VMware Workspace ONE for iOS Developer Guide

The VMware Workspace ONE® Software Development Kit for iOS (Swift), is a set of tools that incorporates functionality into custom-built, iOS applications. It enhances the security and functionality of those applications and helps save time and money.

Table of Contents

Software Version and Compatibility

This version of the Workspace ONE Software Development Kit (SDK) for iOS (Swift) is compatible with the following software.

Software	Version
Workspace ONE SDK for iOS (Swift)	24.09
Workspace ONE UEM management console	2212 or later
Apple iOS	15 or later
Apple Xcode	15 or later
Swift language	Any supported by the above Xcode versions

Developer Resources

Resources for integration of the software development kit (SDK) by application developers can be found on the VMware website, here:

https://developer.omnissa.com/ws1-sdk-uem-ios/

The resources include earlier versions of the Developer Guide documentation, other technical documentation, and the SDK itself. You will require a My Workspace One login in order to download the SDK. Speak with your Workspace ONE UEM representative for access.

Corresponding Objective-C Interfaces

The examples in this document are in Swift. See the AWController Interface file for corresponding Objective-C Interfaces if you import the Workspace ONE SDK for iOS (Swift) into an Objective-C application.

Operational Data

VMware collects a limited set of information from the Workspace ONE SDK to operate and support the SDK within third-party apps, such as notifying customers about feature removal or platform compatibility. This data is anonymized and analyzed in aggregate, and cannot be used to identify the application containing the SDK or end user. This data is sent to api.artemis.omnissa.com. Please refer to Privacy Notices online for more information about VMware data collection and privacy policies.

Set Up the SDK with Your App

Set up your application and the SDK and test the setup. Perform setup steps in order to reduce issues with integration.

Procedure

1. Initialize by adding code to import the SDK and to run the correct protocol.

2. Register a callback scheme and configure the info.plist.

3. Set AWControllerDelegate callback methods.

4. Set keychain sharing to allow applications to share a single sign on session and to share data.
   - Use keychain access groups to share data between applications in the group.
   - Enable keychain sharing for SDK-built applications that already share the same AppIdentifierPrefix and the same keychain access group.

5. Configure an AWSDKDefaultSettings.plist to customize the application with Workspace ONE SDK for iOS (Swift) features.

6. Test the integration of your application with the Workspace ONE SDK for iOS (Swift), including the delivery of profiles from the Workspace ONE UEM console to your application.

Initialize the Workspace ONE SDK for iOS (Swift Package Manager or XCFramework)

Import the SDK and define initial values so that the SDK-built app can start, connect, and communicate successful start up or start up errors.

Although integration through Swift Package Manager (SPM) is recommended, if developers want to integrate SDK through XCFramework instead, navigate to XCFramework Configuration.

Swift Package Manager Configuration (Recommended)

If switching to SPM from XCFramework previously integrated, navigate to Upgrading to Swift Package Manager.

1. Navigate to the Swift package repository and copy the link: https://github.com/vmwareairwatchsdk/iOS-WorkspaceONE-SDK.
2. Open your Xcode app project that needs to integrate the SDK, go to **File > Swift package > Add package dependency**.
3. Xcode will prompt to add a package repository url. Add the url link provided in step 1.
4. Xcode will then prompt to select version requirements: choose version requirements accordingly, and click Next.
5. Once Xcode clones the repo and resolves the dependency, it prompts to add AWSDK as binary to the project, click Finish to enter the Xcode targets general section. You should see the SDK being added under **Frameworks, Libraries, and Embedded Content**.
6. Navigate to Initialization and Implementation of SDK.

XCFramework Configuration

If you have already gone through the Swift Package Manager Configuration (Recommended), you do not need to go through this section. Please navigate to Initialization and Implementation of SDK.

If upgrading to XCFramework from SDK 21.2 or prior, navigate to Upgrading to AWSDK.xcframework

1. Unzip the Workspace ONE SDK DMG file.
2. Drag and drop the AWSDK.xcframework and select to add into your **Frameworks, Libraries, and Embedded Content**, which is on the **General** tab of your project settings.
   This action automatically adds the AWSDK.xcframework into the **Link Binary with Libraries** under **Build Phases**.
3. In Xcode, select your application's target and go to **Info -> URL Types**. Add your callback scheme. Make sure that URL scheme is unique to your application.
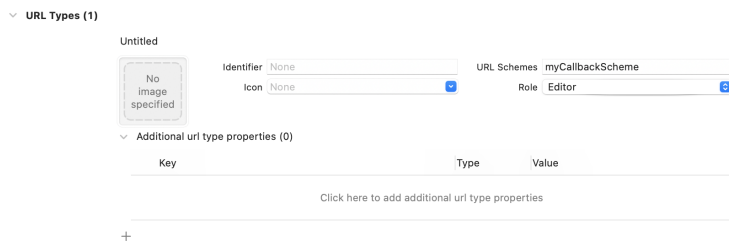
4. If Application Attestation is to be enabled, keep the Application's Team identifier handy before integrating the SDK. Please refer to Application Attestation for more details. Team identifier can be found in the membership details section in the Apple developer account.

