MobileIron AppConnect 4.7.0 for iOS Cordova Plugin Developers Guide

October 20, 2020

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

Contents

Contents **3**

New features and enhancements **13**

Introducing the MobileIron AppConnect for iOS Cordova Plugin **14**

AppConnect for iOS overview 14

Where to get the AppConnect for iOS Cordova Plugin 15

Secure app features 15

AppConnect for iOS Cordova Plugin advantages 16

64-bit and 32-bit app support 16

MobileIron AppConnect components 16

Using a secure app 18

App responsibilities 18

The MobileIron client app and AppConnect library responsibilities 18

Cordova Plugin variants 18

AppConnect Cordova Plugin contents 19

AppConnect for iOS architecture 19

The MobileIron client app and AppConnect apps 21

App checkin and the MobileIron client app 21

The AppConnect passcode auto-lock time and the MobileIron client app 22

Product versions required 22

Securing and managing the app using the AppConnect library 23

Authorization 24

AppConnect passcode and Touch ID/Face ID policy 24

Configuration specific to the app 25

AppTunnel 25

Supported APIs 26

AppTunnel with TCP tunneling 26

Certificate authentication to enterprise services 26
Supported networking methods 26
Unsupported networking methods 27
Data loss prevention policies 27
Custom keyboard control 28
Data protection 28
Optional: Avoiding pasteboard notifications 28
Configuring an App Group on the Apple Developer portal 29
Add the App Group to Info.plist 30
Getting started with the AppConnect for iOS Cordova Plugin 31
Upgrade tasks 31
Getting started tasks 31
Before you begin 32
Getting started task list 32
Run the AppConnect Cordova Plugin installation script 32
Declare the AppConnect URL schemes as allowed 33
Add AppConnect-related entries to your Info.plist 34
Enable screen blurring 34
Allow Face ID 34
Update Xcode project settings 34
Initialize the AppConnect library 35
Wait for the AppConnect library to be ready 35
Specify app permissions and configuration in a plist file 35
Code changes if you manually recreate the iOS platform directory 38
Troubleshooting 39
App crashes due to not waiting for AppConnect ready event 39
Problem 39
Solution 39
AppConnect for iOS Cordova Plugin API 40

AppConnect for iOS Cordova Plugin overview 40
Dual-mode app capabilities 40
The AppConnectCordova JavaScript interface 41
Event handling overview 41
AppConnect Cordova Plugin events 42
Event handling acknowledgments 43
AppConnect ready API details 43
The 'appConnect.isReady' event 43
The isReady() method 43
Event handler for 'appConnect.isReady' event 44
Authorization API details 44
The ACAuthState enumeration 45
The authState() and authMessage() methods 45
The authState() method 45
The authMessage() method 45
Calling authState() and authMessage() when your app launches 45
Method return values after updates to authorization status 46
The 'appconnect.authStateChangedTo' event 46
Event handler for 'appConnect.authStateChangedTo' event 46
The authStateApplied() method 47
The displayMessage() method 47
App-specific configuration API details 48
The config() method 48
Calling config() when your app launches 48
config() return value after updates to app-specific configuration 48
The 'appconnect.configChangedTo' event 49
Event handler for 'appConnect.configChangedTo' event 49
The configApplied() method 49
Pasteboard policy API details 50

The ACPasteboardPolicy enumeration 50
Requirements for successful secure copy to pasteboard 51
The pasteboardPolicy() method 51
Calling pasteboardPolicy() when your app launches 51
pasteboardPolicy() return value after updates to pasteboard policy 52
The 'appconnect.pasteboardPolicyChangedTo' event 52
Event handler for 'appConnect.pasteboardPolicyChangedTo' event 52
The pasteboardPolicyApplied() method 53
Open In policy API details 53
Overview of Open In handling 54
The ACOpenInPolicy enumeration 55
The openInPolicy() and openInWhitelist() methods 55
OpenInPolicy() method 55
OpenInWhitelist() method 55
Calling OpenInPolicy() and OpenInWhitelist() when your app launches 55
Method return values after updates to Open In policy 56
The 'appconnect.openInPolicyChangedTo' event 56
Event handler for 'appConnect.openInPolicyChangedTo' event 56
The openInPolicyApplied() method 57
Info.plist key related to the Open In policy 57
Print policy API details 58
The ACPrintPolicy enumeration 58
The printPolicy() method 58
Calling printPolicy() when your app launches 58
printPolicy() return value after updates to print policy 58
The 'appconnect.printPolicyChangedTo' event 59
Event handler for 'appConnect.printPolicyChangedTo' event 59
The AppConnectCordova.printPolicyApplied() method 59
Getting the AppConnect library version 60

Caching tunneled URL responses 60
iOS active state change events due to AppConnect control switches 61
Situations that trigger the state change notifications 61
Upload progress for AppTunnel data 62
The 'appconnect.uploadProgressDidChange' event 62
Developing third-party dual-mode apps 64
What is a dual-mode app? 64
Dual-mode app states 64
High-level dual-mode app behavior 65
When the app launches for the first time 65
When an app subsequently launches 66
User requests to switch to Non-AppConnect Mode 66
User requests to switch to AppConnect Mode 67
Data loss prevention policy handling 67
Dual-mode API details 67
The ACManagedPolicy enumeration 67
The managedPolicy() method 68
The 'appconnect.managedPolicyChangedTo' event 68
Event handler for 'appConnect.managedPolicyChangedTo' event 68
The stop method 69
The retire method 69
API call sequence when the app launches for the first time 69
API call sequence when the app subsequently launches 70
API call sequence when user requests Non-AppConnect Mode 70
API call sequence when user requests AppConnect Mode 71
Best practices using the AppConnect for iOS Cordova Plugin 72
Display authorization status in the home screen 72
Allow the user to enter credentials manually 72
Limit the size of configuration data from the MobileIron server 73

Consider limitations when using the iOS simulator 73
Enable the AppConnect library to blur screens when the app becomes inactive 74
Do not put secure data in the app bundle 74
Indicate to the user that the app is initializing 74
Disallow custom keyboard use 74
Provide documentation about your app to the MobileIron server administrator 75
Testing for third-party app developers 77
Third-party AppConnect app testing overview 77
Set up MobileIron Core 78
Login to the Admin Portal 78
Enable AppConnect on MobileIron Core 78
Configure the AppConnect global policy 78
Create an AppConnect container policy 79
Set up your end-user device 79
Set up Mobile@Work on an iOS device 79
Install your app on the device 80
Set up the AppConnect passcode on the device 80
Test authorization status handling 80
Change the status to authorized or unauthorized 80
Change the status to retired 81
Reauthorize a retired app 82
Test data loss prevention policy handling 82
Test AppConnect configuration change handling 85
Create an AppConnect app configuration 85
Update the AppConnect app configuration 86
Test using AppTunnel 87
Enable AppTunnel on MobileIron Core 87
Use an existing certificate 88
Generate a certificate 88

Create a certificate authority for using an AppTunnel with HTTP/S tunneling 88
Create a local certificate enrollment setting 89
Configure the Sentry with an AppTunnel service 90
Configure the AppTunnel service in the AppConnect app configuration 91
Test the app documentation 92
Testing for in-house app developers 93
In-house AppConnect app testing overview 93
Set up MobileIron Core 94
Login to the Admin Portal 94
Enable AppConnect on MobileIron Core 94
Create a label for testing your app 94
Upload your app to MobileIron Core if you use AppConnect.plist 95
Verify your AppConnect.plist settings 95
Configure the AppConnect global policy 96
Create an AppConnect container policy, if necessary 96
Set up your end-user device 97
Set up Mobile@Work on an iOS device 97
Install your app on the device 97
Set up the AppConnect passcode on the device 97
Test authorization status handling 97
Change the status to authorized or unauthorized 98
Change the status to retired 99
Reauthorize a retired app 99
Test data loss prevention policy handling 100
Test AppConnect configuration change handling 103
Create an AppConnect app configuration 103
Update the AppConnect app configuration 104
Test using AppTunnel 105
Enable AppTunnel on MobileIron Core 105

Use an existing certificate 106
Generate a certificate 106
Create a certificate authority for using an AppTunnel with HTTP/S tunneling 106
Create a local certificate enrollment setting 107
Configure the Sentry with an AppTunnel service 108
Configure the AppTunnel service in the AppConnect app configuration 109
Test the app documentation 110
AppConnect for iOS Cordova Plugin revision history 111
AppConnect 4.7.0 for iOS Cordova Plugin revision history 111
New features and enhancements summary 112
Resolved issues 112
Known issues 112
Limitations 112
AppConnect 4.6.0 for iOS Cordova Plugin revision history 113
New features summary 113
Resolved issues 113
AppConnect 4.5.3 for iOS Cordova Plugin revision history 114
Resolved issues 114
AppConnect 4.5.2 for iOS Cordova Plugin revision history 114
AppConnect 4.5.1 for iOS Cordova Plugin revision history 114
AppConnect 4.5.0 for iOS Cordova Plugin revision history 114
Resolved issues 114
Known issues 115
AppConnect 4.4.2 for iOS Cordova Plugin revision history 115
Resolved issues 115
Known issues 115
AppConnect 4.4.1 for iOS Cordova Plugin revision history 115
Resolved issues 115
Known issues 116

AppConnect 4.4.0 for iOS Cordova Plugin revision history 116
New features summary 116
Resolved issues 116
Limitations 117
AppConnect 4.3.1 for iOS Cordova Plugin revision history 117
New features 117
Resolved issues 117
AppConnect 4.3 for iOS Cordova Plugin revision history 117
New features 117
AppConnect 4.2.1 for iOS Cordova Plugin revision history 118
New features 118
AppConnect 4.2 for iOS Cordova Plugin revision history 118
Resolved issues 118
Known issues 118
AppConnect 4.1.1 for iOS Cordova Plugin revision history 118
Resolved issues 119
Known issues 119
AppConnect 4.1 for iOS Cordova Plugin revision history 119
New features 119
Certificate pinning support 119
Lock AppConnect apps when screen is off 119
Overriding the Open In Policy for OpenURL with the mailto: scheme 120
AppConnect 4.0 for iOS Cordova Plugin revision history 120
New features 120
iOS 8 no longer supported 120
Drag and Drop data loss prevention policy support 120
Native email control using the Open In DLP policy 121
App extension control using the Open In DLP policy 121
Custom keyboard use controlled by MobileIron server 121

Screen blurring 121
Requirement for Face ID usage Info.plist entry 122
Support for sending AppConnect logs from Mobile@Work 122
Automatic policy status updates sent to MobileIron server 122
Support for storing AppConnect library encryption keys in the Secure Enclave 123
Resolved issues 123
Known issues 123
Limitations 123
AppConnect 3.5 for iOS Cordova Plugin revision history 123
New features 123
iOS 11 compatibility 123
Open In changes 124
Resolved issues 124
Limitations 124
AppConnect 3.1.3 for iOS Cordova Plugin revision history 124
Resolved issues 124
AppConnect 3.1.2 for iOS Cordova Plugin revision history 125
Resolved issues 125
AppConnect 3.1.1 for iOS Cordova Plugin revision history 125
Resolved issues 125
AppConnect 3.1 for iOS Cordova Plugin revision history 125
New features 125
Update to OpenSSL 1.0.2h 125
Open In policy now enforced by AppConnect library 126
Resolved issues 126
Known issues 126
Limitations 126
AppConnect 3.0 for iOS Cordova Plugin revision history 126
Releases prior to AppConnect 3.0 for iOS Cordova Plugin revision history 126

New features and enhancements

This guide documents the following new features and enhancements:

- **Avoid pasteboard notifications**: To avoid pasteboard notifications on users' devices when using AppConnect apps, set up an App Group for your AppConnect apps. Setting up an App Group reduces the amount of flipping between the AppConnect app and the MobileIron client and avoids pasteboard notifications. For more information, see Optional: Avoiding pasteboard notifications.

For new known and resolved issues and limitations, see AppConnect for iOS Cordova Plugin revision history.

Introducing the MobileIron AppConnect for iOS Cordova Plugin

- AppConnect for iOS overview
- Product versions required
- Securing and managing the app using the AppConnect library

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 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.
- 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 Xamarin C# binding

  An app developer using the Xamarin development platform can use C# APIs to create a secure app, or turn an existing app into a secure app.

  See the *AppConnect for iOS SDK Developers Guide*

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 The Following:

- If your AppConnect app is to be distributed from the Apple App Store, due to Apple App Store requirements, your app is required to work as a regular app in addition to working as an AppConnect app.
  See Developing third-party dual-mode apps.
- 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.
- An Apple Developer Enterprise Program account is required to distribute in-house apps. See Apple Developer Enterprise Program.

Where to get the AppConnect for iOS Cordova Plugin

Get the latest AppConnect for iOS Cordova Plugin at https://help.mobileiron.com/s/software.

AppConnect for iOS documentation is available at MobileIron AppConnect for iOS Product Documentation.

Legal notices are also available 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 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.
- Handle data loss prevention.
  The MobileIron server administrator decides whether an app can copy content to the iOS pasteboard, use the document interaction feature (Open In and Open From), use drag and drop, or print. The app uses this information to limit its functionality to prevent data loss through these features. The AppConnect library enforces the pasteboard, Open In, Open From, and drag and drop policies. The app enforces the print policy.

