vmware airwatch

VMware AirWatch iOS SDK Technical Implementation Guide

Empowering your enterprise applications with MDM capabilities

AirWatch SDK v5.9

Have documentation feedback? Submit a Documentation Feedback support ticket using the Support Wizard on support.air-watch.com.

Table of Contents

Chapter 1:
Introduction to the AirWatch SDK for iOS

The VMware AirWatch® Software Development Kit™ (SDK) is a set of tools allowing organizations to incorporate a host of features and functionality into their custom-built iOS applications. The AirWatch SDK enhances the security and functionality of those applications and in turn helps save application development time and money.

Process Overview

Integrating an application with the AirWatch SDK can be broken down into five main steps. A high-level overview of each step is listed below.

Enable the Core SDK Framework within Xcode

These steps detail the core iOS frameworks and the AirWatch SDK frameworks that you add to your project in order for the SDK to function properly. The AirWatch SDK frameworks are made available by running the provided **AirWatch SDK.dmg** file.

In order for your custom application to use the AirWatch SDK , you must first complete the following setup procedures in Xcode:

- Add Required Frameworks
- Configure the Server Connections

The following modules enable the device management framework and allow you to configure device management features into your application:

- Implement the Beacon
- Implement the DataSampler

Select and Implement Additional SDK Modules

VMware Workspace ONE™ UEM provides a number of pre-configured functions for your app that can be controlled from the Workspace ONE UEM console. These modules make up the **Application Management Framework**. You must decide

which SDK modules to use within your application.

Developing Modules

Developers have the option, for most modules, to code the expected behavior or to set the behavior in the Workspace ONE UEM console. If you want the application to behave a certain way every time, you must code this behavior in to the application.

Coding the Logging Level

The exception is **Logging**. You must code the logging level and you must set this option in the Workspace ONE UEM console. This configuration ensures that your network is not burdened with unwanted logging activity.

Using Default Settings for SDK Profiles

Use the **Security Policies** and **Settings** pages to configure settings once and then share them across Workspace ONE UEM applications using the **iOS Default Settings @ [Organization Group]** profile.

You can also use the **Profiles** page to configure custom settings with specific behaviors.

Implementing each module into your app is a two-step process:

1. Implement the functionality for the desired module within your app (in Xcode).
2. Add the corresponding configuration to the SDK Profile (in the Workspace ONE UEM console) that gets assigned to the app.

Using Certificates

The AirWatch SDK allows you to provision and embed certificates into your app upon deployment. The process involves three main steps:

- Configuring the certificate authority (CA) and the CA Template.
- Creating the App Profile in the Workspace ONE UEM console.
- Assigning the App Profile to the application in Workspace ONE UEM prior to app deployment. See Certificate Provisioning.

Set Default Settings within Workspace ONE UEM

In the Workspace ONE UEM console, you must configure default SDK settings to assign it to your app. These settings include configurations specific to each module you plan on utilizing.

Upload the Application to Workspace ONE UEM

Once your app is completely built, you need to upload the file into Workspace ONE UEM using the Workspace ONE UEM console. During this process, you need to assign the SDK profile you created, making the settings defined in the SDK

profile available to your application. The **Mobile Application Management (MAM) Guide** describes how to upload an application.

Deploy the Application

The final step is to deploy your application to managed devices through the Workspace ONE UEM console. Users now have access to the application, along with all the SDK enabled features you've implemented.

For more information on deploying applications to managed devices, see the **Mobile Application Management (MAM) Guide**.

Migrate to the Latest SDK Version

In the latest SDK for iOS, we have updated various UI screens presented by the SDK. These pages now incorporate storyboards that require you to import the **new AWKit.bundle** included in the new SDK DMG. This **AWKit.bundle** contains the compiled storyboards required for the app to function.

We have also integrated with the latest Safari View Controller for various process flows on UI screens. Add **SafariServices.Framework** so all Workspace ONE UEM screens work without error.

Here are the general instructions based on how you use the SDK today:

1. Replace or import the AWKit.bundle.
   - Replace the older version of AWKit.bundle with the newer version provided in the DMG file if you already import AWKit.bundle into your project bundle resources today.
   - Import AWKit.bundle into your project under **Bundle Resources in Xcode Build Phases** if you have never imported the AWKit.bundle into your project.
2. Import the SafariServices.Framework into your project.

Supported Operating Systems and Requirements

The AirWatch SDK for iOS is compatible with the listed operating systems and requires the listed components.

Supported iOS Operating Systems

The AirWatch SDK for iOS supports the use of the Apple operating systems iOS 7.0+. Certain features require a newer operating system and these features are noted.

Requirements

Meet the following requirements before using the AirWatch SDK for iOS:

- To manage organization groups, get access to the Workspace ONE UEM console v9.1+ with the appropriate access rights.
- Know application development using Xcode (for more information, see https://developer.apple.com/xcode/) and

use **Xcode v6.0.1+**.

- To develop the application, get the necessary SDK DMG (AirWatch SDK.dmg) file from Workspace ONE UEM.

Chapter 2:
Getting Started

Perform the following tasks to prepare to use the AirWatch SDK for iOS.

Xcode Components

Add the listed Xcode components.

Add Required Xcode Frameworks

The SDK depends on the following frameworks to function properly. Follow the steps to add the necessary frameworks to your project.

1. Select the project in the Groups & Files pane in Xcode.
2. Ensure that the proper target is selected on the left side, select the **Build Phases** tab.
3. Expand the **Link Binary With Libraries** section.
4. Select the **+** button at the bottom of the section to add the required frameworks.
5. Select each one of the following frameworks and select **Add**.
    - **Accelerate.framework**
    - **AssetsLibrary.framework**
    - **AudioToolbox.framework**
    - **AVFoundation.framework**
    - **CFNetwork.framework**
    - **CoreData.framework**
    - **CoreFoundation.framework**
    - **CoreGraphics.framework**
    - **CoreLocation.framework**

- **CoreMedia.framework**
- **CoreMotion.framework**
- **CoreTelephony.framework**
- **CoreText.framework**
- **CoreVideo.framework**
- **Foundation.framework**
- **ImageIO.framework**
- **Libc++.tbd**
- **Libsqlite3.tbd**
- **Libz.tbd**
- **libxml2.tbd**
- **LocalAuthentication.framework**
- **MediaPlayer.framework**
- **MessageUI.framework**
- **MobileCoreServices.framework**
- **QuartzCore.framework**
- **SafariServices.framework**
- **Security.framework**
- **SystemConfiguration.framework**
- **UIKit.framework**
- **WebKit.framework**

Add Required Xcode Bundle Resources

The SDK depends on the following bundle resources to function properly, so add the necessary bundles to your project. Find some of these bundles inside the **AWSDK.framework** file structure.

- **SDKLocalization.bundle**
- **AWKit.bundle**

> **Important:** Add the **AWKit.bundle** to your project or the application can crash.

Add the AirWatch SDK Frameworks

The AirWatch SDK frameworks are made available by running the provided **AirWatch SDK.dmg** file.

- Drag **AWSDK.framework** into the Frameworks group in the sidebar of Xcode. Make sure to check **Copy items into destination group's folder**.
- Ensure the application is linking against the **AWSDK.framework**. To verify, select the **Build Phases** tab in the properties of the target. Expand the **Link Binary With Libraries** section to see if **AWSDK.framework** is added. If not, select the **+** button and select **AWSDK.framework**.
- Import the umbrella header wherever you use the AirWatch SDK. Add **#import <AWSDK/AWSDKCore.h>** to the top of the file.

Add Linker Flags

Since the AirWatch SDK uses Objective-C categories, you must pass linker flags to the linker to properly load them.

- Select the project or workspace in the **Groups & Files** pane.
- Select the target for the application.
- Select the **Build Settings** tab.
- Ensure the **-ObjC** flag is added in the **Other Linker Flags** entry.