Membership details

Entity name

Team ID

5. Import the Workspace ONE SDK module.
6. Navigate to Initialization and Implementation of SDK.

Initialization and Implementation of SDK

1. Make your **AppDelegate** conform to the **AWControllerDelegate** protocol.

```
import AWSDK

class AppDelegate: UIResponder, UIApplicationDelegate, AWControllerDelegate {
```

2. In the **AppDelegate**, add the following code to initialize and start the SDK.
Do not call the start method in **applicationWillEnterForeground** or **applicationDidBecomeActive**. These start methods result in inconsistent UI behavior.

```
func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?
) -> Bool
{

    let awcontroller = AWController.clientInstance()
    awcontroller.callbackScheme = "myCallbackScheme"
    // enable / disable the App Attestation
    awcontroller.shouldAttestApp = true
    // Should provide the team ID if App Attestation is enabled
    awcontroller.teamID = "<Application's Team Id>"
    awcontroller.delegate = self

    awcontroller.start()
    return true
}
```

3. In the **AppDelegate**, implement the listed method and code to enable the SDK to receive and handle communication from other Workspace ONE UEM applications.

```swift
func application(
    _ application: UIApplication,
    open url: URL,
    options: [UIApplicationOpenURLOptionsKey : Any] = [:]
) -> Bool {

    // `AWController.handleOpenURL` method will reconnect the SDK back to its
    // previous state to continue.
    // If you are handling application specific URL schemes. Please make sure that
    // the URL is not intended for SDK Controller.
    // An example way to perform this.

    let sourceApplication = options[
        UIApplicationOpenURLOptionsKey.sourceApplication] as? String

    let handedBySDKController = AWController.clientInstance().handleOpenURL(
        url, fromApplication: sourceApplication)

    if handedBySDKController {
        AWLogInfo("Handed over open URL to AWController")
        // SDK Controller will continue with the result from Open URL.
        return true
    }

    // Handle if this URL is for the Application.

    return false
}
```

If the application is using **SceneDelegate** to manage scenes, implement the listed method and code to enable the SDK to receive and handle communication from other Workspace ONE UEM applications

```swift
func scene(_ scene: UIScene,
        openURLContexts URLContexts: Set<UIOpenURLContext>) {

    // `AWController.handleOpenURL` method will reconnect the SDK back to its
    // previous state to continue.
    // If you are handling application specific URL schemes. Please make sure that
    // the URL is not intended for SDK Controller.
    // An example way to perform this.

    let sourceApplication = context.options.sourceApplication

    let handedBySDKController = AWController.clientInstance().handleOpenURL(
        url, fromApplication: sourceApplication)

    if handedBySDKController {
        AWLogInfo("Handed over open URL to AWController")
        // SDK Controller will continue with the result from Open URL.
        return true
    }

    // Handle if this URL is for the Application.

    return false
}
```

4. Implement the required delegate method **controllerDidFinishInitialCheck**.

```swift
func controllerDidFinishInitialCheck(error: NSError?) {
    if error != nil {
        AWLogError("Initial Check Done Error: \(error)")
        return
    }
    AWLogInfo("SDK Initial Check Done!")
}
```

Troubleshooting

In case of errors, check the following.

- `airWatchApplicationSchemeNotInAllowedLists` error code 11.

  This error will be passed to the controllerDidFinishInitialCheck delegate method if the **wsonesdk** scheme hasn't been configured. For configuration instructions, see the Configure the Info.plist section.

Upgrading to Swift Package Manager

21.9 SDK introduces Swift Package Manager as a way to distribute Workspace ONE mobile SDK. If upgrading from a prior version that uses XCFramework, removal of existing AWSDK.framework is required before integration.

1. From the **Frameworks, Libraries, and Embedded Content** remove AWSDK.xcframework or AWSDK.framework and AWCMWrapper.framework.
2. Remove StrippingSimulatorSymbols.sh under **Build Phases** if found.
3. Clean your iOS Application build folder.
4. Begin integration with Workspace ONE mobile SDK Swift Package Manager Configuration (Recommended).

Upgrading to AWSDK.xcframework

21.3 SDK introduces XCFramework distribution of Workspace ONE mobile SDK. If upgrading from an older version of SDK, removal of existing AWCMWrapper.framework and AWSDK.framework is required before integration.

1. From the **Frameworks, Libraries, and Embedded Content** remove AWCMWrapper.framework and AWSDK.framework.
2. Remove StrippingSimulatorSymbols.sh under **Build Phases**.
3. Clean your iOS Application build folder.
4. Begin integration with Workspace ONE mobile SDK XCFramework Configuration.

Configure the Info.plist

Register a callback scheme for the Workspace ONE SDK for iOS (Swift) and configure the info.plist file to receive a callback from the Workspace ONE Intelligent Hub for iOS or Workspace ONE.

If your application uses QR scans and Face ID, add corresponding parameters (NSCameraUsageDescription and NSFaceIDUsageDescription) and permissions to the info.plist file.

Prerequisites

Initialize the Workspace ONE SDK for iOS (Swift).

Procedure

1. In Xcode, navigate to Supporting Files.

2. Select the Info.plist file for the app.

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 URL Types section and add a URL Schemes entry.

5. Enter the desired callback scheme in the URL Schemes text box.

6. Add all Workspace ONE UEM anchor application schemes to the **LSApplicationQueriesSchemes** entry.
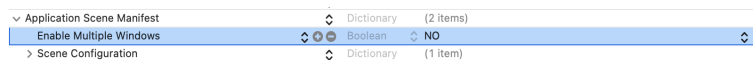
Item number	Type	Value
Item 0	String	airwatch
Item 1	String	AWSSOBroker2
Item 2	String	awws1enroll
Item 3	String	wsonesdk

⌄ LSApplicationQueriesSchemes	Array	(4 items)
Item 0	String	wsonesdk
Item 1	String	awws1enroll
Item 2	String	AWSSOBroker2
Item 3	String	airwatch

Screen Capture: Configuration in the application project plist

7. If this application scans QR codes with the device camera, add permissions for NSCameraUsageDescription. Provide a description for the application to prompt users to scan with QR codes.

