.

Note: If you selected <ANY> for the Service Name, the Server List is not applicable.

10. Select TLS Enabled if the internal servers require SSL.

    Although port 443 is typically used for https and requires SSL, the internal server can use other port numbers requiring SSL.

    Note: If you selected <ANY> for the Service Name, do not select TLS Enabled.

11. Do not fill in Server SPN List. It applies only when the Server Auth field is Kerberos.
12. Select Proxy/ATC only if your testing requires that you direct the AppTunnel service traffic through a proxy server. The proxy server is located behind the firewall and sits between the Sentry and corporate resources. This deployment allows you to access corporate resources without having to open the ports that Sentry would otherwise require.

    If selected, also configure the Server-side Proxy fields: Proxy Host Name / IP and Proxy Port.

13. Click Save.
14. Click View Certificate on the row with your new Sentry.

    This action copies the Sentry’s self-signed certificate that you created to Core.

Configure the AppTunnel service in the AppConnect app configuration

The AppConnect app configuration specifies the AppTunnel services that your app uses. You configured these services on the Sentry.

To configure AppTunnel on an AppConnect app configuration:

1. In the Admin Portal, select Policies & Configs > Configurations.
2. Select Add New > AppConnect > Configuration.

   Note: If you already have an AppConnect app configuration for your app, select it and click Edit in the right-hand pane.

3. Enter a name for the AppConnect app configuration if this is a new one.

   For example: My App’s App Configuration

4. In the Application field, enter the bundle ID of your app if this is a new app configuration.

   For example: com.MyCompany.MySecureApp

5. In the AppTunnel Rules section, click Add+ to add a new AppTunnel configuration.
6. For Sentry, select the Sentry from the drop-down list.
7. For Service, select the service name from the drop-down list.

   You created this service name in “Create a certificate authority for using an AppTunnel” on page 111.

8. For the URL Wildcard, enter the host name or URL of the app server with which the app communicates. If the Service specified for this server in “Configure the Sentry with an AppTunnel service” on page 113 is <ANY>, the host name can use the wildcard character *.

   If a URL request in your app matches the value you enter here, the request uses an AppTunnel.

Examples:

sharepoint1.yourcompany.com

*.yourcompanyname.com

9. For Port, enter the port number that the app connects to.
10. For Identity Certificate:

    If you already had a certificate, select the certificate app setting that you created.

    If you created a certificate authority and a SCEP server, select the SCEP setting that you created.
11. Click Save.

If you are creating a new AppConnect app configuration:

1. Select the new AppConnect app configuration.
2. Select More Actions > Apply To Label.
3. Select the label that you created in "Create a label for testing your app" on page 96.
4. Click Apply.
5. Click OK.

Push the change to your device immediately, by doing the following steps on the device:

1. Launch Mobile@Work.
2. Tap Settings.
3. Tap Check for Updates.
4. Tap Force Device Check-in.

If app is running, Mobile@Work launches and updates the AppConnect app configuration. If your app is not running, Mobile@Work launches and updates the configuration the next time that you run your app. When Mobile@Work has updated the configuration, your app will use AppTunnel for the URLs you specified.

Verify that your app's networking capabilities work as expected.

Test the app documentation

Once your app is ready for in-house distribution, a MobileIron Core administrator configures Core with information about your app. You provide this information in documentation about your app. The documentation includes:

- whether your app enforces the pasteboard, the print policy, and the Open In policy.
- your app's app-specific configuration key-value pairs.
- information about internal servers that your app expects to interact with using AppTunnel.

Test whether your app correctly handles what your documentation specifies.

For more information, see "Provide documentation about your app to the MobileIron Core administrator" on page 68.