Valid Architectures

The SDK currently supports the following architectures:

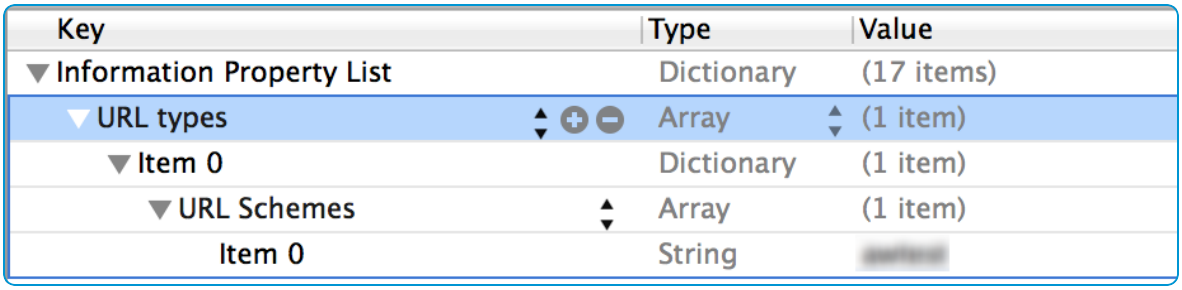
- ARMv7
- ARMv7S
- ARM64

Do not compile the i386 architecture/simulator because the SDK only supports real physical devices.

Callback Scheme Registration

To receive a callback from the AirWatch Agent, the application exposes a custom scheme in the **info.plist**.

1. Navigate to **Supporting Files** in Xcode.
2. Select the file **<YourAppName>-Info.plist**.
3. Navigate to the **URL Types** section. If it does not exist, add it at the **Information Property List** root node of the PLIST.
4. Expand the **Item 0** entry and add an entry for **URL Schemes**.
5. Set the next **Item 0** under **URL Schemes** to the desired callback scheme.



Miscellaneous Entries for the Info.PLIST File

QR Scan

Include **NSCameraUsageDescription** in the application **info.plist** file to enable the SDK to scan QR codes with the device camera.

Provide a description that devices prompt users to allow the application to enable this feature.

FaceID

Include **NSFaceIDUsageDescription** in the application **info.plist** file to enable the SDK to use FaceID.

Provide a description that devices prompt users to allow the application to enable this feature. Consider controlling the message users read. If you do not include a description, the iOS system prompts users with native messages that might not align with the capabilities of the application.

Compile With Xcode 7

If you use Xcode 7 to compile your SDK application, take these steps to ensure that the application functions properly.

1. Add an array key named **LSApplicationQueriesSchemes** to the **info.plist**.
2. Add the bundle identifier of the AirWatch Agent or AWSSOBroker2 application to the array.

   The schemes for the Agent and AWSSOBroker2 are as follows:

   - airwatch
   - AWSSOBroker2
3. Navigate to your Xcode build settings and set **Enable Bitcode** to **No**.

Initialize the SDK

Before you can use the SDK, you must initialize it. The **AWController** class is the main component responsible for initializing the SDK. In addition, it automatically handles and implements certain core SDK functionalities to improve ease of integration for developers, such as the following functions:

- Passcode mode
- Single sign-on (SSO)
- SSID filtering
- Proxy and tunneling

Calling start in **AWController** automatically sets up the proxy to redirect traffic. However, you must wait until the initial **CheckDoneWithError** callback is received before any network traffic redirects. Wait until the SDK finishes setting up before making any network calls through your proxy.

Initialization Example

The example demonstrates how to initialize the SDK.

1. Import the `<AWSDK/AWController.h>` header file. Associate the **AWControllerDelegate** to your app delegate.

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, AWSDKDelegate>
```

2. Inside the app delegate, implement the following code:

```
-(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
        // Configure the Controller by:
        AWController *controller = [AWController clientInstance];
        // 1) defining the callback scheme so the app can get called back,
        controller.callbackScheme = @"UrlScheme";
        // 2) set the delegate to know when the initialization has been completed.
        controller.delegate = self;

        return YES;
}
```

3. Start the SDK initialization inside the **applicationDidBecomeActive** delegate method.

```
-(void)applicationDidBecomeActive:(UIApplication *)application {
        [[AWController clientInstance] start];
}
```

Do not call the start method in **didFinishLaunchingWithOptions** because the SDK may display modal view controllers that rely on a reference view controller. Sometimes, when you use a storyboard, the view controllers have not yet been generated at the time **didFinishLaunchingWIthOptions** is called. To avoid any unstable behavior with the app, call `[[AWController clientInstance] start]` inside **applicationDidBecomeActive** instead.

4. Next, implement the code to handle the callback from the AirWatch Agent or Container app.

```
-(BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString
*)sourceApplication annotation:(id)annotation {
    return [[AWController clientInstance] handleOpenURL:url fromApplication:sourceApplication];
}
```

5. Implement the remaining delegate methods:

```
- (void)initialCheckDoneWithError: (NSError *)error
```

This delegate method is invoked when the SDK initializes. This method is **ALWAYS** called after the SDK passes through the initialization flow. If the initialization is successful, then the error object is nil. If the initialization fails, then the error object contains the reason code for why it fails.

```
- (void)receivedProfiles:(NSArray *)profiles
```

This delegate method is invoked when settings of an SDK profile assigned to this application update on the Workspace ONE UEM console. It notifies the app that new settings are available. The profiles array contains the list of **AWProfile** objects that contain configuration payloads.

```
- (void)unlock
```

This delegate method is invoked immediately after you initiate a new SSO session by inputting the correct password/passcode.

```
- (void)lock
```

This method is invoked when the SSO session has expired and the SDK passcode input view is displayed. It is intended for use as an indicator of when a user no longer has to access the app. This lock allows the developer to implement the necessary logic to take the proper action for when the app is locked.

```
- (void)wipe
```

This method is invoked when the SDK identifies that the device has been wiped or unenrolled from the Workspace ONE UEM console. This method is also invoked when a user reaches the limit of failed passcode attempts defined in the SDK profile.

Note: The AirWatch SDK only invokes this method, and it takes no other actions. The application developer must implement the necessary local app wipe logic.

```
- (void) stopNetworkActivity
```

This method is invoked when the device connects to an SSID that is blacklisted in the SDK profile.

```
- (void) resumeNetworkActivity
```

This method is invoked when the device connections to a valid SSID after network activity is already stopped.

iOS 9+ and the Device Services Server

For iOS 9+, ensure that the Device Services server meets Apple's security requirements. The SDK must communicate with the Device Services server to run. The system might block communication if the server does not comply with requirements. Search the Apple Developer site for current application transport security requirements: https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html#//apple_ref/doc/uid/TP40016198-SW1.

SDK and Application Profiles

The SDK associates with two types of Workspace ONE UEM profiles. These two types are SDK Profiles and application profiles. You assign both types of profiles to the application from the Workspace ONE UEM console. These profiles are different from Workspace ONE UEM Device Profiles.

- **SDK Profiles** – Used to deliver security policies and settings down to the SDK embedded application. Upon receiving an SDK profile, the SDK automatically stores the most recent profile settings in memory.
- **Application Profiles** – Used to deliver certificates from an upload or a certificate authority down to an application. Consider using the challenge handler and integrated authentication instead of application profiles.

Polling for Commands and Profile Updates

The SDK checks for new commands from Workspace ONE UEM when the app is active in the background. Examples of commands are send logs, update SDK profile, and lock application. However, there may be times when your app wants

to check for new commands while active in the foreground. You can do so using the **AWCommandManager** class with the **loadCommands** method.