8. If this application uses Biometric Authentication (FaceID / Touch ID), add permissions for NSFaceIDUsageDescription. Provide a description for the application to prompt users to turn on Face ID. 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. The system doesn't require a comparable usage description for Touch ID.

9. If the application is using SceneDelegate, disable multiple windows in the plist.

⌄ Application Scene Manifest	Dictionary	(2 items)
Enable Multiple Windows	Boolean	NO
> Scene Configuration	Dictionary	(1 item)

Screen Capture: Configuration in the application project plist.

Required and Optional AWController Delegate Callback Methods

Ensure that you added the required initial-check method during initialization and use optional delegate callback methods that are part of the AWController.

Required AWController Delegate Methods

`controllerDidFinishInitialCheck(error: NSError?)`
Called once the SDK finishes its setup.

Optional AWController Delegate Methods

`controllerDidReceive(profiles: [Profile])`
Called when the configurations profiles are received from the management console. The AWController instance or delegate can now access the configuration profiles.

`controllerDidWipeCurrentUserData()`
Called when the SDK has wiped all of its data. The application wipes any of its application specific data.

`controllerDidLockDataAccess()`
Called when the SDK has locked, user will need to unlock with username/password, passcode, touch-id in order to access application.

`controllerDidUnlockDataAccess()`
Called when the SDK has been unlocked by some form of acceptable authentication (username/password, passcode, touch-id).

`applicationShouldStopNetworkActivity(reason: AWSDK.NetworkActivityStatus)`
Called to alert the application to stop its network activity due to some restriction set by the admin's policies such as cellular data connection disabled while roaming, if airplane mode is switched on, SSID does not match what is on console, proxy failed, etc.

`applicationCanResumeNetworkActivity()`
Called to alert the application to resume its network activity because it is now fine to do so based on the device's current connectivity status and policies set by administrator.

`controllerDidDetectUserChange()`
Called when the currently logged in user has changed to alert the application of the change.

`controllerDidReceive(enrollmentStatus: AWSDK.EnrollmentStatus)`
Called when the SDK has received the enrollment status of this device from console. The application can now query the SDK for the enrollment status using the `DeviceInformationController` class after this point or use the `enrollmentStatus` parameter given in this delegate call.

`controllerWillStartRefresh()`
Called when the SDK refresh is going to take place. After this application must ensure AWControllerDelegate receives the controllerDidFinishInitialCheck(error: NSError?) callback with no errors before they call any other SDK methods.

Keychain Access Group Entitlements

Decide whether to enable or disable keychain sharing depending on what behavior you want to use in the app. If you enable sharing, use the correct format so the system signs the app with the entitlement and so apps can share data.

Enable Keychain Sharing for SDK-Built Applications

Enable keychain sharing for SDK-built applications that already share the same 'AppIdentifierPrefix' and the same keychain access group so these apps can share data.

Procedure

1. In Xcode, select your application's target and go to **Capabilities**.
2. Go to **Keychain Sharing** and turn it on.
3. Select the plus icon (+) and name the group as `awsdk`.
4. Drag the new access group to the top of the **Keychain Groups** list.

The following screen capture shows an example configuration.
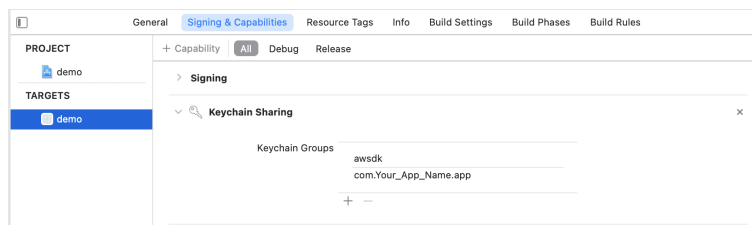
Screen Capture: Configuration in the application project

Things to consider while setting up **Keychain Groups**

1. Do not delete `awsdk` keychain group.
2. Do not change order of keychain group, `awsdk` should be on top in the keychain group list.
3. Do not use `awsdk` keychain group in your app, It should be exclusively used by SDK.

Format of Entitlements

Once keychain sharing entitlement has been enabled and keychain access groups are added to the list of groups (see figure shown above), the keychain sharing access group that application can specify in the keychain query to store the entry in keychain is of the form (AppIdentifierPrefix). (KeychainAccessGroupName). The table below shows examples. If the apps do not specify any keychain access group in the keychain query, the topmost item in the list is picked.

For information on keychain items and sharing on the Apple Developer site article Sharing Access to Keychain Items Among a Collection of Apps as of December of 2018.

Keychain sharing enabled	Keychain Access Group
Yes	• FZJQX8D5U8.awsdk • FZJQX8D5U8.com.yourKeychainGroupName
No	(System uses Bundle ID as the keychain access group for keychain entry) FZJQX8D5U8.com.YourCompany.demo

Tips to Troubleshoot Keychain Enablement

Keychain sharing does not work if:

1. keychain sharing entitlement is not enabled

2. Keychain sharing entitlement is enabled but Apps do not have the same AppIdentifier prefix

Disabled Keychain Sharing

If keychain sharing is not enabled, the WS1 SDK App will not be able to share information among other apps build with using WS1 SDK and each app will form its own cluster. This means each app will have a different authentication sessions (different passcodes if applicable). Apps will also not be able share SSO and SDK related information with other apps built with WS1 SDK.

Different AppIdentifierPrefix

Problem - Applications in a keychain access group cannot share passcodes or data if they have different prefixes. The system treats the different prefixes as separate clusters.

Solution - Edit the prefixes for applicable applications on the Apple Developer portal. However, before you change prefixes, ensure you do not need the data stored with the older prefix. This older data is lost when the prefix changes. Different Keychain Access Groups

Problem - Applications with the same prefix cannot share passcodes or data if they are in different keychain access groups. The system treats the different groups as separate clusters.