Chapter 8

AppConnect for iOS Cordova Plugin revision history

AppConnect for iOS Cordova Plugin version 2.1

New features

AppConnect for iOS Cordova Plugin version 2.1 adds these new features:

- Touch ID support for accessing AppConnect apps
- Certificate authentication to enterprise services

Touch ID support for accessing AppConnect apps

Mobile@Work 6.3 and MobileIron Core 7.5.1 allow the device user to enter his Touch ID (fingerprint) instead of the secure apps passcode to access secure apps. Apps built with this version of the AppConnect for iOS Cordova Plugin prompt for the Touch ID themselves, without switching to Mobile@Work, in the following cases:

- The device user launched or switched to an AppConnect app after the auto-lock timeout expired.
- The auto-lock timeout expired while the device user was running an AppConnect app.

Because the Touch ID prompt does not cover the entire screen, an app must make sure that sensitive data on the app’s screen is not readable until the device user enters the Touch ID. Hide sensitive data according to the app’s requirements and user interface strategies. See “Remove secure data from views before moving to the background” on page 67.

Certificate authentication to enterprise services

Now, without any development, 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 Core administrator configures on Core 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.

Your app takes no action at all.

This feature requires Mobile@Work 6.3.

Resolved issues

The following issue was resolved in AppConnect for iOS Cordova Plugin version 2.1:

- AP-2664, CE-5491: When using AppTunnel, tunneling did not occur if any characters of the Sentry host name were not the same case as the Common Name (CN) field of the certificate used for tunneling. The issue has been fixed.

Usage notes

- AP-2666: Consider the case when you use the keys MI_AC_CLIENT_CERT_# and MI_AC_CLIENT_CERT_#_RULE in an AppConnect app configuration to setup certificate authentication from an AppConnect app to an enterprise service. The AppConnect library within the AppConnect app handles these keys and their values. On the MobileIron Core Admin Portal, the status of the AppConnect app configuration for a device does not necessarily correctly reflect whether the AppConnect library successfully applied the key-value pairs.

- AP-2559: Consider the case when the device user is prompted to enter the Touch ID to access secure apps. Then the user taps Enter Passcode, and then enters the wrong passcode multiple times until iOS forces the user to wait a minute before trying again. Then the user taps Cancel on the passcode prompt. The next time the user attempts to access a secure app, the user is presented the device passcode prompt, instead of the Touch ID prompt.
- iOS-5849: Normally, tapping Cancel on the Touch ID prompt when trying to access AppConnect apps takes the device user to Mobile@Work. However, if Mobile@Work had been stopped and then relaunched, tapping Cancel on the Touch ID prompt causes Mobile@Work to exit. To the device user, this behavior looks like Mobile@Work crashed.
- APG-670: If an AppConnect app does not support iPhone 6 screen resolution, the Touch ID prompt for accessing AppConnect apps results in two overlaid status bars.

AppConnect for iOS Cordova Plugin version 2.0.2

This release has no new features.

Resolved issues

The following issue was resolved in AppConnect for iOS Cordova Plugin version 2.0.2:

- AP-2471: When an AppConnect app made a URL request which used AppTunnel, an issue occurred in some cases when the request was redirected. Specifically, AppTunnel did not correctly handle the redirection request if it contained certain special characters such as % in its URL query parameters. This issue, for example, caused SAML authentication to fail. The issue occurred in AppConnect for iOS Cordova Plugin version 2.0 and 2.0.1. The issue has been fixed.

AppConnect for iOS Cordova Plugin version 2.0.1

This release has no new features.

Resolved issues

The following issue was resolved in AppConnect for iOS Cordova Plugin version 2.0.1:

- AP-2434: A certificate handling issue has been fixed.

AppConnect for iOS Cordova Plugin version 2.0

AppConnect for iOS Cordova Plugin version 2.0 is the first version of the plugin. Its version number is 2.0 because it uses the AppConnect for iOS SDK 2.0. The plugin allows Cordova apps to behave as AppConnect apps, with the advantages and control that the AppConnect for iOS SDK provides.

Known issues

The following known issue is in the AppConnect for iOS Cordova Plugin version 2.0:

- AP-2414: In some cases, when the first AppConnect app on a device checks in for first time, the check in fails. The result is that the app does not become authorized or receive its policies and app-specific configuration.

  Workaround: Relaunch the app.