```
// Receive commands.
[[AWCommandManager sharedManager] loadCommands];
```

Implementing the Beacon

You can set up the Beacon to send device information to the Workspace ONE UEM console by specifying a time interval. Generic device information such as the device name, OS version, and compromised status is sampled. In addition, the Beacon module is used to start location services by specifying a location mode.

Configuration of Location

To take advantage of the location functionality of the Beacon, the host application registers itself as needing location updates in the background.

In the info.plist file, set the **UIBackgroundModes** array with a value configured as **location**.

For information on the location functionality of the Beacon, refer to the **Declaring Your App's Supported Background Tasks** section in the **App Programming Guide for iOS** at https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html#//apple_ref/doc/uid/TP40007072-CH4-SW1.

Sample Code

```
         // Initialize Beacon.   Modify the values as needed.
AWBeacon *_beacon = [[AWBeacon alloc] initWithAPNSToken:nil
        transmitInterval:300
                 locationGroup:nil
                         locationMode:AWLocationModeDisabled
                                 distance:kCLDistanceFilterNone];

        // Starts the beacon to send periodically a ping to the server.
        [_beacon start];

        // Force to send a ping right now.
        [_beacon send];
```

- **initWithAPNSToken** – Determines if your application uses APNS tokens and sends tokens to the Workspace ONE UEM console. You can send this value as nil.
- **transmitInterval** – Represents the frequency in which the Beacon checks in with the Workspace ONE UEM console (in seconds).

- **locationGroup** – Corresponds to the organization group. If your application uses authentication, it prompts users to log in. You can send this value as nil.
- **locationMode** – Uses location services for the Beacon and includes the coordinates when reporting back to the server. This method also sets up the Beacon to run in the background.
  - **AWLocationModeDisabled** – Specifies no location mode.
  - **AWLocationModeStandard** – Captures data using the GPS (on only GPS-enabled devices), which can consume battery power when enabled.

    Important: For GPS sampling to function, ensure your application supports location tracking. For more information, see Apple's documentation at https://developer.apple.com/documentation/corelocation.

  - **AWLocationModeSignificant** – Uses the significant location services from iOS and provides updates only when the device location changes at a significant level. Consider using this mode if you want to use location services.
- **distance** – Determines if you are using the standard location servers and sets the threshold, in meters, of when to generate a location service notification.

Starting and Stopping the Beacon

Once you create an instance of the Beacon with the appropriate configuration settings, you can start it or stop it at any time. Start the Beacon after initializing the SDK, and leave it running permanently.

```
[_beacon start]; // Starts sending information to the AW Console

[_beacon stop]; // Stops sending information to the AW Console
```

Manually Sending a Beacon Message

Instead of waiting for the next interval to send a Beacon to the server, you can explicitly invoke the send command and a packet will be sent to the server.

```
// Force to send a ping right now.
    [_beacon send];
```

Implementing MDM Status

The MDM Status module allows an application to check certain properties and the status of some MDM properties for the device that the application lives in.

This information is useful because you cannot obtain it directly from the SDK. You can use the data to improve security and usability at the application level. For example, a developer may want to check that another application is installed on the same device before exposing or hiding certain features in the application.

- **Device Status** – Indicates the following statuses:
  - Managed Status – Indicates if the device is Enrolled in the Console or not.
  - Compliance Status – Indicates if the device is conforming to all the compliance rules.
- **Requery Method** – Queries the Console to send to the containing device a Query command to collect certain types of device information.
- **Compliance Policies** – Retrieves a list of policies and lists details about each policy.
- **Application List** – Retrieves the list of applications that are available on the same device.

See the sample application for code examples.

Chapter 3:
Test Apps With the AirWatch SDK for iOS

The AirWatch SDK allows you to configure apps dynamically using default settings and application profiles that you can define in the Workspace ONE UEM console and associate to your applications.

- **Default Settings for SDK Profiles** – Provides settings that you can potentially use across multiple Workspace ONE UEM applications, for example branding schemes, or the type of authentication you want to use on all your applications.
- **Application Profiles** – Configures an individual application. For example, when you deploy a certificate for use by a particular application.

You can design your application to use as many or as few predefined settings in profiles, to create a flexible application based on your goals.

Test the over-the-air configuration of your app using application and SDK profiles outlined in the following sections.

Test the Integration and Functions of Applications

It is important to test the integration of your application with the AirWatch SDK , including the delivery of profiles from the Workspace ONE UEM console to your application.

Initialize the SDK in your application to set communication with the Workspace ONE UEM server and test the application.

1. Enroll your test device.

   Enroll devices to the Workspace ONE UEM console to enable communication between them.

   The SDK does not currently support testing in a simulator.

2. Upload the SDK-built app or a placeholder application that has the same bundle ID as the testing application.

   Create an empty application with the bundle ID of the testing-application to identify the application. Upload the empty application to the console and assign a default or custom SDK profile to it.

3. Assign an SDK profile to the application.

   If you do not assign a profile, the SDK does not initialize correctly.

   This step enables the console to send commands to the application with the record.

4. Push the application to test devices.

You save the application and assign it using the flexible deployment feature. Flexible deployment rules push the application to test devices with the app catalog. Use devices for testing that are Workspace ONE UEM managed devices.

You do not have to repush the application every time you make a change.

5. Run your application in Xcode.

   Run your application in Xcode. The console pushes the initialization data to the application when the application installs on test devices. After the application initializes, you can run the application as many times as you want to debug it.

Deliver Profiles to SDK Applications

The AirWatch SDK allows developers to configure their applications dynamically using SDK and application profiles.

- **SDK Profiles** provide common settings for multiple applications. For example branding schemes, or the type of authentication you want to use on all your applications.
- **Application Profiles** deploy certificates for your SDK applications.

The AirWatch SDK allows your application to integrate with the Workspace ONE UEM console to retrieve the most current settings that apply to the application. You can update settings periodically when required without a single change to the source code of the application.

Every time you save an application profile or an SDK profile, a new command to install the profiles goes to the Command Queue. The install command works on all applications and devices that are associated with the saved profiles. From the testing perspective, change what you want to test in the profile.

This change generates a new command that you can retrieve from the test code. The SDK looks for new commands when you background and foreground the application. You can also manually load commands with the **AWCommandManager** method.

Chapter 4:
MAM Functionality with Settings and Policies and the AirWatch SDK

The Settings and Policies section of the Workspace ONE UEM console contains settings that can control security, behaviors, and the data retrieval of specific applications. The settings are sometimes called SDK settings because they run on the AirWatch SDK framework.

You can apply these SDK features to applications built with the AirWatch SDK, to supported Workspace ONE UEM applications, and to applications wrapped by the AirWatch App Wrapping engine. Same features can be applied in both the places as the AirWatch SDK framework processes the functionality.

Types of Options for SDK Settings

Workspace ONE UEM has two types of the SDK settings, default and custom. To choose the type of SDK setting, determine the scope of deployment.

- Default settings work well across organization groups, applying to large numbers of devices.
- Custom settings work with individual devices or for small numbers of devices with applications that require special mobile application management (MAM) features.

Default Settings

Find the default settings in **Groups & Settings > All Settings > Apps > Settings and Policies** and then select **Security Policies**, **Settings**, or **SDK App Compliance**. You can apply these options across all the Workspace ONE UEM applications in an organization group. Shared options are easier to manage and configure because they are in a single location.

View the matrices for information on which default settings apply to specific Workspace ONE UEM applications or the AirWatch SDK and app wrapping.

Custom Settings

Find the custom settings in **Groups & Settings > All Settings > Apps > Settings and Policies > Profiles**. Custom settings for profiles offer granular control for specific applications and the ability to override default settings. However, they also require separate input and maintenance.

Assign the Default or Custom Profile

To apply Workspace ONE UEM features built with the AirWatch SDK, you must apply the applicable default or custom profile to an application. Apply the profile when you upload or edit the application to the Workspace ONE UEM console.

1. Navigate to **Apps & Books > Applications > Native > Internal** or **Public**.
2. Add or edit an application.
3. Select a profile on the **SDK** tab:
   - **Default Settings Profile**
     - For Android applications, select the **Android Default Settings @ <Organization Group>**.
     - For Apple iOS applications, select the **iOS Default Settings @ <Organization Group>**.
   - **Custom Settings Profile** – For Android and Apple iOS applications, select the applicable legacy or custom profile.
4. Make other configurations and then save the application and create assignments for its deployment.

Changes to Default and Custom Profiles

When you make changes to the default or custom profile, Workspace ONE UEM applies these edits when you select **Save**.

Changes can take a few minutes to push to end-user devices. Users can close and restart Workspace ONE UEM applications to receive updated settings.

Authentication

The AirWatch SDK provides helper classes to authenticate credentials against Workspace ONE UEM. An application can limit its access to users by integrating user authentication. Users authenticate to the Workspace ONE UEM console, whether it is a basic enrollment user or an Active Directory account. Authentication allows your application to follow enforced corporate security policies.

Configure the type of authentication the SDK profile uses to communicate with the application.

Single Sign-On and the SDK

To use single sign-on (SSO), an SDK-enabled app must interact with the AirWatch Agent for handling authentication across multiple apps. After initialization, the app can establish an SSO session with the Agent. It can delegate the handling of user authentication and SSO management to the Agent or Container. After a session is established in one app, all the other apps can share the session. Applications do not require authentication or passcodes due to this sharing behavior.

The SSO functionality can also allow the application access to the Workspace ONE UEM enrollment credentials for that device if necessary. When the SSO session expires, access to any SSO app requires the user to enter a passcode (depending on the authentication security policies set) and reinitialize the SSO session. The default settings or SDK profile also defines the maximum number of failed attempts. If the user exceeds this number, the session expires and the wipe delegate method invokes in the associated applications to signal the developer to remove local app data.

To implement the SDK, you must implement the code to initialize the SDK using **AWController** and calling **Start** from the **clientInstance**. Also, implement the lock, unlock, and wipe methods with the other delegate methods of **AWSDKDelegate** inside your app delegate.

After you upload the SDK application to the console and assign the a custom or default profile with SSO enabled, the SDK handles communication with the AirWatch Agent to manage the sessions.

Once the app has finished the communication workflow with the AirWatch Agent, the app can use the credentials.

Important: To get credentials, devices must enroll using the AirWatch Agent or Container. Otherwise, the properties are nil.

For information on using Touch ID with the SDK, see the following Workspace ONE UEM Knowledge Base article: https://support.workspaceone.com/articles/115001676428.

Implementation in Xcode