Solution - Ensure that the applicable keychain access groups have enabled keychain sharing. Merging applications from different groups that use the same account and service names can result in data collisions. Check for the listed situations to prevent collisions.

- The kSecAttrAccessGroup attribute is one of the required attribute that can uniquely identify the item stored or retrieved from the keychain.
- All other attributes, for example kSecAttrAccount and kSecAttrService, that uniquely identify the item stored and retrieved are the same.
- The kSecAttrAccessGroup attribute is not specified in the actual query to store and retrieve from the keychain.

More Information

See Apple documentation for more information on entitlements and keychains at the listed sites (as of March 2018).

- Technical Note TN2415 Entitlements Troubleshooting
- Keychain Services guide

Cluster Session Management and Reduced Flip Behavior for SSO

An application built with Swift that uses the SDK does only flips to retrieve account information. It does not flip to the anchor application to retrieve data, like environment information, and to lock and unlock operations.

In the Workspace ONE SDK for iOS (Objective-C), applications needed to flip to the anchor application to retrieve environment information, account details, and to perform all lock and unlock operations.

Cluster Session Management Explanation

The Workspace ONE SDK for iOS (Swift) includes a mechanism that uses the shared keychain for SDK apps to communicate with other SDK apps on the device. This approach provides benefits from both security and user experience perspectives.

SDK applications built by the same developer account and that are also in the same keychain group or "cluster" can now share an app passcode and an SSO session without requiring a flip to the Workspace ONE Intelligent Hub, Container, or Workspace ONE every time authentication is required.

However, applications on the same device built by different keychain groups cannot take advantage of this passcode sharing capability. There are some scenarios that still require a flip to the Workspace ONE Intelligent Hub or anchor app to obtain the server URL and other setup information. This particular flip should only occur once per cluster of applications.

SDK settings property list and bundle

The SDK settings property list is used to activate and deactivate Workspace ONE features in your application. You create the property list in an iOS bundle in your application.

Create the SDK settings property list and bundle

Proceed as follows.

1. Open your app project in Xcode.

2. Create a Settings Bundle named AWSDKDefaults.bundle.
   AWSDKDefaults.bundle can be created either in the application or it can be included in a framework / package that is consumed by the application.

   One way to do this is as follows.

   1. In the Xcode navigator, select the project.

      The project will be at the top of the navigator.

   2. In the Xcode menu, select File, New, File…

      This opens a template selection interaction.

   3. Select the template: Settings Bundle.

      The template can appear in the Resource category. You can also search for it by filtering, for example for "bundle".

      Select the plain Settings Bundle template, not the WatchKit Settings Bundle template.

   4. Click Next. A file save dialog will open.

   5. Enter the name `AWSDKDefaults.bundle` and select to save in the root of your project.

   The new bundle will be added to the resources that are copied into the app. You can check this in the target build phases, in the Copy Bundle Resources list. If it doesn't appear there, add it now by clicking the plus.

3. Create a property list named AWSDKDefaultSettings.plist in the bundle.

   One way to do this is as follows.

   1. In the Xcode navigator, select the project.

      The project will be at the top of the navigator.

   2. In the Xcode menu, select File, New, File…

      This opens a template selection interaction.

   3. Select the template: Property List.

      The template can appear in the Resource category. You can also search for it by filtering, for example for "property".

   4. Click Next. A file save dialog will open.

   5. Enter the name `AWSDKDefaultSettings.plist` and select to save in the `AWSDKDefaults.bundle` group.

   The new property list file will be added to the bundle.

   If you save the property list somewhere else by mistake, you can drag and drop it into the bundle to fix it.

Add the required entries to the property list

Some features of Workspace ONE for iOS are activated or configured by entries in the AWSDKDefaultSettings.plist file. See the following sections in this document for more information.

- SDK Payloads Reference, Code and Console for a list of features. Note that some features also require configuration in the Workspace ONE UEM management console.

- Branding for how to configure the images and colors in the SDK user interface.

Test the SDK-Built App

Test the integration of your application with the Workspace ONE SDK for iOS (Swift), 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 to test the application.

Procedure

1. Enroll your test 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.
   1. Create an empty application with the bundle ID of the testing-application to identify the application.
   2. 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. Save the application and assign it using the flexible deployment feature.
   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. Flexible deployment rules push the application to test devices with the app catalog.

5. Run your application in Xcode.

Results

The console pushes the initialization data to the application when the application installs on test devices.

What to do next

After the application initializes, you can run the application as many times as you want to debug it.

Delete Workspace ONE SDK Data

Use the `func destroyContainerData()` method in the class WS1SDKContainerCleaner to delete Workspace ONE SDK data from your Workspace ONE SDK-built app and other apps that share the iOS keychain with it.

Important: You cannot recover data deleted by this method.

- After deletion of WS1SDK data, this method will also restart WS1SDK if it was successfully initialized prior calling to this method (i.e the delegate callback func controllerDidFinishInitialCheck(error: NSError?) had been called with error as nil) OR the setup was in progress when this method was called (i.e func controllerDidFinishInitialCheck(error: NSError?) had not been called yet on the AWControllerDelegate)

- It will not restart the WS1SDK if prior calling to this method WS1SDK failed to initialze OR WS1SDK was not started.

Quit and relaunch of other SDK-build apps which share the iOS keychain with the SDK-built app is recommended to avoid undefined behavior.

Method Usage

```
let ws1SDKDataCleaner = WS1SDKContainerCleaner()
ws1SDKDataCleaner.destroyContainerData()
```

SDK Stored Certificate Information

To troubleshoot your SDK-built application, use an AWController API to find and display the Workspace ONE SDK stored certificate information. The API supports numerous certificate types and certificate attributes to query.