- Control custom keyboard use by your app.
  The MobileIron server administrator can choose whether an app can use custom keyboards, and the AppConnect library enforces the choice.. If the administrator does not configure this choice, your app can choose to reject custom keyboard use.
- Blur the app's screens when the app is not in the foreground.
  By default, the AppConnect library enforces this behavior, which can be overridden by the MobileIron server administrator.

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 the MobileIron server.
- 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 the MobileIron server 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.

64-bit and 32-bit app support

Using the AppConnect for iOS Cordova Plugin, you can build an app as a 64-bit app or as a 32-bit app.

MobileIron AppConnect components

The following table describes the MobileIron components that work with he apps that you build with the AppConnect for iOS Cordova Plugin.

TABLE 1. MOBILEIRON COMPONENTS THAT WORK WITH IOS CORDOVA PLUGIN

MobileIron component	Description
MobileIron Core	The MobileIron on-premise server which provides security and management for an enterprise's devices, and for the apps and data on those devices. An administrator configures the security and management features using a web portal.
MobileIron Connected Cloud	MobileIron's cloud offering that has the same functionality as MobileIron Core.
MobileIron Cloud	MobileIron's cloud offering that provides similar functionality as MobileIron Core. However, it does not support all the AppConnect features that MobileIron Core supports.
Standalone Sentry	The MobileIron server which provides secure network traffic tunneling from your app to enterprise servers.
The Mobile@Work for iOS app	A MobileIron app that runs on an iOS device. It interacts with MobileIron Core or Connected Cloud to get current security and management information for the device. It interacts with the AppConnect library to communicate necessary information to your app.
The MobileIron Go app	A MobileIron app that runs on an iOS device. It interacts with MobileIron Cloud to get current security and management information for the device. It interacts with the AppConnect library to communicate necessary information to your app.
The MobileIron AppStation app	A MobileIron client app that runs on an iOS device. It interacts with MobileIron Cloud. It can be used on the device instead of MobileIron Go when the MobileIron Cloud tenant supports Mobile Apps Management (MAM) but not Mobile Device Management (MDM). It interacts with the AppConnect library to communicate necessary information to your app.
The AppConnect library	The MobileIron library that your app uses to get AppConnect information. The AppConnect library is part of the AppConnect framework that your app includes.

Note The Following:

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

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

Using a secure app

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

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

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
- handling the data loss prevention settings
- using the app-specific configuration

The MobileIron client app and AppConnect library responsibilities

The MobileIron client app app and the AppConnect library are responsible for:

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

Cordova Plugin variants

Due to Apple deprecating the UIWebView class, the AppConnect for iOS SDK is available in two variants: one with UIWebView and WKWebView support, and another another with WKWebView support, but no UIWebView support. The AppConnect SDK without UIWebView support is provided for apps that will be submitted to the App Store. The Cordova plugin included with each variant of the SDK provides the same support as the SDK variant.

AppConnect Cordova Plugin contents

The AppConnect Cordova Plugin is available in the AppConnect for iOS SDK ZIP file. The ZIP file is called AppConnectiOSSDK_V<*version*>_<*build*>.zip, where:

- <*version*> is the version number of the SDK.
- <*build*> is the build number of the SDK.

The AppConnect for iOS ZIP file contains a plugins/cordova folder which contains:

- AppConnectCordovaPlugin_V<*version number_xxx*>.zip
  This ZIP file contains all the plugin files, which include:

TABLE 2. CORDOVA PLUGIN CONTENTS

Contents	Description
src/ios folder	Contains the AppConnect library files. The AppConnect library provides your app management and security capabilities. The AppConnect library facilitates communication between your app and the MobileIron client app, which communicates with the MobileIron server.
www folder	Contains AppConnectCordova.js. This file contains the JavaScript interfaces that your app uses to interact with the AppConnect library.
plugin.xml	Defines the AppConnect Cordova Plugin.
Notices.pdf	Contains license information

- A Documentation folder containing this document.
  Check for updates to this document as described in Where to get the AppConnect for iOS Cordova Plugin.
- install_ac_cordova_plugin.sh, which is the script you use to install the AppConnect Cordova Plugin.
- A Samples folder containing the TestAppConnect example
  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.

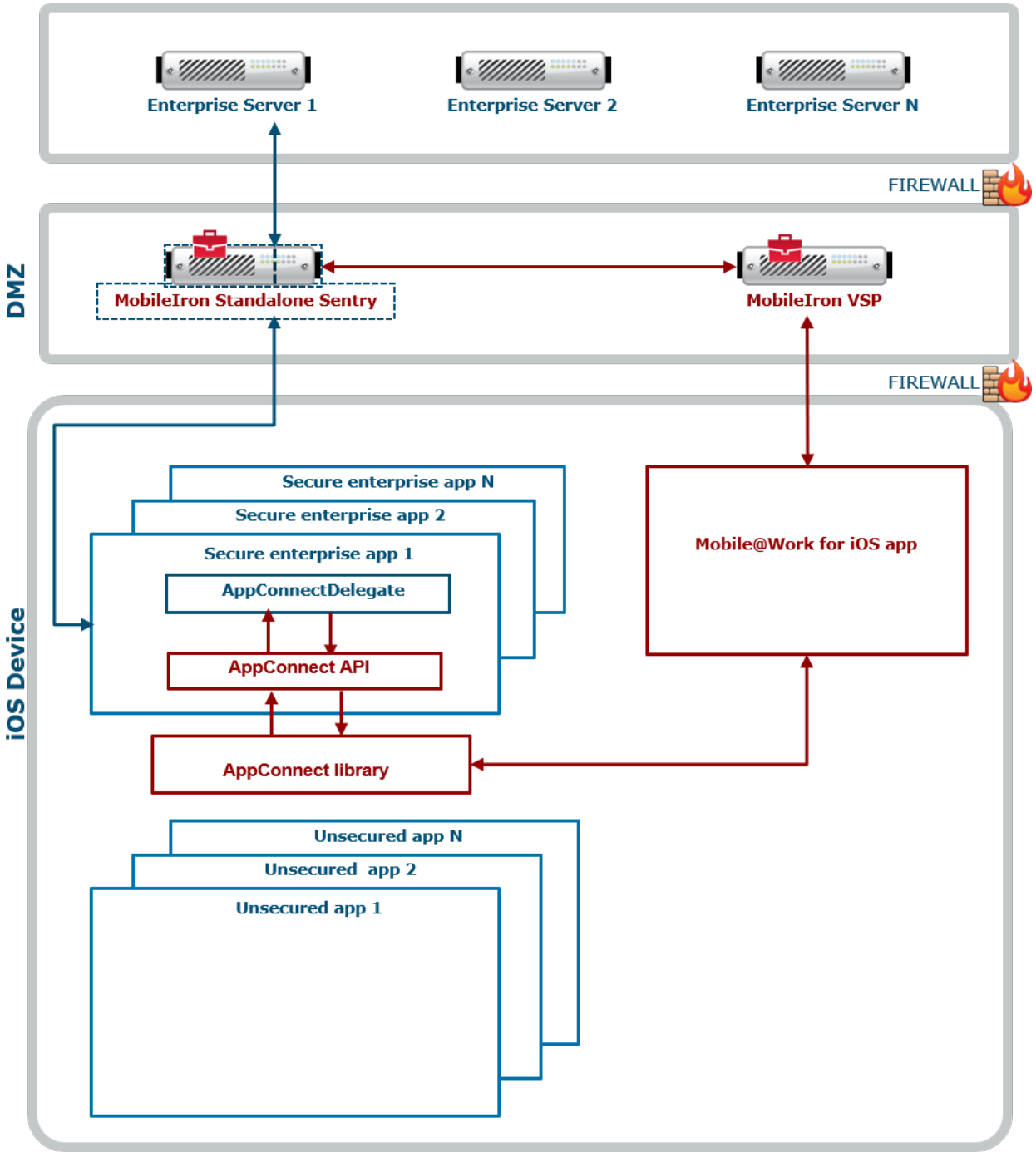
Another plugins/cordova folder is available in the SDK_without_UIWebView folder, which contains the SDK variant that does not support UIWebView.

AppConnect for iOS architecture

You app, using the AppConnect Cordova Plugin, interacts with the MobileIron client app. The MobileIron client app is Mobile@Work for iOS, MobileIron Go for iOS, or MobileIron AppStation for iOS. Mobile@Work interacts with Core and MobileIron Go interacts with MobileIron Cloud. AppStation is used in certain use cases instead of MobileIron Go to interact with MobileIron Cloud when a MobileIron Cloud tenant is set up for Mobile Apps Management (MAM) but not Mobile Device Management (MDM). The AppConnect library also interacts with Standalone Sentry for AppTunnel support.

The following diagram illustrates these interactions between an AppConnect app, the AppConnect library, the MobileIron server, the MobileIron client and the Standalone Sentry. The diagram uses MobileIron Core for the server and Mobile@Work for the client.

FIGURE 1. APPCONNECT FOR IOS ARCHITECTURE



Note The Following:

- Each secure enterprise app communicates with an instance of the AppConnect library.
- The AppConnect library communicates with the MobileIron client 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 MobileIron client app communicates with the MobileIron server to get management and security-related information.
- 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.

The MobileIron client app and AppConnect apps

The MobileIron client app supports AppConnect apps, including the following tasks:

- It communicates with the MobileIron server to get management and security-related information and passes the information to the AppConnect apps.
  The MobileIron client app periodically does an *app checkin* with the MobileIron server to get this information. The administrator configures the app checkin interval on the MobileIron server. It is the maximum time between app checkins while an AppConnect app is running.
- It enforces the AppConnect passcode or Touch ID/Face ID.
  The MobileIron client app prompts the device user to create an AppConnect passcode or Touch ID/Face ID when first launching any AppConnect app. You configure an auto-lock timeout in the AppConnect global policy. After this period of inactivity, The MobileIron client app prompts the device user to reenter his AppConnect passcode or Touch ID/Face ID.

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

App checkin and the MobileIron client app

On each app checkin, the MobileIron client app gets AppConnect policy updates for all the AppConnect apps that have already run on the device. These updates include changes to data loss prevention policies, password settings, app configurations, and AppTunnel settings.

For example, for Mobile@Work, these updates are due to changes on MobileIron Core 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.

The MobileIron client app does an app checkin in the following situations:

- The device user launches an AppConnect app for the first time.
  In this situation, the MobileIron client app 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, the MobileIron client app launches, and the device user sees the the MobileIron client app app momentarily. Once the MobileIron client app has completed the app checkin, the device user automatically returns to the AppConnect app.

The AppConnect passcode auto-lock time and the MobileIron client app

The MobileIron client app launches to prompt the device user for the AppConnect passcode or Touch ID/Face ID in the following situations:

- The auto-lock (inactivity) timeout expires while the device is running an AppConnect app and the AppConnect passcode, or Touch ID/Face 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 the MobileIron client app to log out of AppConnect apps, and then launches an AppConnect app.
- The MobileIron server administrator has changed the complexity rules of the AppConnect passcode, and an app checkin occurs.

In each of these situations, the MobileIron client app launches, and presents the device user with a screen for entering his AppConnect passcode or Touch ID/Face ID. After the device user enters the passcode or Touch ID/Face 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. This information is current at the time of this release. For MobileIron product versions released after this release, see that product version's release notes for the most current support and compatibility information.

TABLE 3. SUPPORT AND COMPATIBILITY

Product	Supported versions	Compatible versions
iOS	11.0 - 14.0	9.0 and lower are not supported
Cordova release and Cordova iOS native platform	7.0.0 with Cordova iOS native platform 4.4.0	Releases after 7.0.0 with Cordova iOS native platform 4.4.0
MobileIron Core and Connected Cloud	10.5.0.0, 10.6.0.0 , 10.7.0.0	10.3.0.0 - 10.4.0.0
Standalone Sentry	9.7.3, 9.8.1	9.5.0 - 9.6.0
Mobile@Work for iOS	12.3.0, 12.4.1	12.1.0 - 12.2.2
MobileIron Cloud	72	Not applicable
MobileIron Go	5.5.1	4.0.0 - 5.5.0
MobileIron AppStation	1.3.0	Not applicable

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

Securing and managing the app using the AppConnect library

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

- Authorization
- AppConnect passcode and Touch ID/Face ID policy
- Configuration specific to the app
- AppTunnel
- Certificate authentication to enterprise services
- Custom keyboard control
- Data loss prevention policies

Additionally, the AppConnect library uses encryption to protect AppConnect-related data, as descrobed in Data protection. However, this protection requires no additional MobileIron server configuration besides requiring an AppConnect passcode or device passcode.

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

1. The MobileIron server administrator decides which app-related settings to apply to a device or set of devices.
2. The MobileIron server sends the information to the MobileIron client app.
3. The MobileIron client app passes the information to the AppConnect library, which is part of the AppConnect Cordova Plugin. The MobileIron client 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*
- *handling the data loss prevention policies*
- *using the configuration specific to the app.*

Authorization

The MobileIron server 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. The MobileIron client app reports device information to the MobileIron server. The server then determines whether to change the user to unauthorized based on security policies on the server.
  When a user becomes unauthorized, the 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.

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.