```
AWEnrollmentAccount *account = [[AWController clientInstance] account];
NSString *username = account.username;
NSString *password = account.password;
NSString *groupID = account.identifier;
```

Active Directory Password Changes

If an Active Directory (AD) password changes and becomes out of sync with the object account of the SDK, use an API to update the SDK credentials.

```
- (void)updateUserCredentialsWithCompletion:(void(^)(BOOL success,NSError *error))completionHandler;
```

If the callback works, then find the new credentials in the SDK account object.

Behavior

- SSO disabled – The system displays an authentication prompt within the SDK app for the user to enter in the new credentials.
- SSO enabled – The SDK app flips to the AirWatch Agent or Container application to update the credentials there. The AirWatch Agent v5.1+ for iOS and the Container v2.1+ support this behavior.

Authentication Type

Configure Workspace ONE UEM applications, applications built using the AirWatch SDK, and app wrapped applications to allow access when users authenticate with a set process. Select an authentication type depending on the credentials desired for access; users can set their own or use their Workspace ONE UEM credentials.

Select an authentication type that meets the security needs of your network. The passcode gives device users flexibility while user name and password offers compatibility with the Workspace ONE UEM system. If security is not an issue, then you do not have to require an authentication type.

Setting	Description
Passcode	Designates a local passcode requirement for supported applications. Device users set their passcode on devices at the application level when they first access the application.
User name and Password	Requires users to authenticate to supported applications using their Workspace ONE UEM credentials. Set these credentials when you add users in the **Accounts** page of the Workspace ONE UEM console.
Disabled	Requires no authentication to access supported applications.

Authentication Type and SSO

Authentication Type and SSO can work together or alone.

- Alone – If you enable an Authentication Type (passcode or user name/password) without SSO, then users must enter a separate passcode or credentials for each individual application.
- Together – If you enable both Authentication Type and SSO, then users enter either their passcode or credentials (whichever you configure as the Authentication Type) once. They do not have to reenter them until the SSO session ends.

SSO Session and the Airwatch Agent

Once an end user authenticates with an application participating in SSO, a session establishes. The session is active until the **Authentication Timeout** defined in the SDK profile is reached or if the user manually locks the application.

When using the Agent as a "broker application" for features such as the single sign-on feature, configure the Airwatch Agent with the applicable SDK profile. If you are using the default SDK profile, ensure that the Agent is configured to use this profile. If you do not set the Agent to use the default SDK profile, then the system does not apply your configurations you configure in the Settings and Policies section.

Challenge Handler and Integrated Authentication

On the **Security Policies** page, you can set **Integrated Authentication** to **Enabled** to allow the SSO credentials or a certificate to be passed on and used for authenticating into Web sites, such as content repositories (SharePoint) or wikis.

The AirWatch SDK does not support the use of SCEP for handling certificates. Do not select SCEP options for certificate authorities for SDK implementations.

Once enabled, you must define a list of allowed sites, which are the only sites supported with Integrated Authentication.

On the application side, use the challenge handler component in the **AWController** class of the AirWatch SDK . Inside the **AWController**, use certain methods to handle an incoming authentication challenge for connections made with **NSURLConnection** and **NSURLSession**. Find the available methods in the list.

Method	Description
Objective-C -(BOOL)canHandleProtectionSpace: (NSURLProtectionSpace*)protectionSpace withError: (NSError**)error **Swift** func canHandle(_ protectionSpace: URLProtectionSpace, withError error: Error?) -> Bool	Checks that the AirWatch SDK has the means to handle this type of authentication challenge. The SDK makes several checks to determine that it can handle challenges. 1. Is the Web site challenging for authentication on the list of allowed sites in the SDK profile? 2. Is the challenge one of the supported types: • Basic • NTLM • Client certificate 3. Does the SDK have a set of credentials to respond with: • Certificate • User name and password If all three of the criteria are met, then this method returns **YES**. The SDK does not handle server trust, so your application must handle **NSURLAuthenticationMethodServerTrust**.
Objective-C -(BOOL)handleChallenge: (NSURLAuthenticationChallenge*)challenge **Swift** func handle(_ challenge: URLAuthenticationChallenge) -> Bool	Responds to the actual authentication challenge from a network call made using **NSURLConnection**. It returns **YES** or **NO** depending on if it can respond to the authentication challenge. The system calls the **canHandleProtectionSpace** method in **AWController** first to validate that the system can process the challenge.

Method	Description
Objective-C -(BOOL)handleChallengeForURLSessionChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *credential))completionHandler; **Swift** func handleChallenge(forURLSessionChallenge challenge: URLAuthenticationChallenge, completionHandler: @escaping (_ disposition: URLSession.AuthChallengeDisposition, _ credential: URLCredential) -> Void) -> Bool	Responds to the actual authentication challenge from a network call made using **NSURLSession**. This method is the same as the handleChallenge method, except the system uses this method with calls made with NSURLSession. This call involves using a completion block to handle authentication challenges.
Objective-C -(void)fetchNewCertificatesWithError:(NSError**)error **Swift** func fetchNewCertificatesWithError(_ error: Error?)	Forces the SDK to fetch a new certificate. The SDK automatically handles retrieving certificates initially during setup, after you call start in **AWController**. However, in the event you must force the SDK to fetch a new certificate, use this method. Ensure that a certificate is properly configured in the authentication and credentials payload of the SDK profile. This method resolves issues with revoked and corrupt certificates.

Integrated authentication requires several configurations to work.

- The URL of the requested Web site must match an entry in your list of **Allowed Sites**.
- The system must make the network call so that the process provides an **NSURLAuthenticationChallenge** object.
- The Web site must return a 401 status code requesting authentication with one of the listed authentication methods.
  - NSURLAuthenticationMethodBasic
  - NSURLAuthenticationMethodNTLM
  - NSURLAuthenticationMethodClientCertificate
- The challenge handler can only use the enrollment credentials of the user when attempting to authenticate with a Web site. If a Web site requires a domain to log in, for example ACME\jdoe, and users enrolled with a basic user name, like jdoe, then the authentication fails.
- For applications using WebView, use the SDK's **handleChallege** method in the URLSession's challenge handler. Display the response on a UIWebView or a WKWebView. Do not use the SDK's handleChallenge method directly inside WKWebView's challenge handler.

Content Repository Behavior

Content repositories use the saved enrollment credentials (which are encrypted and shared with all SSO apps). If the content repository requires a different password, the connecting app prompts the user for the password at the time of accessing the repository.

Sample Code

This example illustrates how to handle a challenge from a network call made through **NSURLConnection**.

Note: This example is generic, so expand upon it in your application to handle errors and fallback scenarios.