API Example

Note: Calling this API without waiting until the SDK calls initialcheckDone(_:) always fails with the error InvalidOperation.ContainerLocked.

```
AWController.clientInstance().retrieveStoredPublicCertificates { (certificateMap, error) in
    if let integratedAuthCert = certificateMap[CertificateUsageKey.identity].first {
        let issuer: String? = integratedAuthCert.value(
            forCertificateAttribute: CertificateInfoKey.issuer)
        let certOCSPRespondersList: [String]? = integratedAuthCert.value(
            forCertificateAttribute: CertificateInfoKey.ocspResponderList)

        // ...

    }

    if let magCert = certificateMap[CertificateUsageKey.magSigning].first {
        let validFrom: Date? = magCert.value(
            forCertificateAttribute: CertificateInfoKey.startDate)
        let validUntil: Date? = magCert.value(
            forCertificateAttribute: CertificateInfoKey.endDate)

        // ...
    }
}
```

Supported Certificate Types

```
@objc(AWCertificateUsageKey)
public class CertificateUsageKey: NSObject {

    /// Certificate of Usage key to reflect Integrated Authentication
    public static let integratedAuthIdentity: String

    /// Certificate of Usage key to reflect Integrated Authentication
    public static let uncategorizedIdentity: String

    /// Certificate of this usage are used for signing requests for MAG Proxy
    public static let magSigning: String

    /// Certificate of this usage are used for signing requests for Tunnel Proxy
    public static let tunnelSigning: String

    ///Certificates of type SSL
    public static let selfSignedSSLCerts: String

    ///Certificates of type Custom Anchors
    public static let customTrustedAnchorCerts: String

    /// SDK doesn't have specific usage for this type of certificates
    public static let others: String
}
```

Supported Certificate Attributes to Query

```
///
/// Use these strings as keys for retrieving attributes and raw data of certificates
/// from AWController.storedCertificates() API
@objc(AWCertificateInfoKey)
public class CertificateInfoKey: NSObject {
    /// Raw Certificate data in DER format
    public static let rawCertificate: String = "exportCertificateData"
    /// Return type of value - String?
    public static let subjectName: String = "subjectName"
    /// Return type of value - String?
    public static let subjectUserID: String = "subjectUserID"
    /// Return type of value - String?
    public static let subjectIdentifier: String = "subjectIdentifier"
    /// Return type of value - String?
    public static let emailAddress: String = "emailAddress"
    /// Return type of value - Data?
    public static let serialNumber: String = "serialNumber"
    /// Return type of value - String?
    public static let commonName: String = "commonName"
    /// Return type of value - String?
    public static let issuer: String = "issuer"
    /// Return type of value - String?
    public static let algorithm: String = "algorithm"
    /// Return type of value - Date?
    public static let startDate: String = "startDate"
    /// Return type of value - Date?
    public static let endDate: String = "endDate"
    /// Return type of value - String?
    public static let subjectAlternativeName: String = "subjectAlternativeName"
    /// Return type of value - String?
    public static let keyUsage: String = "keyUsage"
    /// Return type of value - String?
    public static let extendedKeyUsage: String = "extendedKeyUsage"
    /// Return type of value - String?
    public static let universalPrincipalName: String = "universalPrincipalName"
    /// Return type of value - [String]?
    public static let ocspResponderList: String = "ocspResponderList"

}
```

API to Retrieve Identity Certificates

The Workspace ONE SDK for iOS (Swift) provides an API to retrieve all stored identities fetched from the Workspace ONE UEM console so that SDK-built apps can access resources secured with certificates.

The admin configures trusted certificates as **Credentials** in the SDK profile. When the SDK fetches the SDK profile, it also fetches and stores the CA certificates.

API to Retrieve Identity Certificates

```
exportIdentityCertificates(completion: @escaping IdentityCertficatesCompletionHandler)
```

Discussion

Use this API to retrieve all SDK stored identity certificates along with passwords. Call this API after the SDK initialises to get the latest set of stored certificates. All valid PKCS #12 along with their passwords are returned. This API ensures the returned certificates are valid at the time of call.

The completion handler is not called on Main Thread.

Parameter Explanations

The completion handler takes the listed parameters.

- The parameter completion is a block to execute after certificate retrieval completes.

```
((
    _ pkcs12CertificateMap: [String: [PKCS12Certificate]]?, _ error: NSError?
) -> Void)
```

- The pkcs12CertificateMap parameter is a dictionary with an array of PKCS stored in the SDK.

- Expect the map with keys from CertificateUsageKey.integratedAuthIdentity and CertificateUsageKey.uncategorizedIdentity. The map is empty in case there is no stored certificate in the SDK.

- The parameter error is an error object that returns one of two values; why the SDK failed to return certificates or nil if the request was successful.

For the listed scenarios, the completion handler returns the listed maps and errors.

- If there are no certificates stored, then the handler returns an empty map and a nil error.
- If there are valid and expired certificates in storage, then the handler returns a map that contains valid certificates and a nil error.
- If an error occurs, then the map is nil and the error indicates why the SDK failed to return certificates.
- If stored certificates are valid, then the map is not empty and the hadler returns a nil error.

Protocol Example for P12/PKCS #12 Certificate Data

```
public protocol PKCS12Certificate {
    var data: Data { get }
    var importExportPassphrase : String { get }
}
```

Issues with P12 Passwords that Use Cipher 98 rc2–40-cbc

The API rotates the P12 password to a random string so that the SDK does not give the actual password to the app. If any stored P12 password use the cipher 98 rc2–40-cbc (which is not FIPS compliant), the SDK exports that P12 password to a FIPS compliant cipher and returns it with an updated password. However, if the API must export and update the password, it does not return the applicable certificate.