AppConnect passcode and Touch ID/Face ID policy

The MobileIron server administrator determines:

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

The AppConnect library and the MobileIron client app enforce an AppConnect passcode as follows:

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

Your app does not handle the AppConnect passcode or Touch ID/Face ID at all. The AppConnect library and the MobileIron client app enforce the passcode or Touch ID/Face 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 server administrator specifies the key-value pairs for each app on the MobileIron server. The administrator applies the appropriate set of key-value pairs to a set of devices. Sometimes more than one set of key-value pairs exists on the server for an app if different users require different configurations. For example, the administrator can assign a different server address to users in Europe than to users in the United States.

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

Your app uses the AppConnect for iOS Cordova Plugin to get the configuration and to be notified of changes. Then your app applies the configuration according to its requirements. For more information, see App-specific configuration API details .

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 AppTunne with HTTP/S tunneling. The MobileIron server administrator handles all AppTunnel configuration on the server. Once the administrator has configured tunneling for the app on the server, the AppConnect library, the MobileIron client app, 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 MobileIron server administrator uses AppConnect's app-specific configuration to specify the enterprise server URL that the app uses. See Configuration specific to the app.

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

Supported APIs

AppTunnel with HTTP/S tunneling supports typical JavaScript network APIs, such as XMLHttpRequest or jQuery calls. More generally, AppTunnel with HTTP/S tunneling supports any network API that Cordova binds to the Objective-C APIs NSURLConnection or NSURLSession (although NSURLSession in a background session is not supported). If you use a Cordova plugin that uses other Objective-C APIs, including WKWebView, or accesses sockets directly, AppTunnel with HTTP/S tunneling is not supported.

AppTunnel with TCP tunneling

AppTunnel can tunnel TCP traffic between an app and a server behind the company's firewall. AppTunnel with TCP tunneling does not require an app to be an AppConnect app; both AppConnect apps and standard apps can use AppTunnel with TCP tunneling.The MobileIron server administrator configures Advanced AppTunnel, including installing MobileIron Tunnel (an iOS app) on the device. *Your app takes no actions related to using AppTunnel with TCP tunneling.*

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 server administrator configures on the server which certificate for the app to use, and which connections use it. The AppConnect library, which is part of every AppConnect app, makes sure the connection uses the certificate.

Your app takes no action at all.

Supported networking methods

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

- NSURLConnection
- NSURLSession

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

- Networking libraries that use NSURLConnection or NSURLSession.
- UIWebView

Unsupported networking methods

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

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

Data loss prevention policies

An app can leak data if it uses iOS features such as copying to the iOS pasteboard, document interaction (Open In), drag and drop, and print capabilities. A MobileIron server administrator specifies on the server 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.
  Your app enforces the print policy by enabling or disabling printing capabilities based on the print policy.
- The pasteboard policy specifies whether your app is allowed to copy content *to* the iOS pasteboard. If copying content is allowed, the policy specifies whether all apps, or only AppConnect apps, can paste the copied content *from* the pasteboard.
  The AppConnect library enforces the pasteboard policy. Your app disables or enables any special user interfaces that allow copying.
- The Open In policy specifies the apps, including the extensions that apps provide, with which your app can share documents. The policy specifies no apps, all apps, all AppConnect apps, or a set of apps. A set of apps is called the whitelist. Whether your app can share documents with the native iOS mail app is also controlled by the Open In policy.
  The AppConnect library enforces the Open In policy. Your app disables or enables any special user interfaces that give the user the option to use Open In.
- The Open From policy specifies the apps, including the extensions that apps provide, from which your app can receive documents when the other app uses the Open In iOS feature. The policy specifies no apps, all apps, all AppConnect apps, or a set of apps. A set of apps is called the whitelist. The AppConnect library enforces this policy. Your app provides no code to support this policy.
- The drag and drop policy specifies whether AppConnect apps can drag content to all other apps, to only other AppConnect apps, or not at all. The AppConnect library enforces this policy. Your app provides no code to support this policy.

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

Your app uses the AppConnect for iOS Cordova Plugin to get the data loss prevention policies and to be notified of changes. Then your app handles the policies according to its requirements.

For more information, see:

- Pasteboard policy API details
- Open In policy API details

- Print policy API details

Custom keyboard control

Custom keyboard extensions sometimes send data to servers when a device user enters data into an app. They send this data for assistance with word-prediction, for example. To stop this potentially harmful data loss, the MobileIron server administrator configures whether custom keyboards are allowed for an app by setting a key-value pair in the app's configuration. The key is called MI_AC_IOS_ALLOW_CUSTOM_KEYBOARDS. The key-value pair is consumed by the AppConnect library; your app does not receive it.

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

If the server administrator does not include the key-value pair for your app, the AppConnect library does not allow the app to use custom keyboards.

Related topics

Disallow custom keyboard use

Data protection

The MobileIron client app 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/Face ID to access AppConnect apps.

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

Your app does not handle data protection for AppConnect-related data. The MobileIron client app and the AppConnect library provide this data protection.

Optional: Avoiding pasteboard notifications

To avoid pasteboard notifications on users' devices when using AppConnect apps, set up an App Group for your AppConnect apps. App Groups are an iOS mechanism to share data between apps. Setting up an App Group also reduces the amount of switching between the AppConnect app and the MobileIron client. The following is an overview of the setup needed to avoid pasteboard notifications.

Overview

1. In the Apple Developer portal,
   a. Create an App Group.
   b. Add the App Group to your AppConnect app's App ID.
   c. Update and download the Provisioning Profile.
2. Update the app to use the new Provisioning Profile.
3. Configure App Group capability and App Group for the app in Xcode.

   NOTE: Ensure that the app group name configured in Xcode matches the App Group name that you configured in the Apple Developer Portal.
4. Add the App Group to the app's Info.plist.

For detailed instructions for each step, see the links in the following related topics.

Related topics

- For information about how to create an App Group, add it to your AppConnect app's App ID, and save and download the updated Provisioning Profile for your app on the Apple Developer portal, see Configuring an App Group on the Apple Developer portal
- For information about configuring App Group capability and App Groups for you app, see the following Apple documentation:
  - Adding Capabilities to Your App
  - Configure app groups
- For information about adding the App Group to the app's Info.plist, see Add the App Group to Info.plist.

The feature is available if the components are at the following version through the latest as supported by MobileIron:

- The AppConnect app uses AppConnect 4.7.0 SDK.
- The iOS device uses iOS 14.
- The MobileIron client is one of the following
  - MobileIron Go 5.5.1
  - Mobile@Work 12.4.1

NOTE: AppConnect apps continue to use the pasteboard if an App Group, as described in this section, is not set up.

Configuring an App Group on the Apple Developer portal

You create an App Group, add it to your AppConnect app's App ID, and save and download the updated Provisioning Profile for your app on the Apple Developer portal.

Procedure

1. On the Apple Developer portal, go to **Certificates, Identifiers & Profiles > Identifiers**.
2. Select **App Groups**, and create an App Group.
   When you create an App Group, you add a name and an Identifier for the App Group. The name can be anything, as long as it is unique.
3. After you create the App Group, go to **Certificates, Identifiers & Profiles > Identifiers**.
4. Select **App IDs**, and click the AppConnect app.
5. Select **App Groups > Configure**, and select the App Group to assign to the AppConnect app.
6. After you update the App Group for the AppConnect app, to **Certificates, Identifiers & Profiles > Identifiers**.
7. Select **Profiles**, and click the provisioning profile for your app to edit.
8. Click **Edit > Save > Download**.
9. Double-click the Provisioning Profile in the Finder to import it into Xcode.

Add the App Group to Info.plist

The App Group is created by the app developer in the Apple Developer Portal using their Apple developer account. On the Apple developer portal, create an App Group and add it to your AppConnect apps.

Add the following key-value pair in the app's Info.plist:

- MI_APP_CONNECT
  This key is the root key, and its value is a dictionary of key-value pairs
- MI_AC_ACCESS_GROUP
  This key provides the App Group Identifier that the AppConnect library uses to access the app's shared container. The value is the app's App Group Identifier.

  Example

  In the example, group.com.thirdparty.enterprise.ios.appconnect is the App Group Identifier.

Getting started with the AppConnect for iOS Cordova Plugin

- Upgrade tasks
- Getting started tasks
- Troubleshooting

Upgrade tasks

To upgrade an app to the current version of AppConnect for iOS Cordova Plugin, do the following:

1. Replace the AppConnect.framework bundle in the project folder.
2. If you are using the AppConnectExtension.framework, replace the AppConnectExtension.framework bundle in the project folder.
3. Declare the `alt-appconnect`URL scheme in your app's Info.plist as another allowed URL scheme. See Upgrade tasks.

If your SDK at a version prior to 4.0, see the task list in the following table.

TABLE 4. UPGRADE TASK LIST

Cordova plugin version from which you are upgrading	Upgrade task list
Prior to Version 4.0	• Update the AppConnect for iOS Cordova Plugin in your app with the new plugin • Declare the `alt-appconnect`URL scheme in your app's Info.plist as another allowed URL scheme. See Declare the AppConnect URL schemes as allowed. • Add AppConnect-related entries to your Info.plist. • Update Xcode project settings • If you blocked custom keyboard usage in your app, remove that code. The AppConnect library handles whether custom keyboards are allowed. For details, see Custom keyboard use controlled by MobileIron server.

Getting started tasks

Use these tasks to begin development of Cordova AppConnect apps.

Once you have completed these tasks, your app is ready to use the Cordova plugin to, for example, enforce MobileIron server settings and apply app-specific configurations from the MobileIron server.

Before you begin

- Get the latest version of the AppConnect for iOS Cordova Plugin from https://help.mobileiron.com/s/software.
- Be sure you have the required product versions for working with apps built with the AppConnect for iOS Cordova plugin.
  See Product versions required .

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
2. Declare the AppConnect URL schemes as allowed
3. Add AppConnect-related entries to your Info.plist
4. Update Xcode project settings
5. Initialize the AppConnect library
6. Wait for the AppConnect library to be ready

Optionally, you can create an AppConnect.plist file. See Specify app permissions and configuration in a plist file.

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.

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.

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.

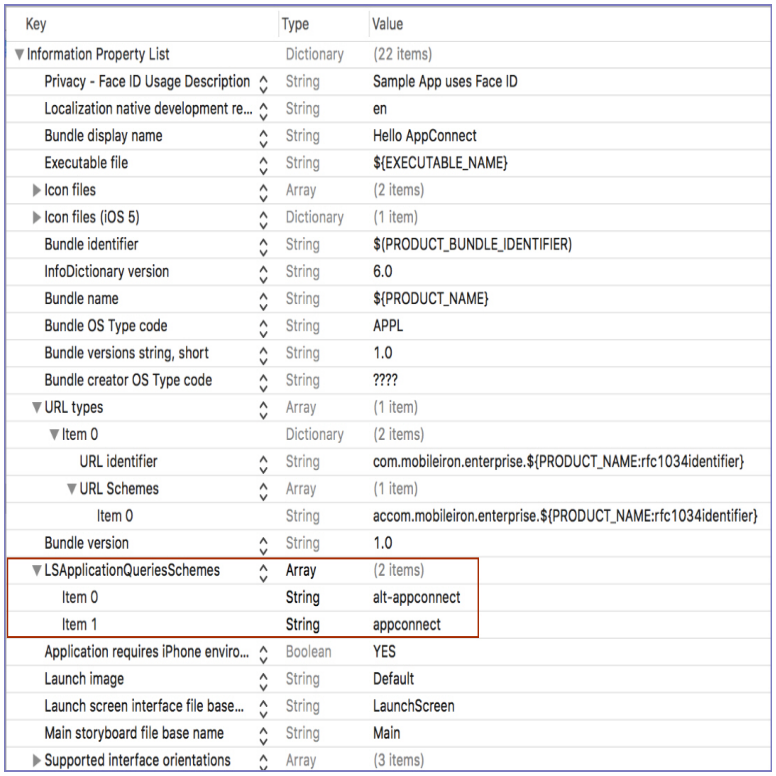
Declare the AppConnect URL schemes as allowed

Declare the `appconnect` and the `alt-appconnect`URL schemes in your app's Info.plist as allowed URL schemes. Your app's instance of the AppConnect library:

- uses the `appconnect` URL scheme to communicate with Mobile@Work or MobileIron Go.
- uses the `alt-appconnect` URL scheme to communicate with MobileIron AppStation.

To allow the `appconnect` and `alt-appconnect` URL schemes, add a key called `LSApplicationQueriesSchemes` as shown in this example from HelloAppConnect's HelloAppConnect-Info.plist:

FIGURE 2. APPCONNECT URL SCHEME VIEWED IN XCODE



Do the following steps (based on Xcode 7.1):

1. Open the app's .xcodeproj file in Xcode.
2. Select your app's Info.plist file.
3. Select Editor > Add Item in the menu.
4. For the Key, enter `LSApplicationQueriesSchemes.`
5. For the Type, select Array.
6. Select the new key.
7. Select Editor -> Add Item.
8. Set the value of the Item to `appconnect`.
9. Select the new key again.
10. elect Editor -> Add Item.
11. Set the value of the Item to `alt-appconnect`.

Add AppConnect-related entries to your Info.plist

- Enable screen blurring
- Allow Face ID

Enable screen blurring

The AppConnect library can automatically blur your app's screen whenever it is not active, and unblur it when the screen becomes active again. This security measure protects the app's data from being captured in screenshots.