```
- (void)connection:(NSURLConnection *)connection
willSendRequestForAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge{
        NSError*error;

        if([[AWController clientInstance]
        canHandleProtectionSpace:challenge.protectionSpace
        withError:&amp;error]){

                if([[AWController clientInstance]
                handleChallenge:challenge]){

                        NSLog(@"Challenge handled successfully");

                }else{

                        NSLog(@"Challenge could not be handled");
                }

        }else{
                //SDK does not have the means to handle this
                authentication. Add your own fallback and SSL logic
                here.

        }
}
```

Offline Access

The AirWatch SDK provides a way to allow access to the application when the device is offline and not communicating with the mobile network. It also allows access to Workspace ONE UEM applications that use the SSO feature while the device is offline.

Offline Behavior

The SDK automatically parses the SDK profile and honors the offline access policy once **AWController** is started. If you enable offline access and an end-user exceeds the time allowed offline, then the SDK automatically presents a blocker view to prevent access into the application. The system calls the **AWSDKDelegate's** lock method so your application can act locally.

Detect a Change of Users in Shared Device Mode

The SDK for iOS detects a user change on a shared device during initialization (and on background-foreground if it's already initialized). Ensure your application implements the needed delegate method and performs an application data wipe appropriate for the previous user.

Behavior When User Changes

When a user change happens, the system calls the **userChanged** delegate method in AWController. The SDK prompts the user about the user change. After the user acknowledges the prompt, the SDK initializes as a new user.

Detect a User Change and Offline Access

The SDK needs to confirm a user change when the device is offline and comes back online. The SDK behavior depends on if offline access is enabled or disabled.

Offline Access Enabled

The SDK must prompt the user to flip to the anchor application and retrieve data on the last users for confirmation.

If the user did not change, then the anchor application flips back to the SDK application and continues with the existing user. If the user changed, the SDK wipes previous user data and sends the **userChanged** delegate method call back to your application.

Offline Access Disabled

When the user tries to access the application and the user is offline, then the SDK blocks access. The SDK tells the user they must be online to access the application.

Code User Change Detection When Application is Offline

In your application project, white list a new scheme called **awcontextid**.

1. Open the info.plist file for your application.
2. Add a new scheme **awcontextid** under the **LSApplicationQueriesSchemes** key.

Compliance or Compromised Protection

The AirWatch SDK provides helper class **AWCompliance** to disallow the application on compromised devices. The **AWCompliance** is a service that runs in the background to check if the device is compromised. If it identifies the application is running on a jailbroken device, it sends the notifications configured in the SDK Profile under the **Compliance** settings.

Implementation in Xcode

The following code shows how to initialize the Compliance service to start monitoring the device for a jailbroken status.

Retrieve Compliance Settings From Cached Settings

```
// Start the AWCompliance Service and register for notifications
NSError *error = nil;
if (![AWCompliance startService:&amp;error])
        NSLog(@"An error occurred: %@", error);
else {
        NSLog(@"Successfully started AWCompliance Service");
                // Add a notification for the DisplayMessage SDK Action
        [[NSNotificationCenter defaultCenter] addObserver:selfselector:@selector
(handleDisplayMessageNotif:)
        name:AWComplianceDisplayMessageNotificationobject:nil];
                // Add other notification handlers here if desired.
        }
- (void)handleDisplayMessageNotif:(NSNotification*)notif
{
id action = [[notif userInfo] objectForKey:AWComplianceUserInfoObjectKey];
if ([action isKindOfClass:[AWAction class]])
        {
        NSLog(@"%@", [(AWAction*)action value]);
        } else {
        NSLog(@"There is no action stored in the userinfo object");
        }
}
```

Manually Checking for the Compromised Status of the Device

You can check the Compromised Status of the device directly in your application, whether the device is online or offline.

Your application can use only this function if the device has run a Beacon call successfully at least once in the past.

Run the Beacon

Run the Beacon in your application to make sure you can use the Compromised Status check functionality. If this function is invoked without having run a Beacon in the past, it returns a 'not available' status code.

Checking Device Status

The sample code shows how to check the status of the device.

```
if ([[AWCompliance sharedInstance] jailBrokenStatus] == AWDeviceJailBroken) {
        NSLog(@"Device is jailbroken. Take necessary actions");
    }
```

There are three possible statuses that are returned from the method.

- **AWDeviceJailBroken:** The device is identified as compromised.
- **AWDeviceNotJailBroken:** The device is not identified as compromised.
- **AWJailBrokenStatusNotAvailable:** The status is not available because the system has not run a Beacon call successfully on the device since last enrollment.

Proxy, App Tunneling

The purpose of app tunneling is to redirect the traffic in your application through a specific gateway. To access internal resources in your organization, use the VMware Tunnel as the proxy.

Known Limitations and Other Considerations

Due to platform and other technical limitations, only network traffic made from certain network classes can tunnel. Consider the purpose of the listed classes and review their known limitations.

- **NSURLConnection** – Calls made with **NSURLConnection** tunnel. There is one exception to this behavior. If calls are made synchronously on the main thread, they do not tunnel.
- **NSURLSession** – Calls made using **NSURLSession** tunnel only on iOS 8+ devices and depending on the configuration used. Default and ephemeral configuration types tunnel. However, background configuration types do not tunnel.
- **CFNetwork** – Most calls made using **CFNetwork** tunnel. However, **CFSocketStream** do not tunnel.
- **URLs that contain .local** – Requests with URLs containing .local do not tunnel. Various Apple services on the device use this .local string pattern. The SDK does not tunnel these requests through the VMware Tunnel to avoid interfering with these services.
- **WKWebView** - Requests made with WKWebView do not tunnel so use **UIWebView**.

Activate App Tunneling

You do not need extra code to use the app tunnel. You only need the SDK initialization code in the section titled Initialize the SDK.

To activate app tunneling, make sure that this app has an SDK profile assigned to it in the Workspace ONE UEM console and that the profile has **App Tunneling** enabled with a proper proxy configuration. When you call start in **AWController**, it reads the SDK profile assigned to your application. If needed, it also starts the traffic redirection service for the application.

After you receive the **initialCheckDoneWithError** callback from the **AWSDKDelegate**, check to see if the error object is nil or not.

Content Filtering

The Forcepoint content filtering component in the SDK is used for infrastructures that use a Forcepoint proxy or content filter. No code is needed for implementing this functionality other than calling start in **AWController** with a proper content filtering configuration defined in the SDK profile.

Geofencing

Geofence settings are configured within an SDK profile. To do so, create an SDK profile or edit an existing one. A tab labeled **Geofencing** is visible on the left side of the profile editor. After you select the **Enabled** check box, enter settings to customize a geofence. A profile containing the geofence settings are obtained through an install profile command.

Restrictions, DLP

In this section of the SDK profile, you can identify what type of restriction rules you implement in your application.

Initialize the AWRestrictions Class

To monitor the MDM enrollment and allow offline mode restrictions, the application must initialize the **AWRestrictions** class.

The example code starts the monitoring of the defined SDK restrictions. The system can initialize it multiple times with no adverse effect.

```
NSError *error = nil;
[AWRestrictions startService:&error];

if (error)
{
NSLog(@"AWRestrictions startService error: %@", error);
}
```

Check Device Enrollment

Use the SDK to retrieve the enrollment status of the device. The snippet of code shows how to perform that check.