SDK Payloads Reference, Code and Console

Some features, also called payloads, require extra code in the application, entries in config files, and settings in the console to work. Others only require, extra code, config entries, or a console setting.

SDK Payloads

Table 1. Workspace ONE SDK for iOS (Swift) Payloads and Needed Configurations

SDK Capability	Add Code or Config Entries (Beyond AWController)	Set in the Console
Force Token For App Authentication	No	Yes Enable This setting controls how the system allows users to access SDK-built applications, either initially or through a forgot- passcode procedure. When enabled, the system forces the user to generate an application token through the Self- Service Portal (SSP) and does not allow user name and password.
Authentication	Yes	Yes • Enable • Set a type.
SSO	Yes Enable keychain sharing.	Yes Enable
Integrated authentication	Yes Use the challenge handler.	Yes • Enable • Enter allowed sites. • Set an authentication option.

App tunnel proxy	No	Yes
		• Enable
		• Select a mode.
		  ○ Configure the proxy components of the VMware Tunnel.
		  ○ If not using VMware Tunnel, ensure the integration of the selected proxy with your Workspace ONE UEM deployment.

Data loss prevention (DLP)	Yes • Set the AWSDKDefault bundle and the AWSDKDefaultSettings.plist. • To use the third-party keyboards feature, implement the shouldAllowExtensionPointIdentifier API in the UIApplicationDelegate.	Yes • Enable • Set the supported restriction.
Analytics	Yes • Set the AWDataSampler. • Set the AnalyticsHelper. • Decide to use the SDK or the Workspace ONE Intelligent Hub for telecom data.	Yes • Enable • If the setting is Do Not Disturb, set privacy.
Branding	Yes	Yes

SDK Capability	Add Code or Config Entries (Beyond AWController)	Set in the Console
	Add values to the AWSDKDefaultSettings.plist.	• Enable • Set colors. • Upload images.
Sample data and MDM information	Yes • Use the beacon. The SDK sends the beacon but you can manually send the beacon when desired. • Query the DeviceInformationController singleton class.	No
Compromised protection	No Use code to check the status of devices with the application.	Yes Enable
Dynamic Compromise Detection	No Have the app consume the supported SDK version.	No Ensure that devices can access specified URLs for rule updates.
Custom settings	Yes Use the AWCustomPayload object.	Yes • Enable • Enter code.
Geofencing	Yes Implement region monitoring. See the Apple developer website for details, for example: Monitoring the User's Proximity to Geographic Regions.	Yes • Enable • Set the area.
Logging	Yes Add APIs for logging. See the sample applications for examples.	Yes • Enable • Set the level. • Set wi-fi.
Offline access	No	Yes • Enable • Set time allowed to be offline.
Encryption	Yes Use methods in the AWController to encrypt and decrypt data.	No However, the strength of the encryption depends on the authentication method set in the Workspace ONE UEM console.
SDK App Compliance > Application Version	No Use the latest SDK frameworks.	Yes • Enable • Add the application identifier. • Select an operator. • Enter the applicable application version. The console blocks non-compliant devices.
SDK App Compliance >OS Version	No Use the latest SDK frameworks.	Yes • Enable • Select an operator.

SDK Capability	Add Code or Config Entries (Beyond AWController)	Set in the Console
		• Select the OS version. • Select an action. The console supports the block and wipe actions.
Apple Push Notifications	Yes Add methods to AppDelegate.swift.	Yes • Enable APNs in the app. • Upload the production APNs certificates.
Certificates and Credentials Payloads	Yes Use APIs to fetch certificates, authenticate, and validate the server trust.	Yes Admin configures and adds certificates to the console with an SDK profile.

Authentication Type Payload Description

Set access to your application with the authentication type paload. Use a local passcode, Workspace ONE UEM credentials, or require no authentication.

Select an authentication type in the Workspace ONE UEM console and use the provided SDK helper classes in your application.

Setting	Description
Passcode	Designates a local passcode requirement for the application. Device users set their passcode on devices at the application level when they first access the application.
Username and Password	Requires users to authenticate to the application with their Workspace ONE UEM credentials.
Disabled	Requires no authentication to access the application.

Authentication Type and SSO Setting Behaviors

You can use keychain sharing, the authentication type, and the single sign-on (SSO) option to make access to your application persistent.

Keychain Access Group Required

You must have a shared space, a keychain access group, so that applications signed in the correct format can share keychain entries. See Keychain Access Group Entitlements for information on the signing format. See Tips to Troubleshoot Keychain Enablement for common issues with keychain sharing.

Enable Authentication Type and SSO

If you enable both authentication type and SSO, then users enter either their passcode or credentials once. They do not have to reenter them until the SSO session ends.

Enable Authentication Type Without SSO

If you enable an authentication type without SSO, then users must enter a separate passcode or credentials for each individual application.

Prerequisites to Use SSO

Workspace ONE UEM allows access to iOS applications with single sign on, however. To use SSO, set console, application, and anchor application components and query the SSO status.

SSO Components

- Enable the SSO setting in the SDK default settings and policies in the Workspace ONE UEM console.
- Initialize the SDK in the AppDelegate.
- Ensure an anchor application is on devices like the Workspace ONE Intelligent Hub or Workspace ONE. The anchor application deployment is part of the Workspace ONE UEM mobile device management system.

Query the Current SSO Status

To query the SSO status of the iOS application, wait for the **controllerDidFinishInitialCheck** method to finish. Look in the **DeviceInformationController** class for the **ssoStatus** property. If the **controllerDidFinishInitialCheck** method is not finished, the SSO status returns as **SSO disabled**.