To enable screen blurring, add the key MI_AC_PROVIDE_SCREEN_BLUR to your app's Info.plist as a Boolean. Set the value to YES.

When you set the Info.plist key MI_AC_PROVIDE_SCREEN_BLUR to YES, the MobileIron server administrators can disable screen blurring by setting a key-value pair on the server for your app's configuration. The server key is MI_AC_ENABLE_SCREEN_BLURRING with the value false.

NOTE: If you already implemented screen blurring in your app, remove that code and use the MI_AC_PROVIDE_SCREEN_BLUR plist key. Using the plist key ensures that all AppConnect apps behave consistently.

Allow Face ID

Include **Privacy - Face ID Usage Description** to your app's info.plist, with a string value indicating the purpose of Face ID use. For example, add the value **AppConnect**. If you manually add this key, its name is NSFaceIDUsageDescription.

Server administrators can allow the use of Touch ID or Face ID instead of an AppConnect passcode. Therefore, this Info.plist entry is required on iOS 11 through the most recently released version as supported by MobileIron.

Update Xcode project settings

Running the AppConnect Cordova Plugin installation script created an Xcode project for your app.

In the Xcode project, do the following:

1. Add a **Run Script** section to the **Build Phases** settings of your Xcode project's target.
2. Add post_embed_actions.sh, located in the top-level of the extracted AppConnect SDK directory, to the scripts to run.
   This script removes extra architectures from the AppConnect app's binary.

   IMPORTANT: Removing desktop architectures is required before submitting your app to the Apple App Store.
3. Make sure **Enable Bitcode** is set to **No** in **Build Options** in the **Build Settings** of the Xcode project's target.

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.

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.

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 prevention 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 the MobileIron server to make it easier for the server administrator to set up your app with the correct data loss prevention 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 the MobileIron server, the server uses this plist file to automatically create server policies that contain your specified data loss prevention 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 prevention 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 prevention policies and app-specific configuration by using the AppConnect for iOS Cordova Plugin APIs, described in AppConnect for iOS Cordova Plugin API.
   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 prevention 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:

FIGURE 3. APPCONNECT.PLIST VIEWED IN XCODE

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`, `openfrom`, `openfromwhitelist`,`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 .
`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`.
`openfrom`	• `allow` The app is allowed to receive documents from all other apps when they use Open In. • `disable` The app is not allowed to receive documents from any other apps when they use Open In. • `whitelist` The app is allowed to receive documents only from apps listed in the `openfromwhitelist` . • `appconnect` The app is allowed to receive documents only from other AppConnect apps.

Key	Possible values and meanings
`openfromwhitelist`	Semi-colon separated list of the bundle IDs of the apps from which the app is allowed to receive documents. This key is necessary when the `openfrom` 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. Then, any app can copy from the content from the pasteboard. • `disable` Pasteboard interaction is not allowed. • `appconnect` Pasteboard interaction is allowed only with other AppConnect apps. That is, this option allows the device user to be able to copy content from your app to the iOS pasteboard. Then, only other AppConnect apps can copy the content from the pasteboard.
`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.

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 .

AppConnect for iOS Cordova Plugin API

- AppConnect for iOS Cordova Plugin overview
- AppConnect ready API details
- Authorization API details
- App-specific configuration API details
- Pasteboard policy API details
- Open In policy API details
- Print policy API details
- Getting the AppConnect library version
- Caching tunneled URL responses
- iOS active state change events due to AppConnect control switches

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 with HTTP/S tunneling.
- Getting notifications of iOS active state changes due to certain control switches to and from the MobileIron client app.

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 the MobileIron server.

For more information, see Developing third-party dual-mode apps.

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.

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
- AppConnect ready API details

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

- AppConnect ready API details
- Authorization API details
- App-specific configuration API details
- Pasteboard policy API details
- Open In policy API details
- Print policy API details
- Getting the AppConnect library version
- Caching tunneled URL responses

NOTE: The AppConnectCordova interface also provides methods specifically for dual-mode apps. These methods are described in Developing third-party dual-mode apps.

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
- upload progress for data tunneled with AppTunnel

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.
`'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.
`'appconnect.uploadProgressDidChange'`	Upload progress for AppTunnel data has changed. Handling this event is optional. Handle it if your app supports tunneling data with AppTunnel.

For details about handling each event, see:

- AppConnect ready API details
- Authorization API details
- App-specific configuration API details
- Pasteboard policy API details
- Open In policy API details
- Print policy API details

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 server 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`

IMPORTANT: 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 prevention 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 prevention 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 .

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 MobileIron server log files.

The displayMessage() method

The AppConnect Cordova Plugin provides a method that causes the the MobileIron client 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 the MobileIron server. For an overview of this feature, see Configuration specific to the app.

The config() method

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

```
AppConnectCordova.config()
```

Return value: A JavaScript object of key-value pairs. Each key and value is a string corresponding to the key and value configured on the MobileIron server. For example:

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

If no key-value pairs are configured on the MobileIron server, the return value is an empty object: []

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

Event handler for 'appConnect.configChangedTo' event

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

1. Stores the updated JavaScript object 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 server 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 the MobileIron server. 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 MobileIron server 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 the MobileIron server. For an overview of this feature, see Data loss prevention policies.

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 cannot write data to the pasteboard.
                     // The AppConnect library enforces this status and ensures
                     // that the app cannot modify the pasteboard contents.

    AUTHORIZED:   1, // The application may write data
                     // to the pasteboard which gets shared among all apps.
                     // (Both AppConnect and non-AppConnect apps can read this data).

    SECURECOPY:   2  // The application may write data to the general pasteboard which
                     // is shared with authorized AppConnect apps.
                     // The AppConnect library implements the underlying technology so
                     // that the data written to the general pasteboard by one
                     // AppConnect app is only readable by authorized AppConnect apps.
                     // If the app is not authorized, or if the device user has not
                     // entered the AppConnect passcode when required, writing to the
                     // general pasteboard will fail, and reading from
                     // it will read unsecured content, if any.
}
```

Handle the pasteboard policy status as follows:

- Both `AUTHORIZED` and `SECURECOPY` indicate that copying content to the pasteboard is allowed. The AppConnect library handles making sure all apps or only AppConnect apps can paste the data. When the value

is `SECURECOPY`, the AppConnect library encrypts the data copied to the pasteboard, and decrypts the data when it is pasted to another AppConnect app.

- The status `UNAUTHORIZED` indicates that writing data to the pasteboard is not allowed. The AppConnect library enforces the status `UNAUTHORIZED`. Therefore, with this status, *even if you use an API to write to the pasteboard, the data is not written*.

  - Although the AppConnect library does not allow writing data to the pasteboard, your app should disable special user interfaces, if any, that it uses for copying content to the pasteboard. By disabling such user interfaces, your app does not give the end user the impression that copying is possible when the AppConnect library has disabled it.
  - iOS behavior still causes the copy button to display, which can cause an end user who taps the copy button to expect that text has been copied.

Requirements for successful secure copy to pasteboard

The pasteboard policy `SECURECOPY` means that the AppConnect library encrypts and decrypts pasteboard data. However, this encryption requires that:

- The app is authorized.
- The device user has entered the AppConnect passcode (or Touch ID/Face ID), if the MobileIron server administrator required one.

If either of these requirements are not true when the pasteboard policy is `SECURECOPY`:

- Writing content to the pasteboard (copying) fails. No data is written.
- Reading content from the pasteboard (pasting) reads unsecured content, if any.

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

The AppConnect library enables or disables the app's ability to copy content to the pasteboard depending on the `pasteboardPolicy` value:

- Copying is enabled for `AUTHORIZED`.
- Copying is disabled for `UNAUTHORIZED`.
- Copying is enabled for `SECURECOPY`, unless the app is unauthorized or the user has not entered a required AppConnect passcode.

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, depending on the `AppConnectCordova.ACPasteboardPolicy` value returned from `AppConnectCordova.pasteboardPolicy()`, disable or enable special user interfaces, if any, that the app uses for copying content to the pasteboard. Although the AppConnect library enables or disables writing data to the pasteboard, your app should not give the end user the impression that copying is possible when the AppConnect library has disabled it.

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 the MobileIron server, 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. Disables or enables copying to the pasteboard as follows:
   - Enables copying for `AUTHORIZED`.
   - Disables copying for `UNAUTHORIZED`.
   - Enables copying `SECURECOPY`, unless the app is unauthorized or the user has not entered a required AppConnect passcode..
3. 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 special user interfaces, if any, that the app uses for copying content to the pasteboard. Although the AppConnect library disables writing data to the pasteboard, your app should not give the end user the impression that copying is possible when the AppConnect library has disabled it. • Calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
`AUTHORIZED` or `SECURECOPY`	• Enables special user interfaces, if any, that the app uses for copying 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 MobileIron serverlog 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 the MobileIron server. For an overview of this feature, see Data loss prevention policies.

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.

Internally, a Cordova app can use either the UIDocumentInteractionController or UIActivityViewController classes from Objective-C. The class which the app uses impacts Open In handling as described in Overview of Open In handling.

- Overview of Open In handling

- The ACOpenInPolicy enumeration
- The openInPolicy() and openInWhitelist() methods
- The 'appconnect.openInPolicyChangedTo' event
- Event handler for 'appConnect.openInPolicyChangedTo' event
- The openInPolicyApplied() method
- Info.plist key related to the Open In policy

Overview of Open In handling

The behavior of the AppConnect library, and the actions your app takes, depend on the Open In policy status.

The possible status values are:

- `AUTHORIZED` -- The application is allowed to use Open In.
- `UNAUTHORIZED` -- The application is not allowed to use Open In.
- `WHITELIST` -- The application is allowed to use Open In to send documents only to apps in the whitelist.

IMPORTANT:

- Regardless of the Open In policy status, when an app makes an Open In request, iOS always displays all the apps that support the document type.
- Do not use the Objective-C class UIActivityViewController to perform Open In functionality. For example, make sure any Cordova plugin you use to provide the iOS Open In capability does not use UIActivityViewController. Because of iOS implementation, the AppConnect library cannot determine which app the end user selects, and therefore, whether the app is in the whitelist. To ensure security, the AppConnect library does not allow Open In to any app when the Open In policy is `ACOPENINPOLICY_WHITELIST` and the class used is UIActivityViewController.

The following table summarizes the behavior of the AppConnect library and the actions your app takes. It assumes you use UIDocumentInteractionController, and do not use UIActivityViewController.

Open In status	AppConnect library actions	Your app's actions
`AUTHORIZED`	The AppConnect library performs no actions on Open In behavior.	Enable user interfaces, if any, that give the user the option to use Open In. For example, if your app presents a menu item for Open In, the menu item should be enabled.
`UNAUTHORIZED`	If a user taps on any of the apps, the AppConnect library substitutes a dummy file with a mangled name. Therefore, the target app cannot open the file. Target app error handling varies. For example, some apps display an error pop-up.	Disable user interfaces, if any, that give the user the option to use Open In. For example, if your app presents a menu item for Open In, the menu item should be disabled. By disabling such user interfaces, your app does not give the end user the impression that Open In is possible when the AppConnect library has disabled it.
`WHITELIST`	If a user taps on an app that is not in the whitelist, the AppConnect library substitutes a dummy file with a mangled name. Therefore, the target app cannot open	Enable user interfaces, if any, that give the user the option to use Open In. For example, if your app presents a menu item for Open

Open In status	AppConnect library actions	Your app's actions
	the file. Target app error handling varies. For example, some apps display an error pop-up.	In, the menu item should be enabled.

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. Because the AppConnect library enforces Open In to only the whitelisted apps, your app uses this list only if it wants to inform the user about the list.