```
AWDeviceStatusConfiguration *configuration = [[AWDeviceStatusConfiguration
alloc] initWithHostName:nilendpointPath:nil deviceStatusAction:nil];

// Create the device status controller.
AWDeviceStatusController *statusController = [[AWDeviceStatusController
alloc] initWithConfiguration:configuration];

// Query AirWatch to determine if the device is enrolled.
[statusController queryDeviceEnrollmentStatus:^(BOOL enrolled, NSError
*error)
{
// Log the result of the enrollment check.
NSLog(@"This device %@ enrolled.", (enrolled == YES) ? @"is" : @"is not");
// Clean up.
[configuration release];
[statusController release];
```

```
    }];
```

Use DLP to Control the Copy and Paste of Data Out and Into Your SDK-Built Application

Control the copy and paste interaction between your SDK-built applications and non-SDK-built applications. Use the two settings **Enable Copy and Paste Out** and **Enable Copy and Paste Into**.

Behavior

- **Enable Copy and Paste Out** - When you set **Enable Copy and Paste Out** to **No**, you can only paste copied data from your SDK-built application out to other SDK-built applications.
- **Enable Copy and Paste Into** - When you set **Enable Copy and Paste Into** to **No**, you can only paste copied data from other SDK-built applications into your SDK-built application.

Initial Set Up of the Bundle and PLIST

To add this functionality, create a bundle and PLIST file, locally, and set the keys and values.

1. Create a bundle named **AWSDKDefaults**.
2. Create a PLIST named **AWSDKDefaultSettings.plist** and put it in the **AWSDKDefaults** bundle.
3. In the PLIST, create a Boolean named **AWClipboardEnabled** and set it to **YES**.

After you add the local flag, and your admin sets the default or custom SDK policies for these features in the console, the SDK enforces the restriction. It enforces it across your application's user interfaces that use cut, copy, and paste in the listed classes and subclasses.

- **UITextField**
- **UITextView**
- **UIWebView**
- **WKWebView**

Considerations and Limitations

There are specific limitations with certain UI classes.

UIWebView and WKWebView

- The SDK evaluates javascript in UIWebView and WKWebView to get the HTML of selected content. If using WKWebView, javascript must be enabled. Javascript is always enabled in UIWebView.
- You cannot copy Images in DOC and PDF files loaded in UIWebView or WKWebView due to a technical limitation.

Out of Process Classes

The SDK does not support copy-out and copy-in restrictions in views that are out of process. For example, the feature does not work in the listed views, and this list is not exhaustive.

- SFSafariViewController
- UIDocumentInteractionViewController
- QLPreviewController

Other Limitations

- Two sets of SDK-built applications that have different SSO settings (for example, one is set with SSO on and another with SSO off) cannot share the pasteboard.
- You cannot copy from an application which has no restriction (**Enable Copy and Paste Out** set to **Yes**) and paste that content into a restricted application (**Enable Copy and Paste Into** set to **No**).
- You cannot share a pasteboard between two or more sets of applications that are in different keychain groups.

  For example, AirWatch productivity applications and custom SDK-built applications cannot share the clipboard. However, multiple custom SDK-built applications from the same developer that are in the same keychain group can share the clipboard.

Use DLP to Control Links to Open in VMware Browser, VMware Boxer, or VMware Inbox

Configure applications built with the AirWatch SDK to open in the VMware Browser and to compose emails in VMware Boxer or VMware Inbox. This feature enables end users to use alternative systems other than Safari and the Mail app. To develop this feature, create a bundle in your iOS application and configure Workspace ONE UEM to enforce the behaviors in the bundle.

Configure both systems, the browser and email systems, for this feature to work. Perform the procedures in the listed order.

1. Initial Set Up of the Bundle and PLIST
2. Enable Links for Browser
3. Enable Links for Inbox
4. Contain Data to Browser and Inbox

Initial Set Up of the Bundle and PLIST

Perform these steps before you enable any links. Use this bundle and PLIST for both HTTP/HTTPS links and MAILTO links.

1. Create a bundle named `AWSDKDefaults`.
2. Create a PLIST named `AWSDKDefaultSettings.plist` and put it in the `AWSDKDefaults` bundle.

Enable Links for Browser

To enable the application to open HTTP / HTTPS links in the VMware Browser, enable a few dictionary and PLIST flags.

1. Work in the `AWSDKDefaults` bundle.
2. Create a dictionary named `AWURLSchemeConfiguration` and put it in the `AWSDKDefaultSettings.plist`.
3. Inside the `AWURLSchemeConfiguration` dictionary, create a new Boolean entry with the key name **enabled**

and set the Boolean value to **Yes**.

If you set the Boolean value to **No**, then the HTTP and HTTPS links open in Safari. If set to **Yes**, then your SDK app opens in VMware Browser.

Enable Links for Boxer or Inbox

To enable the application to open MAILTO links in Boxer or Inbox, enable a few dictionary and PLIST flags.

1. Work in the `AWSDKDefaults` bundle.
2. Create a dictionary named `AWMaitoSchemeConfiguration` and put it in the `AWSDKDefaultSettings.plist`.
3. Configure the `AWMailtoSchemeConfiguration` dictionary, create a new Boolean entry with the key name as **enabled** and set the Boolean value to **Yes**.

   If you set the Boolean value as **No**, then MAILTO links open in the native mail. If set to **Yes**, then your SDK app looks to see if you enabled data loss prevention in the SDK profile.

   - DLP Enabled – The app opens in Boxer or Inbox.
   - DLP Disabled – The app opens in the iOS Mail app.

Contain Data to Browser and Inbox

Use the data loss prevention, DLP, settings in the Workspace ONE UEM default SDK profile to enforce the application to use VMware Browser and VMware Boxer or VMware Inbox.

If you do not enable data loss prevention in the SDK policy, the application opens links in Safari and composes email in the iOS Mail app.

1. Navigate to **Groups & Settings > All Settings > Apps > Settings and Policies > Security Policies**.
2. Select **Enabled** for **Data Loss Prevention**.
3. Disable the **Enable Composing Email** check box for the MAILTO links. If you do not disable this option, the application opens from the Mail app and not from Inbox.

Limitation With MFMailComposeViewController

If you use the `MFMailComposeViewController` scheme in your MessageUI framework, this functionality is not supported. The system cannot specify how end users access your application when it is an attachment in an email. End-users access the application with the Mail app and not Inbox.

SupportInformationController

The **SupportInformationController** class allows you to query for the email address and telephone numbers for contacting enrollment support which you can display on the application UI.

Use a Blocker Screen

The AirWatch SDK for iOS blocks the application's content when a device returns to active with a blocker screen. Upon activity, the screen goes away.

To code this behavior add the following property lists and bundles.

1. Create a PLIST entitled **AWSDKDefaultSettings.plist**.
2. Create or use the **AWBlockerViewEnableKey** with a Boolean value to enable or disable the blocker screen.
3. Add the PLIST to AWSDKDefaults.bundle.

Branding

Branding can change the look of the application with minimal development, and it can delegate several user interface properties to the configuration settings in the SDK profiles.

Some branding features are branding blocker screens and authentication screens. See the **AWProfile** class to access all the branding functionality.

Dimensions for Images on iOS Devices

It is difficult to find a single image that displays perfectly on every mobile device. However, certain ratios and dimensions for the images displayed on iOS devices can work for most displays.

Find out the ratios that often work best for branding and icons when you upload images for iOS devices.

Max Constraints

- iPhone – Not exceeding a ratio of 2.88 width over height.
- iPad – Not exceeding a ratio of 4.39 width over height.

Logo Ratios

- iPhone – 1.35 width over height.
- iPad – 1.26 width over height.

Other Considerations

If the image exceeds a height of 111 points (iPhone) or 175 points (iPad), then the image scales down while maintaining the aspect ratio. Points, which are specific to Apple iOS, differ from pixels. The conversion from points to pixel depends specifically on the device. Examples include the following ratios:

- iPhone 4 – 1 point = 1 pixel.
- Retina iPads – 1 point = 2 pixels.
- iPhone 6 Plus – 1 point = 3 pixels.

Logging

The Logging module of the AirWatch SDK allows developers to instrument their applications to discover bugs or any issues when the application is deployed to users.

Code the Level

You must code the logging level and you must set this option in the Workspace ONE UEM console. This configuration ensures that your network is not burdened with unwanted logging activity.

SDK Log Types

Workspace ONE UEM displays logs for applications that report application fails and that report application-specific data. These logs integrate with the AirWatch SDK so that you can manage applications built by it.

Find logs for applications in **Apps & Books > Analytics > App Logs**.

Setting	Description
Application Logs	This type of log captures information about an application. You set the log level in the default SDK profiles section, **Groups & Settings > All Settings > Apps > Settings and Policies > Settings > Logging**. You must add code into the application to upload these logs to the Workspace ONE UEM console.
Crash Logs	This type of log captures data from an application the next time the application runs after it crashes. These logs are automatically collected and uploaded to the UEM console without the need for extra code in the SDK application.

SDK Log Levels

Workspace ONE UEM groups logging messages into categories to distinguish critical issues from normal activities.

The Workspace ONE UEM console reports the messages that match the configured logging level plus any logs with a higher critical status. For example, if you set the logging level to Warning, messages with a Warning and Error level display in the UEM console.

Level	Logging Syntax	Description
Error	AWLogError("{log message}")	Records only errors. An error displays failures in processes such as a failure to look up UIDs or an unsupported URL.
Warning	AWLogWarning("{log message}")	Records errors and warnings. A warning displays a possible issue with processes such as bad response codes and invalid token authentications.
Information	AWLogInfo("{log message}")	Records a significant amount of data for informational purposes. An information logging level displays general processes, warning, and error messages.
Debug or Verbose	AWLogVerbose("{log message}")	Records all data to help with troubleshooting. This option is not available for all functions.

Access Logs and Events for SDK Applications

Access events that are sent by the AirWatch SDK for applications.

Access SDK and Wrapped App Events

Access SDK application logs from the App Logs page.

1. Navigate to **Apps & Books** > **Applications > Logging > SDK Analytics**.
2. View events by application ID and sample time.

Access SDK App Logs

Download or delete SDK application logs from the App Logs page.

1. Navigate to **Apps & Books** > **Applications > Logging > App Logs**.
2. Find log files by **App Name** and download or delete the files from the actions menu.

Analytics

Analytics track the important events that occur within your application. The system uses these metrics to analyze use patterns and to account for how people use your app.

Analytic Types

The AirWatch SDK for iOS offers the several types of analytics.

- Event Analytics – Records and reports information about events specific to your organization that you code into the application.
- Data Usage Analytics – Records and reports information about network traffic to track telecom statistics.

Implement the DataSampler

The DataSampler module (formerly known as Interrogator) samples detailed device data and reports it back to the Workspace ONE UEM console. Device details such as analytics, call logs, GPS location, and network adapters are all sampled with the DataSampler.

Important: For GPS sampling to function, ensure your application supports location tracking. For more information, see Apple's documentation at https://developer.apple.com/documentation/corelocation.

The DataSampler samples and transmits on two different time intervals. Device samples remain on to the disk and the system removes them after transmitted. This process allows the developer to sample statistics multiple times before sending them to Workspace ONE UEM. Samples stored on the disk are useful when a device does not have network connectivity.

AWDataSampler is a singleton object. There can only be one DataSampler for each process.

Configuration

These parameters are required to set up a DataSampler.

- **sampleModules** – Names the bit mask whose flags specify which modules to use.
- **defaultSampleInterval** – Specifies the time in seconds between DataSampler samples for all modules by default.
- **defaultTransmitInterval** – Specifies the time in seconds between DataSampler transmissions for all modules by default.
- **traceLevel** – Determines the error and information logging level of the DataSampler module when it is running.