SSO Configurations and System Login Behavior for iOS Applications

Workspace ONE UEM allows access to iOS applications with single sign on enabled in two phases. Workspace ONE UEM checks the identity of the application user and then it secures access to the application.

Application Access With SSO Enabled

The authentication process to an application with Workspace ONE UEM SSO enabled includes two phases: accessing the app and securing persistent access.

1. Identify user for app access - The first phase ensures that the user's credentials are valid. The system identifies the user first by silent login. If the silent login process fails, then the system uses a configured, authentication system. Workspace ONE UEM supports username and password, token, and SAML.
2. Secure persistent app access - The second phase grants the user access to the application and keeps the session live with a recurring authentication process. Workspace ONE UEM supports passcode, username and password, and no authentication (disabled).

Authentication Behavior By SSO Configuration

The SSO configuration controls the login behavior users experience when they access applications. The authentication setting and the SSO setting affect the experience of accessing the application.

Table 1. Login Behavior for Users when Passcode is Set for SSO

Authentication Phase	SSO Enabled	SSO Disabled
Identify	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system identifies credentials against a common authentication system (username and password, token, and SAML).	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system identifies credentials against a common authentication system (username and password, token, and SAML).
Secure	• Prompt if passcode exists: The system does not prompt for the passcode if the session instance is live. • Prompt if passcode does not exist: The system prompts users to create a passcode. • Session shared: The system shares the session instance across applications configured with Workspace ONE UEM SSO enabled.	• Prompt if passcode exists: The system prompts users the application passcodes. • Prompt if passcode does not exist: The system prompts users to create a passcode. • Session not shared: The system does not share the session or the passcode with other applications.

Table 2. Login Behavior for Users when Username and Password is Set for SSO

Authentication Phase	SSO Enabled	SSO Disabled
Identify	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system identifies credentials against a common authentication system (username and password, token, and SAML).	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system prompts for application login credentials.
Secure	• Prompt: The system does not prompt for the login credentials if the session instance is live. • Session shared: The system shares the session instance across applications configured with Workspace ONE UEM SSO enabled.	• Prompt: The system prompts for the login credentials for the application on every access attempt. • Session not shared: The system does not share the session with other applications.

Table 3. Login Behavior for Users when Disabled is Set for SSO

Authentication phase	SSO enabled	SSO disabled
Identify	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system identifies credentials against a common authentication system (username and password, token, and SAML).	• Silent login: The system registers credentials with the managed token for MDM. If silent login fails, the system moves to the next identification process. • Authenticate: The system prompts for application login credentials.
Secure	Prompt: The system does not prompt users for authentication.	Prompt: The system does not prompt users for authentication.

Integrated Authentication and the Challenge Handler

Use integrated authentication to pass single sign on (SSO) credentials or certificates to authenticate to web sites like content repositories and wikis. Set the payload in the Workspace ONE UEM console and add a list of allowed sites. Then use the challenge handler in your application to handle incoming authentication challenges.

Challenge Handler Methods for Challenges

Find the challenge handler in the AWController class of the SDK. Inside the AWController, use the listed methods to handle an incoming authentication challenge for connections made with NSURLConnection and NSURLSession.

Table 1. Descriptions of Challenge Methods

Method	Description

``
func canHandle(
    _ protectionSpace: URLProtectionSpace,
    withError error: Error?
) -> Bool
``
	Checks that the Workspace ONE SDKcan 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?
	　○ 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.

``
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.

Requirements for Integrated Authentication

For integrated authentication to work, communication between the allowed sites and the challenge handler must use a 401 status code, specific authentication methods, and the correct credentials.

- 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 that requests 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.

- If your application uses an embedded web view, you can use the SDK `handleChallenge` method either in a URLSession challenge handler, or in a WKWebView challenge handler. If you use handleChallenge in a URLSession challenge handler, display the response in a UIWebView or WKWebView instance. **Note:** The `UIWebView` has been deprecated as of December 2020 by Apple.

SCEP Support to Retrieve Certificates for Integrated Authentication

The Workspace ONE SDK supports the SCEP protocol, with limitations, to retrieve certificates for integrated authentication. To use SCEP certificates for your SDK-built application, ensure integrated authentication is enabled and that SCEP is configured in the console as a certificate authority.

Note: Workspace ONE SDK requires a challenge within Certificate Signing Request(CSR) to block unauthenticated authorization requests.

Supported SAN Information Types

The SDK fully supports the listed Subject Alternative Names (SAN) information types in certificate attributes.

- dNSName
- ntPrincipalName
  Note: When you configure this information type, it displays as an entry nested under the otherName attribute. Although otherName is not supported, ntPrincipalName is supported even as a nested entry of otherName.
- rfc822Name
- uniformResourceIdentifier

Supported with Correct Format

The Workspace ONE SDK supports the listed SAN information types but you must use the correct format or the SDK ignores them.

- iPAddress
- registeredID

Not Supported

The Workspace ONE SDK does not support the listed SAN information types. If you configure them, the SCEP process fails.

- Custom
- directoryName
- ediPartyName
- otherName
- x400Address

Methods for a Pending Status from the SCEP Certificate Authority

Use the **AWController** method to modify SCEP certificate fetches to account for when the SCEP certificate authority returns a pending status for the fetch.

Pending Status of Certificate Fetches

Some configurations set the SCEP certificate authority to not issue the certificate until a request is approved. In this scenario, the authority returns a pending status to the SDK. You can use the methods in AWController to configure the retry logic and monitor the retry progress.