```
AppConnectCordova.openInWhitelist()
```

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

NOTE: When the Open In policy on the MobileIron server specifies "All AppConnect apps", the Open In status value is `ACOPENINPOLICY_WHITELIST`. The `openInWhitelist` lists all the currently authorized AppConnect apps. Therefore, your app handles the "All AppConnect apps" server setting the same way it handles the "whitelist" server setting.

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 user interfaces, if any, that give the user the option to use the Open In feature. The choice depends on the `AppConnectCordova.ACPOpenInPolicy` value returned from `AppConnectCordova.OpenInPolicy()`.

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"]` Because the AppConnect library enforces Open In to only the whitelisted apps, your app uses this list only if it wants to inform the user about the list. NOTE: When the Open In policy on the MobileIron server 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" server setting the same way it handles the "whitelist" server setting.

Event handler for 'appConnect.openInPolicyChangedTo' event

When a change has occurred to the Open In policy or whitelist on the MobileIron server, 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 user interfaces, if any, that give the user the option to use the Open In feature. • Calls the `AppConnectCordova.openInPolicyApplied()` method.
`AUTHORIZED`	• Enables user interfaces, if any, that give the user the option to use the Open In feature. • Calls the `AppConnectCordova.openInPolicyApplied()` method.
`WHITELIST`	• Enables user interfaces, if any, that give the user the option to use the Open In feature. • 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 MobileIron server log files.

Info.plist key related to the Open In policy

Your app can override the Open In policy when the policy blocks the iOS native email app when the app uses OpenURL with the `mailto:` scheme. Overriding the Open In policy for this scenario means that the iOS native email app is opened even though the Open In policy is one of the following:

- `ACOPENINPOLICY_UNAUTHORIZED`
- `ACOPENINPOLICY_WHITELIST`, and the whitelist does not contain the bundle IDs of the native iOS email app.

To override the Open In policy for this scenario, add the key MI_AC_DISABLE_SCHEME_BLOCKING with the value `YES` in the MI_APP_CONNECT dictionary in the app's Info.plist.

NOTE: The MobileIron server administrator can also override the Open In policy for this scenario by adding the key MI_AC_DISABLE_SCHEME_BLOCKING with the value `true` to the app's app-specific configuration.

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 the MobileIron server. For an overview of this feature, see Data loss prevention policies.

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 the MobileIron server, 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 MobileIron server 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 AppTunnel with HTTP/S tunneling, as described in AppTunnel. 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 AppTunnel with HTTP/S tunneling 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 with HTTP/S tunneling, 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 with HTTP/S tunneling. 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 with HTTP/S tunneling.

IMPORTANT:

- Wait for the AppConnectCordova event `'appconnect.isReady'` before calling `AppConnectCordova.allowLocalCachingForTunneledRequests()`.
- Do not cache sensitive data.

iOS active state change events due to AppConnect control switches

Control switches from an AppConnect app to the MobileIron client app and then back to the app in certain situations. You can receive events when the app is about to move from or to the iOS active state due to these AppConnect control switches.

Use these events to preserve your app's state before it re-signs from the iOS active state, and restore your app's state when it moves back to the iOS active state. For example, if your app is in full screen mode, preserve that fact so that the app can return to full screen mode.

Implement event listeners for the following events:

- `'appconnect.applicationWillResignActiveForAppConnect'`
- `'appconnect.applicationDidBecomeActiveFromAppConnect'`

NOTE: If the event listeners are not implemented in `'appconnect.applicationWillResignActive'` and `'appconnect.applicationDidBecomeActive'`, the app's state is not updated immediately in the MobileIron client.

Situations that trigger the state change notifications

The following situations trigger the iOS active state change notifications:

- The app checkin interval expires while an AppConnect app is running. The MobileIron client app gets AppConnect policy updates for all the AppConnect apps, and then control switches back to the app that was running.
- The auto-lock time expires while an AppConnect is running.

NOTE: The following conditions also cause control to switch to the MobileIron client app, but do not trigger the state change notifications:

- the first time an app is launched
- the first time an app is relaunched after iOS terminated it
- after the device is powered on and the device user first launches an AppConnect app.
- after the device user logs out of secure apps in the MobileIron client app, and then relaunches an AppConnect app.

Furthermore, if control switches to the MobileIron client app, but, due to user actions, does not directly switch back to the app, the AppConnect Cordova Plugin does not generate the `'appconnect.applicationDidBecomeActiveFromAppConnect'` event. For example, the event is not generated if

control switches from the app to the MobileIron client app because the auto-lock time expires, but the user presses the Home button instead of entering the AppConnect passcode.

Upload progress for AppTunnel data

The AppConnect library and the MobileIron client app are responsible for tunneling network connections using AppTunnel with HTTP/S tunneling.

The AppConnect for iOS Cordova Plugin generates the `'appconnect.uploadProgressDidChange'` event when an AppConnect app uploads data through AppTunnel. The event is triggered only for URLs which are tunneled by AppConnect as configured in the MobileIron unified endpoint manager (UEM). The MobileIron UEM are: MobileIron Core and MobileIron Cloud.

The 'appconnect.uploadProgressDidChange' event

To allow your app to display the upload progress for data uploaded through AppTunnel, add the following event listener to your document object:

```
'appconnect.uploadProgressDidChange'
```

The following table describes the parameters.

TABLE 5. APPCONNECT.UPLOADPROGRESSDIDCHANGE EVENT PARAMETERS

Parameters	Description
requestURL	the requested URL
bytesWritten	the number of bytes written in the latest write
totalBytesWritten	the total number of bytes written for the connection with this request
totalBytesExpectedToWrite	the number of bytes the connection expects to write NOTE: In some cases, such as for streams, the totalBytesExpectedToWrite parameter may be 0. The appconnect.uploadProgressDidChange event is sent on NSURLSession delegate callback. The parameters from the NSURLSession delegate callback (URLSession:task:didSendBodyData:totalBytesSent:totalBytesExpectedToSend:) are forwarded to appconnect.uploadProgressDidChange event. See also Apple developer documentation: https://developer.apple.com/documentation/foundation/nsurlsessiontaskdelegate/1408299-urlsession?language=objc

Example