```
[[AWAnalytics mAnalytics] setEnabled:YES];
    // Initialize DataSampler. Modify the values in the initialization
    AWDataSamplerConfiguration *config = [[AWDataSamplerConfiguration alloc]

        initWithSampleModules:(AWDataSamplerModuleAnalytics | AWDataSamplerModuleGPS)
```

```
        defaultSampleInterval:3600
        defaultTransmitInterval:14400
        traceLevel:Error];//This bit mask will enable analytics and GPS sampling

// Configure Data Sampler
[[AWDataSampler mDataSamplerModule] setConfig:config];

// Start the Data Sampler Service.
NSError *error;
[[AWDataSampler mDataSamplerModule] startUp:&error];
```

Modules Available for Sampling

These modules are available for sampling in the DataSampler.

- **AWDataSamplerModuleSystem**
- **AWDataSamplerModuleAnalytics**
- **AWDataSamplerModuleGPS**
- **AWDataSamplerModuleNetworkData**
- **AWDataSamplerModuleNetworkAdapter**
- **AWDataSamplerModuleWLAN2Sample**

Gather Telecom Data

Disable the **AWDataSamplerModuleNetworkData** mask if you gather telecom data using the AirWatch Agent. If you enable this mask for the SDK, then you receive duplicate data from the Agent and from the SDK.

Set Do Not Disturb

You can use the SDK to set the do-not-disturb (DND) status on the Workspace ONE UEM server. You must enable the DND policy in the Workspace ONE UEM console. You can find the policy at **Groups & Settings > All Settings > Devices & Users > General > Privacy > DO NOT DISTURB section**.

The two relevant methods are fetch**DeviceDNDStatus** and **setDeviceDNDStatus** found in the **AWDeviceDNDStatus** object. The example illustrates how to implement a toggle button for DND.

```
[AWDeviceDNDStatus fetchDeviceDNDStatus:^(BOOL responseStatus, BOOL dndStatus, NSDate *dndTime,
NSError *error){
    if(dndStatus){
        [AWDeviceDNDStatus setDeviceDNDStatus:NO completionBlock:^(BOOL responseStatus, BOOL
```

```
            dndStatus, NSDate *dndTime, NSError *error){
                NSLog(@"DND Disabled");
            }];
        }else{
            [AWDeviceDNDStatus setDeviceDNDStatus:YES completionBlock:^(BOOL responseStatus, BOOL
            dndStatus, NSDate *dndTime, NSError *error){
                NSLog(@"DND Enabled");
            }];
        }
    }];
```

Develop Event Analytics with the DataSampler

The event analytics function requires enabling analytics in the Workspace ONE UEM console and setting up the DataSampler module to report the analytics.

1. Code the events into your SDK application.
2. Set up the DataSampler module and configure it to report analytics.
3. Enable analytics in the Workspace ONE UEM console. Use this option to turn analytics on and off so that you do not use excessive bandwidth when you do not need data collected.
    a. Navigate to **Groups & Settings > All Settings > Settings & Policies > Settings > Analytics**.
    b. Enable the feature.

You must perform both of these tasks or the application does not collect data and it does not transmit the data to the Workspace ONE UEM console.

Developing Data Usage Analytics

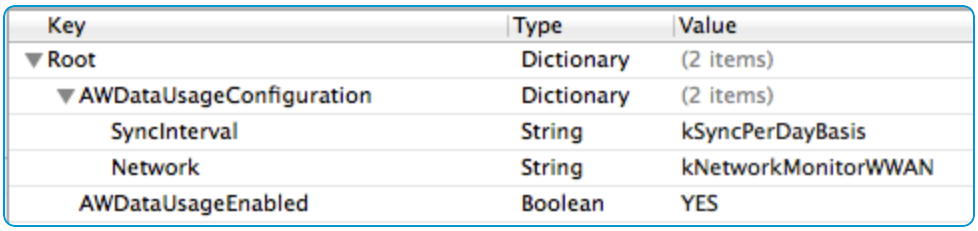
The AirWatch SDK for iOS allows your application to monitor the amount of network traffic used specifically by your application. It also reports network traffic used to the Workspace ONE UEM console for telecom tracking purposes. The SDK hooks into the common iOS networking classes to monitor the amount of data transferring in an application.

How to Enable Data Tracking in Your SDK Application

Enable the data tracking module by embedding a PLIST into your application project.

1. Create a PLIST named **AWSDKDefaultSettings.plist**, add it to your application project, and copy it to your bundle resources.
2. Define the **AWDataUsageEnabled** boolean flag in the PLIST.

3. Define the **AWDataUsageConfiguration** dictionary with **SyncInterval** and **Network** in the PLIST.



Key	Type	Value
▼ Root	Dictionary	(2 items)
▼ AWDataUsageConfiguration	Dictionary	(2 items)
SyncInterval	String	kSyncPerDayBasis
Network	String	kNetworkMonitorWWAN
AWDataUsageEnabled	Boolean	YES

AWSDKDefaultSettings.plist

The following keys are in the PLIST.

AWDataUsageConfiguration

- SyncInterval – Interval which defines how often the samples of data transmit to the Workspace ONE UEM server.
  - kSyncOnResume
  - kSyncPerDayBasis

    If data use tracking is enabled but SyncInterval is not defined, this key is the default value.
  - kSyncEveryOneHour
  - kSyncEveryTwoHours
  - kSyncEveryFourHours
  - kSyncEveryEightHours
- Network – Type of data to collect.
  - kNetworkMonitorWWAN – Track only cellular data.

    Default if network is not defined and data tracking is enabled.
  - kNetworkMonitorWIFI – Track only WIFI data.
  - kNetworkMonitorBoth – Track both WIFI and cellular data.
- AWDataUsageEnabled – Enable tracking.

Transmit to the Workspace ONE UEM Server

No additional code is required for transmitting data usage analytics to the Workspace ONE UEM server. The SDK checks on every app launch if the interval for transmitting to Workspace ONE UEM has been reached and handles the transmission accordingly.

Event analytics requires implementation of the DataSampler class to transmit.

Supported Networking Classes

SDK data usage tracking is supported for the listed iOS network classes.

- NSURLSession

  With the exception that traffic made using dataTaskWithRequest and dataTaskWithURL is not monitored on iOS 7 devices.

- NSURLConnection
- AVPlayer

Custom Settings

The AirWatch SDK allows you to define your own custom settings for your application using an SDK Profile. You can paste raw text in this section, and the SDK makes this content available inside the application using the **AWCustomPayload** object.

You can define an XML, JSON, or key-value pairs for your settings and parse the raw text in the application once it is received. However, you can use other text formats like csv or plain text.

Implementation in Xcode

The sample shows how to retrieve custom settings from the local cached settings.