```
document.addEventListener('appconnect.uploadProgressDidChange',
```

```
                                        this.uploadProgressDidChange,
                                        false);
...
uploadProgressDidChange: function (event) {
        console.log("(" + event.requestURL + ")uploadProgressDidChange: " + event.bytesWritten +
"B, " + event.totalBytesWritten + "B, " + event.totalBytesExpectedToWrite + "B");
        document.getElementById('progress').innerHTML = "<font color=\"green\">Uploading " +
event.totalBytesWritten + "B of " + event.totalBytesExpectedToWrite + "B</font>";
}
```

Developing third-party dual-mode apps

- What is a dual-mode app?
- Dual-mode app states
- High-level dual-mode app behavior
- Dual-mode API details

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 the MobileIron server.

When running as an AppConnect app, the app is in *AppConnect Mode*, because MobileIron, through the MobileIron server, the MobileIron client app, 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:

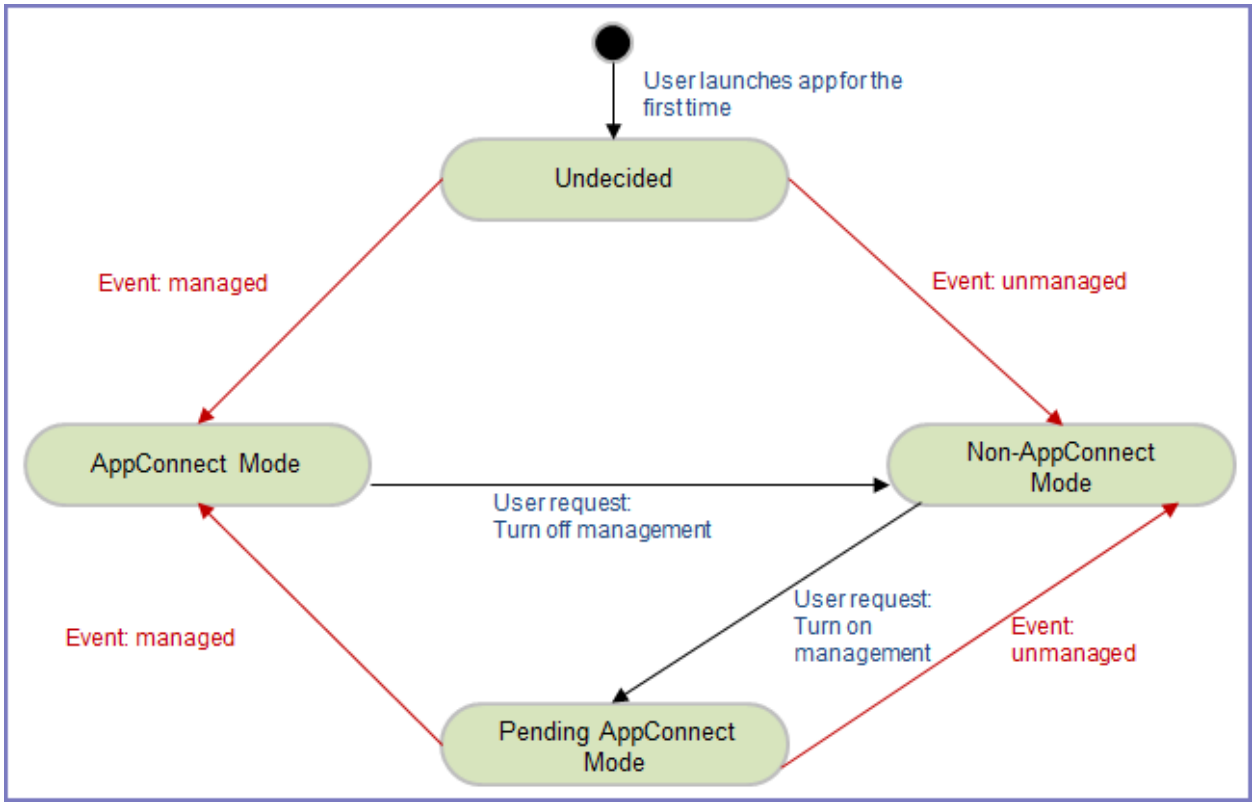
- Undecided

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 for more information about these state transitions.

FIGURE 4. DUAL-MODE APP 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 the MobileIron client app 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, the MobileIron server 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.

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.

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, the MobileIron server 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.

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.

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
- API call sequence when user requests Non-AppConnect Mode
- API call sequence when user requests Non-AppConnect Mode

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.

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

The retire method

The following method informs the AppConnect library that the app is retiring. Normally, the MobileIron server 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.

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.

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
- Allow the user to enter credentials manually
- Limit the size of configuration data from the MobileIron server
- Consider limitations when using the iOS simulator
- Enable the AppConnect library to blur screens when the app becomes inactive
- Do not put secure data in the app bundle
- Indicate to the user that the app is initializing
- Disallow custom keyboard use
- Provide documentation about your app to the MobileIron server administrator

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, a MobileIron server administrator can set up configuration information for your app on the server. 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 the MobileIron server. 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 the MobileIron server

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

As described in Configuration specific to the app, a server administrator can set up configuration information for your app on the server. 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 the MobileIron server, a large item can impact server performance. It can also slow connectivity between the MobileIron server and the MobileIron client app for iOS app. A very large item can possibly cause the communication protocol between the MobileIron server and the MobileIron client app to fail entirely.

Consider limitations when using the iOS simulator

To fully test an AppConnect app, debug on a tethered device using Xcode, as you would for any other app. On a device, your testing includes the MobileIron client app app, which is necessary for the complete flow of data from the MobileIron server 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 behave 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 the MobileIron client app is available, and the MobileIron client app 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.

Enable the AppConnect library to blur screens when the app becomes inactive

AppConnect 4.0 for iOS added support for blurring screens when the app becomes inactive. Use this capability of the AppConnect library, as described in Enable screen blurring. If your app provided its own screen blurring, remove that code. By using the AppConnect library's screen blurring capability, all AppConnect apps behave consistently.

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.

Disallow custom keyboard use

Custom keyboard extensions sometimes send data to servers when a device user enters data into an app. They send this data for assistance with word-prediction, for example. This behavior has potential for harmful data loss. Therefore, code your app to not allow custom keyboard use. MobileIron server administrators can control whether your app can use a custom keyboard by specifying a key-value pair (MI_AC_IOS_ALLOW_CUSTOM_

KEYBOARDS) on your app's configuration. Your app can control whether custom keyboards are allowed if the server administrator has enabled the key-value pair.

Related topics

Custom keyboard control

Provide documentation about your app to the MobileIron server administrator

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

- whether your app enforces the print policy.
  The server 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 handles the pasteboard policy.
  The server administrator needs to know whether allowing or not allowing your app to copy content to the pasteboard has impact on your app's behavior.
  Although the AppConnect library enforces the pasteboard policy, inform the MobileIron server administrator if your app enables or disables any special user interfaces depending on the policy status. This documentation allows the administrator to better understand your app's expected behavior.
- whether your app handles the Open In policy.
  Although the AppConnect library enforces the Open In policy, inform the MobileIron server administrator if your app enables or disables any special user interfaces depending on the policy status. This documentation allows the administrator to better understand your app's expected behavior.
  Also, if 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, specify whether your app expects to work with AppConnect with HTTP/S tunneling, or whether it requires AppConnect with TCP tunneling.
  Also, provide information about the 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.
- If your app is a dual-mode app, provide dual-mode app behavior.

  - Provide expected behavior and features in AppConnect mode versus non-AppConnect mode.
  - If your app allows the device user to switch between AppConnect mode and non-AppConnect mode, document what the device user must do.
- Whether your app uses the AppConnect-provided screen blurring capability
  Server administrators need to know whether your app will be impacted if they disable screen blurring for your app.
- Whether your app includes the MI_AC_DISABLE_SCHEME_BLOCKING key set to YES in its Info.plist.

Testing for third-party app developers

- Third-party AppConnect app testing overview
- Set up MobileIron Core
- Set up your end-user device
- Test authorization status handling
- Test data loss prevention policy handling
- Test AppConnect configuration change handling
- Test using AppTunnel
- Test the app documentation

Third-party AppConnect app testing overview

Test your app using the instructions in this chapter or the instructions in Testing for in-house app developers based on the following table:

Your role	Testing instructions
Third-party app developer	This chapter
In-house app developer whose organization uses MobileIron Cloud	This chapter
In-house app developer whose organization uses MobileIron Core or Connected Cloud.	See Testing for in-house app developers.

Testing with MobileIron Core as described in this chapter 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.

For testing your app, MobileIron provides you access to MobileIron Connected Cloud, the cloud offering of the on-premise server MobileIron Core. MobileIron also provides you access to Standalone Sentry if necessary. You then use a web portal called the Admin Portal to make configuration changes necessary for testing your app.

NOTE: Apps that you test with MobileIron Connected Cloud and Mobile@Work will also work with MobileIron Cloud and supported versions of MobileIron Go.

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:

- Contact MobileIron to provide you with a Core (Connected Cloud) and (if necessary) Standalone Sentry.
- Get Mobile@Work from the Apple App Store.

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.
2. Enable AppConnect on MobileIron Core.
3. Configure the AppConnect global policy.
4. Create an AppConnect container policy.

NOTE: These instructions are for Core 9.7.0.0.

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.
2. Select Additional Products > Licensed Products.
3. Select AppConnect For Third-party And In-house Apps if it is not already selected.
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 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.
2. Install your app on the device.
3. Set up the AppConnect passcode on the device.

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 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 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 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.
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 Select AppConnect Apps if you want the device user to be able to copy content from the AppConnect app and paste it into only other AppConnect apps.
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. 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

DLP policy	Description
	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	• verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special 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.
Allow copy/paste to for AppConnect Apps only	• verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special user interface that offers the ability to cut or copy data is available and enabled. • verify that the user can paste the data from the pasteboard only into other AppConnect apps. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Do not allow copy/paste to	• verify that the user cannot to cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special 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	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for AppConnect apps	Verify that your app enables user interfaces, if any, that give the user the option to use Open In.

Policy change	What to verify
	Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for whitelisted apps	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. Also, verify that your app calls the `openInPolicyApplied()` method.
Do not allow open in	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. 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 > App 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 Certificate Enrollment or Certificates setting Certificate Enrollment and Certificate settings that are configured in Policies & Configs > Configurations appear in the dropdown list. When you choose a Certificate Enrollment or Certificate 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 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.
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 HTTP and HTTPS 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 HTTP/S 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 with HTTP/S tunneling, do these high-level steps:

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

Enable AppTunnel on MobileIron Core

To enable AppTunnel on MobileIron Core:

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

Use an existing certificate

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel with HTTP/S tunneling. This device authentication involves:

- Providing Standalone Sentry with a root certificate.
- Providing the device with an identity cert to present to the Standalone Sentry. The identity cert is provisioned from the certificate authority (CA) that originated the root certificate.

If you already have an existing certificate, typically a .p12 file, you can use it for both purposes.

To upload the certificate to MobileIron Core:

1. In the Admin Portal, go to Policies & Configs > Configurations.
2. Select Add New > Certificate Enrollment > Single File Identity.
3. For Name, enter any name.
   For example: Tunneling Identiity Certificate
4. For Certificate 1, click Browse to select the .p12 or .pfx file of the identity certificate.
5. For Password 1, enter the password for the certificate's private key, if applicable.
6. Click Save.

Generate a certificate

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel with HTTP/S tunneling. This device authentication involves:

- Providing Standalone Sentry with a root certificate.
- Providing the device with an identity cert to present to the Standalone Sentry. The identity cert is provisioned from the certificate authority (CA) that originated the root certificate.

One convenient way to get these certiinvolves making MobileIron Core a local certificate authority (CA).

This process involves the following high-level steps:

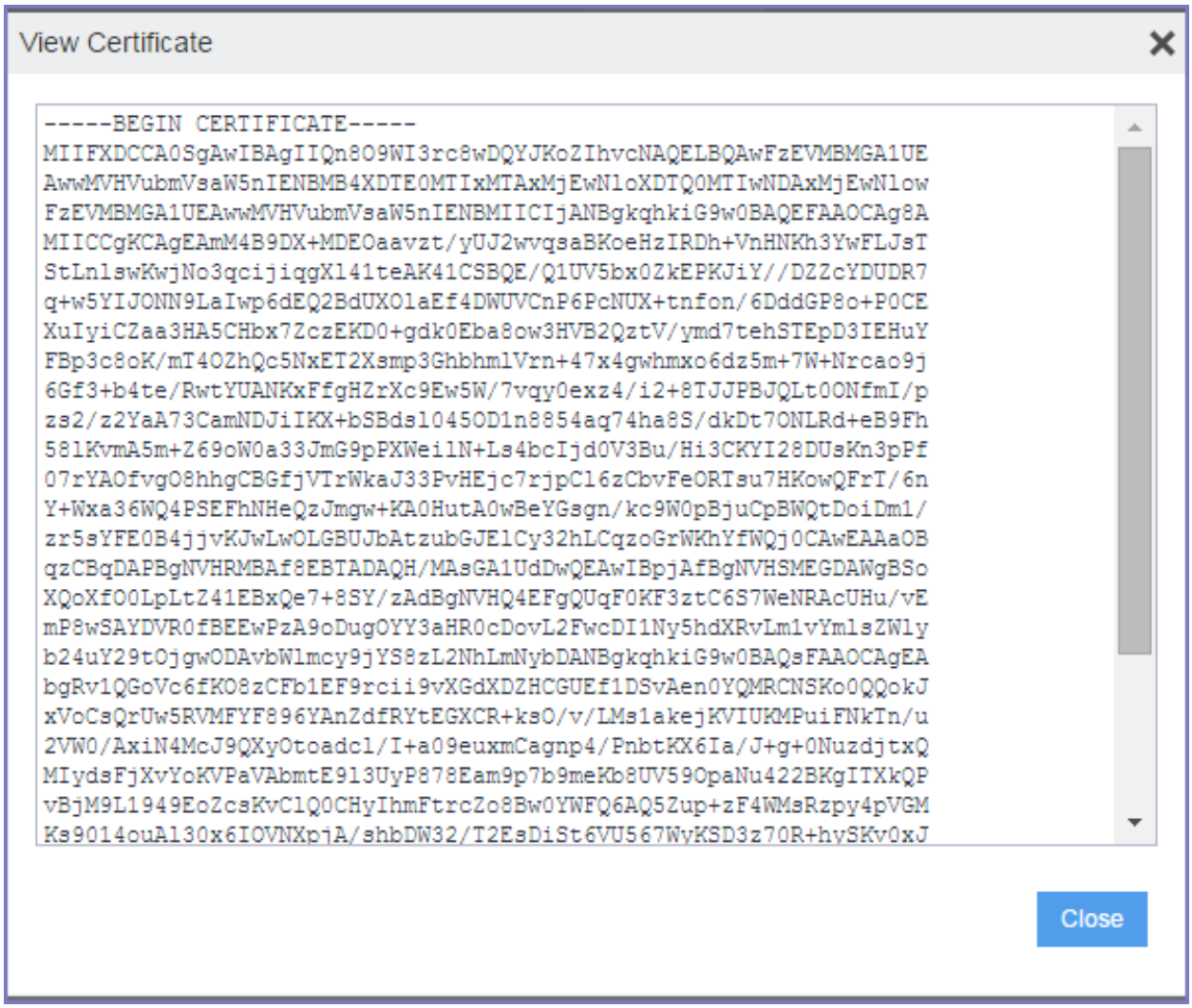
1. Create a certificate authority for using an AppTunnel with HTTP/S tunneling
2. Create a local certificate enrollment setting

Create a certificate authority for using an AppTunnel with HTTP/S tunneling

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

1. In the Admin Portal, select Services > 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.
   A screen titled Certificate Template displays.
7. Click Save.
8. Click View Certificate next to your new local certificate authority.

FIGURE 5. VIEW CERTIFICATE DISLAY



9. Copy all the text in into a text file.
10. Save the text file.
    You will upload this text file later as the root certificate for authenticating devices to the Standalone Sentry.

Create a local certificate enrollment setting

After you configure MobileIron Core as a local CA, you create a local certificate enrollment setting. This setting configures MobileIron Core acting as a local CA to generate identity certificates for the devices to present to Standalone Sentry.

To create a local certificate enrollment setting:

1. In the Admin Portal, select Policies & Configs > Configurations
2. Select Add New > Certificate Enrollment > Local.
   A dialog appears entitled New Local Certificate Enrollment Setting.
3. Enter a descriptive name in the Name field.
   For example: Tunneling certificate

4. For Local CA, select the certificate authority you created for AppTunnel.
5. For Subject, enter “cn=tunneling”.
   The value can be any string.
6. For Key Length, select 2048.
7. Click Issue Test Certificate.
   The issued test certificate displays.
8. Click OK to close the displayed certificate.
9. Click Save to save the local certificate enrollment setting.

Configure the Sentry with an AppTunnel service

To support AppTunnel with HTTP/S tunneling, configure the Sentry with the internal servers that your app uses.

Do the following:

1. In the Admin Portal, go to Services > 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 local certificate authority, select Identity Certificate.
6. Click Upload Certificate.
   If you already had a certificate, upload it.
   If you created a local certificate authority, upload the certificate text file that you created in Create a certificate authority for using an AppTunnel with HTTP/S tunneling. It is the root certificate for authenticating devices to the Standalone Sentry.
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.

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.

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.

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 with HTTP/S tunneling on an AppConnect app configuration:

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

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 with HTTP/S tunneling.
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 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 AppTunnel with HTTP/S tunneling.
   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 setting that you created in Use an existing certificate.

If you created a local certificate authority, select the local certificate enrollment setting that you created in Create a local certificate enrollment setting. This selection will result in the device receiving an identity certificate from Core that it will present to the Standalone Sentry for device authentication.

11. Click Save.

If you are creating a new AppConnect app configuration:

1. Select the new AppConnect app configuration.
2. Select 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 with HTTP/S tunneling 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.
- whether your app makes HTTPS connections that use the AppConnect feature for certificate authentication to an enterprise service.
- expected dual-mode behavior.

Test whether your app correctly handles what your documentation specifies.

For more information, see Provide documentation about your app to the MobileIron server administrator.

Testing for in-house app developers

- In-house AppConnect app testing overview
- Set up MobileIron Core
- Set up your end-user device
- Test authorization status handling
- Test data loss prevention policy handling
- Test AppConnect configuration change handling
- Test using AppTunnel
- Test the app documentation

In-house AppConnect app testing overview

Test your app using the instructions in this chapter or the instructions in Testing for third-party app developers based on the following table:

Your role	Testing instructions
In-house app developer whose organization uses MobileIron Core or Connected Cloud.	This chapter.
In-house app developer whose organization uses MobileIron Cloud	See Testing for third-party app developers
Third-party app developer	See Testing for third-party app developers

Testing with MobileIron Core as described in this chapter 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 an in-house AppConnect app developer, contact your organization's Core administrator to get access to a Core and Standalone Sentry (if necessary) 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.

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.
2. Enable AppConnect on MobileIron Core.
3. Create a label for testing your app.
4. Upload your app to MobileIron Core if you use AppConnect.plist.
5. Verify your AppConnect.plist settings.
6. Configure the AppConnect global policy.
7. Create an AppConnect container policy, if necessary.

NOTE: These instructions are for Core 9.7.0.0.

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.
2. Select Additional Products > Licensed Products.
3. Select 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 Devices & Users > 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. Select Manual for the Type.
6. 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 Catalog.
2. Select iOS for Platform.
3. Click Add+.
   The iOS Add App Wizard starts.
4. Click In-House.
5. Click Browse to select your app's .ipa file.
6. Click Next.
7. Click Next.
8. Click Finish.
   The app is now in Core's App Catalog. Core has created an AppConnect container policy and AppConnect app configuration based on your AppConnect.plist.
9. Select the row listing your app.
10. Select Actions > Apply To Label.
11. Select the label that your created in Create a label for testing your app.
12. Click Apply.
    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 prevention 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.
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 Actions > Apply To Label.
8. Select the test label that you created in Create a label for testing your app.
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.
2. Install your app on the device.
3. Set up the AppConnect passcode on the device.

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 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.
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 Actions > Remove From Label.
4. Select the label that you created in Create a label for testing your app.
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 Actions > Apply To Label.
4. Select the label that you created in Create a label for testing your app.
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.
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 Select AppConnect Apps if you want the device user to be able to copy content from the AppConnect app and paste it into only other AppConnect apps.
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	• verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special 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.
Allow copy/paste to for AppConnect Apps only	• verify that the user can cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special user interface that offers the ability to cut or copy data is available and enabled. • verify that the user can paste the data from the pasteboard only into other AppConnect apps. Also, verify that your app calls the `AppConnectCordova.pasteboardPolicyApplied()` method.
Do not allow copy/paste to	• verify that the user cannot to cut or copy text, images, or other data to the pasteboard. • where appropriate, verify that any special 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	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for AppConnect apps	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.
Allow open in for whitelisted apps	Verify that your app enables user interfaces, if any, that give the user the option to use Open In. Also, verify that your app calls the `AppConnectCordova.openInPolicyApplied()` method.

Policy change	What to verify
Do not allow open in	Verify that your app disables user interfaces, if any, that give the user the option to use Open In. 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.

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 > App 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 Certificate Enrollment or Certificates setting Certificate Enrollment and Certificate settings that are configured in Policies & Configs > Configurations appear in the dropdown list. When you choose a Certificate Enrollment or Certificate 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 Actions > Apply To Label.
11. Select the label that you created in Create a label for testing your app.
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.
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 HTTP and HTTPS 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 HTTP/Stunneling 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 with HTTP/S tunneling, do these high-level steps:

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

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.
2. Select Additional Products > Licensed Products.
3. Select AppConnect For Third-party And In-house Apps if it isn't already selected.
4. Select AppTunnel For Third-party And In-house Apps if it isn't already selected.
5. Click Save.

Use an existing certificate

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel with HTTP/S tunneling. This device authentication involves:

- Providing Standalone Sentry with a root certificate.
- Providing the device with an identity cert to present to the Standalone Sentry. The identity cert is provisioned from the certificate authority (CA) that originated the root certificate.

To upload the certificate to MobileIron Core:

1. In the Admin Portal, go to Policies & Configs > Configurations.
2. Select Add New > Certificate Enrollment > Single File Identity.
3. For Name, enter any name.
   For example: Tunneling Identiity Certificate
4. For Certificate 1, click Browse to select the .p12 or .pfx file of the identity certificate.
5. For Password 1, enter the password for the certificate's private key, if applicable.
6. Click Save.

Generate a certificate

Standalone Sentry only allows AppConnect apps on authenticated devices to use AppTunnel with HTTP/S tunneling. This device authentication involves:

- Providing Standalone Sentry with a root certificate.
- Providing the device with an identity cert to present to the Standalone Sentry. The identity cert is provisioned from the certificate authority (CA) that originated the root certificate.

One convenient way to get these certiinvolves making MobileIron Core a local certificate authority (CA).

This process involves the following high-level steps:

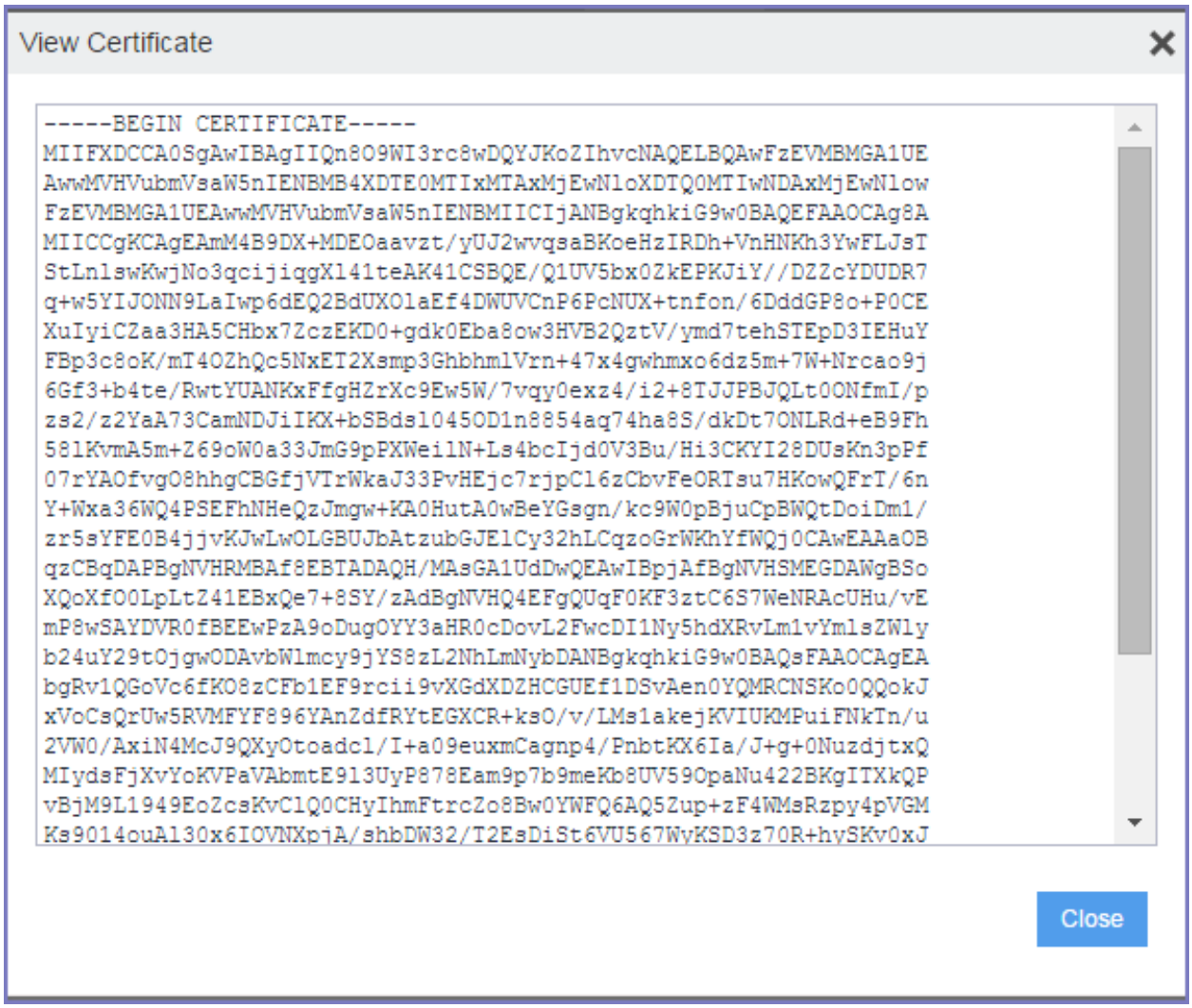
1. Create a certificate authority for using an AppTunnel with HTTP/S tunneling
2. Create a local certificate enrollment setting

Create a certificate authority for using an AppTunnel with HTTP/S tunneling

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

1. In the Admin Portal, select Services > 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.
   A screen titled Certificate Template displays.
7. Click Save.
8. Click View Certificate next to your new local certificate authority.

FIGURE 6. VIEW CERTIFICATE DISPLAY



9. Copy all the text into a text file.
10. Save the text file.
    You will upload this text file later as the root certificate for authenticating devices to the Standalone Sentry.

Create a local certificate enrollment setting

After you configure MobileIron Core as a local CA, you create a local certificate enrollment setting. This setting configures MobileIron Core acting as a local CA to generate identity certificates for the devices to present to Standalone Sentry.

To create a local certificate enrollment setting:

1. In the Admin Portal, select Policies & Configs > Configurations
2. Select Add New > Certificate Enrollment > Local.
   A dialog appears entitled New Local Certificate Enrollment Setting.
3. Enter a descriptive name in the Name field.
   For example: Tunneling certificate

4. For Local CA, select the certificate authority you created for AppTunnel.
5. For Subject, enter “cn=tunneling”.
   The value can be any string.
6. For Key Length, select 2048.
7. Click Issue Test Certificate.
   The issued test certificate displays.
8. Click OK to close the displayed certificate.
9. Click Save to save the local certificate enrollment setting.

Configure the Sentry with an AppTunnel service

To support AppTunnel with HTTP/S tunneling, configure the Sentry with the internal servers that your app uses.

Do the following:

1. In the Admin Portal, go to Services > 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 local certificate authority, select Identity Certificate.
5. Click Upload Certificate.
   If you already had a certificate, upload it.
   If you created a local certificate authority, upload the certificate text file that you created in Create a certificate authority for using an AppTunnel with HTTP/S tunneling. It is the root certificate for authenticating devices to the Standalone Sentry.
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.

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.

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.

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 > App Configuration.
2. Select Add New > AppConnect > App Configuration.

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 with HTTP/S tunneling.
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 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 AppTunnel with HTTP/S tunneling.
   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 setting that you created in Use an existing certificate.

If you created a local certificate authority, select the local certificate enrollment setting that you created in Create a local certificate enrollment setting. This selection will result in the device receiving an identity certificate from Core that it will present to the Standalone Sentry for device authentication.

11. Click Save.

If you are creating a new AppConnect app configuration:

1. Select the new AppConnect app configuration.
2. Select Actions > Apply To Label.
3. Select the label that you created in Create a label for testing your app.
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 with HTTP/S tunneling 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.
- whether your app makes HTTPS connections that use the AppConnect feature for certificate authentication to an enterprise service.
- expected dual-mode behavior.

Test whether your app correctly handles what your documentation specifies.

For more information, see Provide documentation about your app to the MobileIron server administrator.

AppConnect for iOS Cordova Plugin revision history

- AppConnect 4.7.0 for iOS Cordova Plugin revision history
- AppConnect 4.6.0 for iOS Cordova Plugin revision history
- AppConnect 4.5.3 for iOS Cordova Plugin revision history
- AppConnect 4.5.2 for iOS Cordova Plugin revision history
- AppConnect 4.5.1 for iOS Cordova Plugin revision history
- AppConnect 4.5.0 for iOS Cordova Plugin revision history
- AppConnect 4.4.2 for iOS Cordova Plugin revision history
- AppConnect 4.4.1 for iOS Cordova Plugin revision history
- AppConnect 4.4.0 for iOS Cordova Plugin revision history
- AppConnect 4.3.1 for iOS Cordova Plugin revision history
- AppConnect 4.3 for iOS Cordova Plugin revision history
- AppConnect 4.2.1 for iOS Cordova Plugin revision history
- AppConnect 4.2 for iOS Cordova Plugin revision history
- AppConnect 4.1.1 for iOS Cordova Plugin revision history
- AppConnect 4.1 for iOS Cordova Plugin revision history
- AppConnect 4.0 for iOS Cordova Plugin revision history
- AppConnect 3.5 for iOS Cordova Plugin revision history
- AppConnect 3.1.3 for iOS Cordova Plugin revision history
- AppConnect 3.1.2 for iOS Cordova Plugin revision history
- AppConnect 3.1.1 for iOS Cordova Plugin revision history
- AppConnect 3.1 for iOS Cordova Plugin revision history
- AppConnect 3.0 for iOS Cordova Plugin revision history
- Releases prior to AppConnect 3.0 for iOS Cordova Plugin revision history

AppConnect 4.7.0 for iOS Cordova Plugin revision history

This release provides the following:

- New features and enhancements summary
- Known issues
- Limitations

New features and enhancements summary

This release includes the following new features and enhancements:

- **Avoid pasteboard notifications**: To avoid pasteboard notifications on users' devices when using AppConnect apps, set up an app group for your AppConnect apps. Setting up an app group reduces flipping between the AppConnect app and the MobileIron client and avoids pasteboard notifications. For more information, see Optional: Avoiding pasteboard notifications.

Resolved issues

This release includes the following new known issues:

- **AP-5578**: Fixed an issue due to which the check-in request failed when users were prompted to authenticate.

Known issues

This release includes the following new known issues:

- **AP-5557**: On iOS 12 devices, users are not required to authenticate when using AppConnect 4.7.0 apps. This is an intermittent issue.
  Workaround: Upgrade to iOS 13 through the latest version as supported by MobileIron.
- **AP-5567**: The following issue is seen if the AppConnect Global policy is applied to some AppConnect apps and the AppConnect Container policy is applied to other AppConnect apps, and Authorize is not checked for the option "Apps without an AppConnect container policy." AppConnect apps that are packaged using the 4.7.0 SDK and do not have a container policy will not go into an unauthorized state till the next check in.
  Workaround: Shorten the check in interval to reduce the time interval where the unauthorized apps continue to work until the next check in.
- **AP-5484**: Instead of displaying a failed to authenticate error message, users are asked to contact their IT administrator.
- **AP-5482**: When users cancel Touch ID authorization, they are shown a 'Contact Admin' error message instead of a 'Failed to authenticate' user error message.

Limitations

This release includes the following new limitations:

- **AP-5497**: On iOS 12 or earlier versions as supported by MobileIron, device users are prompted for biometric authentication more than once.
- **AP-5504**: To accommodate pasteboard changes in iOS 14, users cannot copy from AppConnect 4.6 (previous versions of AppConnect apps as supported by MobileIron) apps to AppConnect 4.7 apps if the Data Loss Prevention policy only allows copy/paste to other AppConnect apps. Copy/paste works when copying from 4.7 to older apps or when the policy is set to allow all apps.
- **AP-5563**: Since the Pasteboard is used for sending logs, notifications are seen when sending AppConnect app logs.
- **AP-5559**: If an AppConnect app is set up to avoid pasteboard notifications (MI_AC_ACCESS_GROUP key in added to the app's Info.plist file), sometimes the MobileIron client may log out while the user is still using the AppConnect app. As a result, users are prompted to authentication when they activate the app.
- **AP-5553**: Configuration for an AppConnect are seen as Pending in the MobileIron client in Settings > Secure apps even after users launch the app is authorized.

AppConnect 4.6.0 for iOS Cordova Plugin revision history

This release provides the following:

- New features summary
- Resolved issues

New features summary

This release includes the following new features and enhancements:

- **Improvements to memory consumption**: Secure File I/O APIs have been optimized to decrease memory consumption while processing large files.
- **Two Cordova plugin variants**: Due to Apple deprecating the UIWebView class, the AppConnect for iOS SDK is available in two variants: one with UIWebView and WKWebView support, and another with WKWebView support, but no UIWebView support. The AppConnect SDK without UIWebView support is provided for apps that will be submitted to the App Store. The Cordova plugin included with each variant of the SDK provides the same support as the SDK variant.
  See Cordova Plugin variants and AppConnect Cordova Plugin contents.

Resolved issues

This release includes the following new resolved issues:

- **AP-5422**: Fixed issue with tunneled requests authentication when app implements `URLSession:didReceiveChallenge:completionHandler:` method of the URLSessionDelegate protocol.

- **AP-5328**: Fixed an issue where AppConned apps flipped to the MobileIron client app for password authentication. Now the passcode prompt is seen within the wrapped app.