```
// Get an instance of the Command Manager
AWCommandManager *commandManager = [AWCommandManager sharedManager];
// Get a pointer to the SDK Profile stored locally.
AWProfile *profile = [commandManager sdkProfile];
// profile may be nil if an AWProfile has not been received from the console
if (profile == nil)
{
   NSLog(@"There is no SDK Profile currently installed");
} else
{
   AWCustomPayload *customPayload = [profile customPayload];
   if (customPayload != nil)
   {
      NSString *customSettings = [customPayload settings];
      // Do custom processing on your settings.
      NSLog(@"%@", customSettings);
   }
}
```

Chapter 5:
Certificate Provisioning (Legacy Process)

Provisioning certificates to your app involves three steps.

- Configure the certificate authority (CA) and the CA template. Workspace ONE UEM has numerous guides outlining how to configure various certificates. See the applicable guide for information on configuring CAs and CA templates in the Workspace ONE UEM console.
- Create the app profile in the Workspace ONE UEM console.
- Assign the app profile to the application in Workspace ONE UEM prior to app deployment.

Create the Application Profile

You can create the necessary application profile after you configure the CA and certificate template settings in the Workspace ONE UEM console. This profile is deployed to the app just like an SDK profile.

1. Navigate to **Groups & Settings > All Settings > Apps > Settings And Policies > Profiles**.
2. Select **Add Profile** and then choose the **Application Profile** for iOS.
3. Fill out the General information, make sure to give the profile a name, and then choose **Credentials**.
4. Select **Defined Certificate Authority** and then choose the correct **Certificate Authority** and **Certificate Template** from the choices provided.
5. Select **Save**.

Assign the application profile to the application during the upload process outlined in the **Mobile Application Management (MAM) Guide**.

Retrieve a Certificate From an Application Profile

Retrieving certificates requires extra considerations because it involves more than handling a profile. The system stores most profile payloads locally. However, it does not store AWCertificatePayload locally. To retrieve the certificate, you must wait for notification that the system downloaded the certificate.

Important: For the SDK to check the server for a certificate, you must call the loadCommands method or the notification does not trigger.

The code shows how to retrieve a certificate payload so that your application can consume and use the certificate.

```
Register the notification observer at start up.
        [[NSNotificationCenter defaultCenter]
        addObserver:self

                selector:@selector(handleUpdatedProfile:)

                        name:AWNotificationCommandManagerInstalledNewProfile
                        object:nil];
```

Implement the **handleUpdatedProfile** that will receive the notifications when a command is processed and can extract the certificate information.

```
- (void)handleUpdatedProfile:(NSNotification
*)notification
{
// Get the profile that just got received in this call;
it could be an Application Profile, or an SDK Profile.

AWProfile *profile = (AWProfile *)notification.object;

// IMPORTANT: If expecting an application profile with a
certificate, you can ONLY obtain the certificate values
from the notification object.

// For security reasons the certificate does not get
stored locally, like all the other settings in an SDK
profile

if (profile.certificatePayload){

AWCertificatePayload *certificatePayload =
profile.certificatePayload;

if ([[certificatePayload certificateData] length] > 0 & &
[[certificatePayload certificatePassword] length] > 0)
{
```

```
        NSString *certificateName = [certificatePayload
        certificateName];

        NSData *certificateData = [certificatePayload
        certificateData];

        NSString *certificatePassword = [certificatePayload
        certificatePassword];

        // TO DO: Use the certificate here... If you don't
        consume it here, it will not be available later in
        the AWCommandManager.

        }
}
```

Chapter 6:
SDK and Application Payload Classes

This section lists all classes used to represent configuration settings from the SDK or application profiles. You can use them to access a specific setting that determines the behavior of your application.

Payload Lists

The following lists outline the payloads supported by the AirWatch SDK for iOS.

SDK Profile Payloads

- **AWAnalyticsPayload** – Represents the analytics group of an SDK profile.
- **AWAuthenticationPayload** – Represents the authentication group of an SDK profile.
- **AWBrandingPayload** – Represents the branding group of an SDK profile.
- **AWCompliancePayload** – Represents the compliance group of an SDK profile.
- **AWCustomPayload** – Represents the custom group of an SDK profile.
- **AWGeofencePayload** – Represents the geofence group of an SDK profile.
- **AWLoggingPayload** – Represents the logging group of an SDK profile.
- **AWRestrictionsPayload** – Represents the restrictions group of an SDK profile (this group was formerly known as **Access Control** in the SDK Profile).

Application Profile Payloads

- **AWCertificatePayload** – Represents the credentials group of an application profile.

Chapter 7:
Integrate With Swift Applications

Swift is a programming language developed by Apple as an alternative to C-based languages. The language strives to make writing code faster and safer with many features including supporting inferred types, automatically managing memory, and the required initialization of variables before use.

Integration Process

The high-level steps for integration include the following processes:

1. Initialize the AirWatch SDK for iOS so that it can integrate with Swift.
2. Add source code for the AirWatch SDK features you want in the app.

   See topics in this guide for available advanced app management features to code and add to the app.
3. Upload and push the app from the Workspace ONE UEM console.

   See the **Workspace ONE UEM Mobile Application Management (MAM) Guide** for instructions on deploying apps to devices using the Workspace ONE UEM console.

Initialize the SDK

To initialize the SDK, import the AirWatch SDK to your Xcode project and implement the required methods.

Note: Bridging Headers are not required after SDK 5.9.X.

Import the AirWatch SDK

Review the frameworks in your Xcode project and import the AirWatch SDK .

1. Review the following libraries, resources, and entries. See Xcode Components on page 8.
   - Check that the required libraries and bundle resources are added to your project.
   - Check that the Info.plist contains the required entries to register the callback scheme.

2. Set the **Enable Bitcode** option to **No** in your Xcode build settings.
3. Add the -ObjC flag in **Other Linker Flags** in your Xcode build settings.
4. Enter `import AWSDK` in your AppDelegate.swift to import the AirWatch SDK.

Implement Required Methods

Implement the methods from the AWSDKDelegate class into the AppDelegate.swift in Xcode.

1. Conform to the AWSDKDelegate.
2. Set the callback scheme and the delegate.

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {
            AWController.clientInstance().callbackScheme="URLScheme";
            AWController.clientInstance().delegate = self;
            return true
}
```

3. Start the AWController.

```
func applicationDidBecomeActive(application: UIApplication) {
            AWController.clientInstance().start();
}
```

4. Implement remaining delegate methods.

```
func application(application: UIApplication, openURL url: NSURL, sourceApplication: String?,
annotation: AnyObject) -> Bool {
return AWController.clientInstance().handleOpenURL(url, fromApplication:
 sourceApplication);
}
func initialCheckDoneWithError(error: NSError!){ }
func receivedProfiles(profiles: [AnyObject]!){ }
func unlock() {
}
func lock() {
}
func wipe()
}
func stopNetworkActivity() {
}
```

```
func stopNetworkActivity(networkActivityStatus: AWNetworkActivityStatus) {
}
func resumeNetworkActivity() {
}
```