AppConnect 4.5.3 for iOS Cordova Plugin revision history

This release provides the following:

- Resolved issues

Resolved issues

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

- AP-5376, APG-1177: Fixed an issue where redirected server requests could fail to connect.

AppConnect 4.5.2 for iOS Cordova Plugin revision history

This release does not provide any updates to the SDK.

AppConnect 4.5.1 for iOS Cordova Plugin revision history

This release does not provide any updates to the SDK.

AppConnect 4.5.0 for iOS Cordova Plugin revision history

This release provides the following:

- Resolved issues
- Known issues

Resolved issues

This release provides the following new resolved issues:

- AP-5256: Workaround for a bug in a third-party app security framework, which caused a crash when used with AppConnect.
- AP-5241: Fixed crash in [ACAppInterfaceBus displayMessage:scheme:completion:].
- AP-5199: Sometimes AppConnect apps failed to unlock using biometric authentication if the device passcode was set as the fallback option. Users may have seen this issues if the Check-in interval and the AutoLock interval are small and equivalent. This issue is fixed.

Known issues

This release includes the following new known issues:

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

AppConnect 4.4.2 for iOS Cordova Plugin revision history

This release provides the following:

- Resolved issues
- Known issues

Resolved issues

This release provides the following new resolved issues:

- AP-5245: Fixed a Secure File I/O thread-safety issue which could cause I/O errors when writing to multiple files simultaneously. Note that I/O to individual files should always be done from a single thread.
- AP-5253: Fixed an exception when launching apps in Xcode's Simulator.

Known issues

This release includes the following new known issue:

- AP-5252: Web@Work 2.9.0.0 for iOS with Chromium does not trust some sites. For more information, see the following Knowledge Base article in the MobileIron Community: Web@Work - Certain sites may not be trusted when using Chromium engine.

AppConnect 4.4.1 for iOS Cordova Plugin revision history

This release provides the following:

- Resolved issues
- Known issues

Resolved issues

This release includes the following new resolved issues:

- AP-5233: Under certain conditions when adding cookies to a network request, the cookies were dropped after receiving an HTTP 302 redirect. This issue is fixed.

Known issues

This release includes the following new known issues:

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

AppConnect 4.4.0 for iOS Cordova Plugin revision history

This release provides the following:

- New features summary
- Resolved issues
- Limitations

New features summary

This release includes the following new features and enhancements:

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

Resolved issues

This release provides the following new resolved issues:

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

- AP-5206: Previously, the AppConnect for iOS SDK was not calling applicationDidBecomeActive. This issue is fixed.

Limitations

This release includes the following new limitations:

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

AppConnect 4.3.1 for iOS Cordova Plugin revision history

New features

- **Track upload progress for data tunneled through AppTunnel**
  A new event `'appconnect.uploadProgressDidChange'` is provided to track the progress of data uploads through AppTunnel.
  See Upload progress for AppTunnel data.
- Support for the armv7s architecture is deprecated.

Resolved issues

This release provides the following new resolved issue:

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

AppConnect 4.3 for iOS Cordova Plugin revision history

New features

- **Support for MobileIron AppStation**
  Apps built with the AppConnect 4.3.0 for iOS Cordova Plugin can run with MobileIron AppStation as the MobileIron client app instead of MobileIron Go. Administrators can use MobileIron AppStation on devices which are interacting with a MobileIron Cloud tenant that supports Mobile Apps Management (MAM) but not Mobile Device Management (MDM).
  For your AppConnect app to support AppStation:
  - Declare the `alt-appconnect`URL scheme in your app's Info.plist as another allowed URL scheme.
    See Declare the AppConnect URL schemes as allowed.
    Also see Upgrade tasks or Getting started tasks.
- **Support for Open From data loss prevention policy**

The AppConnect 4.3.0 for iOS Cordova Plugin adds support for the Open From data loss protection policy. The AppConnect library enforces the policy as configured on the MobileIron server. No Cordova APIs are added and no app changes are necessary.

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

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

AppConnect 4.2.1 for iOS Cordova Plugin revision history

New features

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

AppConnect 4.2 for iOS Cordova Plugin revision history

This release of the AppConnect for iOS Cordova Plugin has no new features.

Resolved issues

- AP-4919: Fixed an issue that caused an AppConnect app to crash when it used the same object as a delegate for multiple UI elements.
- AP-4150: After an AppConnect SDK or Cordova app became inactive and the AppConnect library blurred the screen, a noticeable delay occurred when removing the blur when the app became active. This issue has been fixed.

Known issues

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

AppConnect 4.1.1 for iOS Cordova Plugin revision history

This AppConnect release has no new features.

Resolved issues

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

Known issues

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

AppConnect 4.1 for iOS Cordova Plugin revision history

This AppConnect release has several new features. It has no new known or resolved issues or limitations.

New features

- Certificate pinning support
- Lock AppConnect apps when screen is off
- Overriding the Open In Policy for OpenURL with the mailto: scheme

Certificate pinning support

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

Using certificate pinning requires:

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

This feature requires no additional development in the app.

Lock AppConnect apps when screen is off

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

This feature requires:

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

This feature requires no additional development in the app.

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

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

The AppConnect library overrides the Open In policy for native email if either of the following are true:

- The MobileIron server administrator added the key MI_AC_DISABLE_SCHEME_BLOCKING with the value `true` to the app's app-specific configuration.
- The app added the key MI_AC_DISABLE_SCHEME_BLOCKING with the value `YES` in the MI_APP_CONNECT dictionary in the app's Info.plist.

THE MI_APP_CONNECT dictionary is new in this release.

See Open In policy API details .

AppConnect 4.0 for iOS Cordova Plugin revision history

New features

- iOS 8 no longer supported
- Drag and Drop data loss prevention policy support
- Native email control using the Open In DLP policy
- App extension control using the Open In DLP policy
- Custom keyboard use controlled by MobileIron server
- Screen blurring
- Requirement for Face ID usage Info.plist entry
- Support for sending AppConnect logs from Mobile@Work
- Automatic policy status updates sent to MobileIron server
- Support for storing AppConnect library encryption keys in the Secure Enclave

iOS 8 no longer supported

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

See Product versions required .

Drag and Drop data loss prevention policy support

MobileIron server administrators can set a drag and drop policy for each AppConnect app. It specifies whether AppConnect apps can drag content to all other apps, to only other AppConnect apps, or not at all. The AppConnect library enforces this policy. Your app provides no code to support the drag and drop policy.

NOTE: This feature is not supported with MobileIron Cloud.

See Data loss prevention policies.

Native email control using the Open In DLP policy

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

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

App extension control using the Open In DLP policy

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

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

Custom keyboard use controlled by MobileIron server

The MobileIron server can now control custom keyboard use by your AppConnect app. If the administrator does not configure this choice, your app can choose to reject custom keyboard use.

See Custom keyboard control.

Screen blurring

AppConnect 4.0 for iOS adds support for blurring screens when the app becomes inactive. If your app provided its own screen blurring, remove that code. By using the AppConnect library's screen blurring capability, all AppConnect apps behave consistently.

To enable screen blurring, add the key MI_AC_PROVIDE_SCREEN_BLUR to your app's Info.plist as a Boolean. Set the value to YES.

When you set the Info.plist key MI_AC_PROVIDE_SCREEN_BLUR to YES, the MobileIron server administrators can disable screen blurring by setting a key-value pair on the server for your app's configuration. The server key is MI_AC_ENABLE_SCREEN_BLURRING with the value false.

See Enable screen blurring.

Requirement for Face ID usage Info.plist entry

Include **Privacy - Face ID Usage Description** to your app's info.plist, with a string value indicating the purpose of Face ID use. For example, add the value **AppConnect**. If you manually add this key, its name is NSFaceIDUsageDescription.

Server administrators can allow the use of Touch ID or Face ID instead of an AppConnect passcode. Therefore, this Info.plist entry is required on iOS 11 through the most recently released version as supported by MobileIron.

See Allow Face ID.

Support for sending AppConnect logs from Mobile@Work

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

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

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

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

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

Automatic policy status updates sent to MobileIron server

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

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

This change has no impact on your app's implementation. Your app should continue to always call the appropriate notification acknowledgment method:

Support for storing AppConnect library encryption keys in the Secure Enclave

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

To benefit from this feature, the device must:

- have Apple's Secure Enclave hardware.

Devices that have biometric security have Secure Enclave hardware.

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

MobileIron Go does not support this feature.

Resolved issues

- AP-4446: Fixed an issue where the authStateChangeTo event was not called when using the AppConnect Cordova Plugin.
- AP-4202: Custom protocol classes set to NSURLSessionConfiguration were previously ignored in AppConnect apps. This issue has been fixed.
- AP-4133: Added ability to use NSURLConnection with NSURLSession networking with AppTunnel.

Known issues

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

Limitations

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

AppConnect 3.5 for iOS Cordova Plugin revision history

New features

iOS 11 compatibility

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

IMPORTANT: Upgrade your app to use the AppConnect 3.5 for iOS Cordova Plugin for your app to run on iOS 11 devices. Apps using Plugin versions prior to 3.1.3 crash on iOS 11 devices. Apps using version 3.1.3 do not crash, but the AppConnect library does not handle the pasteboard data loss prevention policy correctly.

For more information, see Product versions required on page 22.

Open In changes

This version of the AppConnect for iOS Cordova Plugin has the following changes to Open In handling:

- The AppConnect library supports a new key-value pair from the MobileIron server that tells the library not to enforce the Open In policy. A MobileIron server administrator determines if this behavior is appropriate for an enterprise. An app makes no changes relating to this feature.
  See “Overriding the Open In Policy for the app” in the administrator documentation *MobileIron Core MobileIron Core AppConnect and AppTunnel Guide*.
- Because of iOS implementation changes, if an app uses the Objective-C class UIActivityViewController to execute Open In, when the Open In policy specifies a whitelist, Open In to *all* apps is not allowed. Therefore, use only UIDocumentInteractionController to execute Open In. For example, if you use a Cordova plugin to use the Open In capability, make sure it uses UIDocumentInteractionController.
  See Overview of Open In handling on page 54.

Resolved issues

- AP-4196: When an AppConnect Cordova app loaded an HTML page, the AppConnect for iOS Cordova Plugin did not re-initialize properly. The issue has been fixed.
- AP-4184: Fixed an installation failure of the AppConnect Cordova Plugin for apps using Cordova 7.0.0.
- AP-4145: URL requests made on a background thread were not tunneled if the AppConnect library in the app had not received the AppTunnel rules. The issue has been fixed because the AppConnect library now blocks URL requests until after it has received the AppTunnel rules.
- AP-3917: When a URL request using NTLM authentication was tunneled with AppTunnel, an error occurred when the device user was prompted with the user credentials dialog. The dialog displayed the Standalone Sentry host name instead of the URL request’s host name. The issue has been fixed.
- AP-3564: The sample Cordova app TestAppConnect did not become authorized on iOS 9 devices. The issue has been fixed.

Limitations

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

AppConnect 3.1.3 for iOS Cordova Plugin revision history

This release has no new features.

Resolved issues

- AP-4054: The HTTP error code 403 was not always reported to apps using AppTunnel. This issue has been fixed.

- AP-4149: In some cases, enterprises that used both AppTunnel and a global HTTP proxy policy resulted in AppConnect apps having no access to the network. The issue occurred when an AppTunnel rule caused a tunneling attempt for requests to the URL for the proxy auto-configuration (PAC) file. The issue occurred for all AppTunnel rules that did one of the following:
  - used a wildcard character in the AppTunnel rule's hostname such that the PAC file URL matched the rule
  - explicitly named the PAC file URL in the AppTunnel rule's hostname

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

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

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

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

AppConnect 3.1.2 for iOS Cordova Plugin revision history

This release has no new features.

Resolved issues

- AP-4062: Fixed a critical issue that caused an AppConnect app to crash if all of the following are true:
  - The app uses AppTunnel with either HTTP/S tunneling or TCP tunneling.
  - The AppConnect log level "Debug" is activated for the app.
  - The device is registered with MobileIron Core 9.4.0.0.

AppConnect 3.1.1 for iOS Cordova Plugin revision history

This release has no new features.

Resolved issues

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

AppConnect 3.1 for iOS Cordova Plugin revision history

New features

Update to OpenSSL 1.0.2h

The AppConnect library now uses OpenSSL version 1.0.2h.

Open In policy now enforced by AppConnect library

The AppConnect library now enforces the Open In policy. An app no longer enforces the policy itself. An app now only has to enable or disable any special user interfaces that give the user the option to use Open In. For example, if your app presents a menu item for Open In, the menu item should be enabled. By disabling such user interfaces, your app does not give the end user the impression that Open In is possible when the AppConnect library has disabled it.

If you are upgrading your app to use the AppConnect 3.1 for iOS SDK, make the appropriate modifications to your code.

For details, see Open In policy API details on page 53.

Resolved issues

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

Known issues

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

Limitations

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

AppConnect 3.0 for iOS Cordova Plugin revision history

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

Releases prior to AppConnect 3.0 for iOS Cordova Plugin revision history

For the revision history of releases prior to AppConnect 3.0 for iOS Cordova Plugin, see the"MobileIron AppConnect 4.2 for iOS Cordova Plugin Developers Guide", available at MobileIron AppConnect for iOS Product Documentation.