Ensure the Certificate Authority Server Handles Retry Requests

The Workspace ONE SDK retries the fetch request based on the parameters in the modified code or using the default behavior (retries every 5 milliseconds for 10 tries). If a certificate authority server is not configured to handle retry requests caused by the pending status, the fetch never completes.

Methods for Pending Status

Use the **AWController** to modify the retry timeout and maximum number of retry attempts when fetching SCEP certificates. Also, use the SDK delegate method to notify the SDK-built application on the progress of the pending SCEP certificate fetch.

Table 1. Pending Status Methods

Configuration	Code Examples
Modify the retry timeout and maximum number of retry attempts.	Modify the AWController. ``` public func setPendingCertificateRetry(timeout: Double, maxAttempts: Int) -> Bool ``` Here is an example of code modifications that set the timeout value to 10 seconds and the maximum number of retry attempts to 8. ``` let success = AWController.clientInstance.setPendingCertificateRetry(timeout: 10.0, maxAttempts: 8) ``` **Note:** If you do not configure the timeout and retry attempts, then the timeout value defaults to 5 milliseconds and the maximum number of retry attempts defaults to 10.

Use the delegate method for pending status notifications.	Use a delegate method to notify about the pending status of the fetch.
	
	``
	@objc(didFinishPollingForPendingCertificateIssued:error:)
	optional public func controllerDidFinishPollingForPendingCertificate(
	    certificateIssued: Bool,
	    error: NSError?
	)
	``
	
	Here is an example of the delegate method for notification.
	
	``
	func controllerDidFinishPollingForPendingCertificate(
	    certificateIssued: Bool, error: NSError?
	) {
	    // Application logic goes here
	}
	``

Table 2. Error Codes for Pending Status

| Error Code	Description
certificateIssuancePending	The certificate is pending.
retryIntervalNotReached	The timeout is not reached for retry. You can set in setPendingCertificateRetry.
maximumAllowedAttemptsEllapsed	The maximum attempts have been reached for polling. You can set it in setPendingCertificateRetry.

Using Persistent Tokens with Integrated Authentication

SDK supports the usage of Persistent Tokens to resolve authentication challenges with NSURLAuthenticationMethodClientCertificate authentication method. Applications can use the existing SDK API to handle certificate Challenge. Currently SDK supports using PIV-D application as the Persistent Token Provider. The support is from iOS 14 or later.

```
func handleChallenge( forURLSessionChallenge challenge: URLAuthenticationChallenge,
                              completionHandler: @escaping ( disposition: URLSession.AuthChallengeDisposition,
                                                             credential: URLCredential )
                )
```

Setting up the Application to utilise Persistent Tokens

1. Add "com.apple.token" in keychain sharing groups.
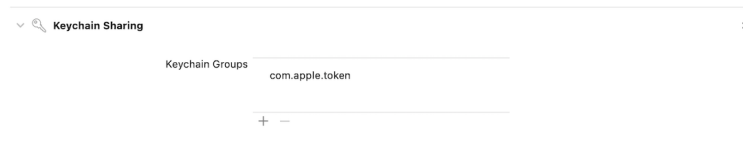
Screen Capture: Adding an entry into keychain sharin groups

2. Please Cross check the keychain access group in the Entitlements file. If it is showing $(AppIdentifierPrefix)com.apple.token as shown below, please remove $(AppIdentifierPrefix) from the keychain access group .
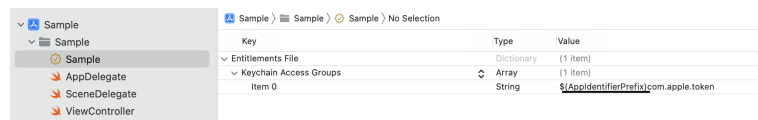
Screen Capture: Cross verification in entitlements file

3. After the correction, Entitlements file should look like this.
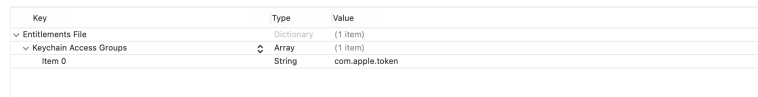
Screen Capture: Correction in entitlements file

UEM configuration to enable Persistent Token support

UEM adminstrator can enable this feature with the help of Custom Setting.

```
{
    "CustomSDKSettings": {
        "PersistentTokenConsumption": {
            "Allowed": true,
            "RestrictedToProviders": ["com.air-watch.pivd.pivdtoken"]
        }
    }
}
```

Changes to Active Directory Passwords

Use an API to update the Workspace ONE SDK for iOS (Swift) credentials when there are 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. An example for using this API is for situations where the password changed for access to sites controlled by integrated authentication configurations.

```
AWController.clientInstance().updateUserCredentials(with: { (success, error) in
    ///insert completion handler code here
})
```

Find the new credentials in the SDK account object after the callback successfully returns.

Configure VMware Tunnel for App Tunneling

The VMware Workspace ONE Tunnel provides several secure methods for individual applications that use the VMware Workspace ONE SDK to access corporate resources. Select from two options, VMware Tunnel and VMware Tunnel - Proxy.

- **VMware Tunnel**: The Workspace ONE SDK for iOS (Swift) provides app tunneling without adding code to the application. However, you need to configure app tunneling in the Workspace ONE UEM console. This option is the preferred Tunnel.

- **VMware Tunnel - Proxy**: The Tunnel-Proxy component uses HTTPS tunneling to use a single port to filter traffic through an encrypted HTTPS tunnel for connecting to internal sites such as SharePoint or a wiki. The Workspace ONE SDK for iOS (Swift) provides app tunneling without adding code to the application. However, you need to configure app tunneling in the Workspace ONE UEM console.

Note: If users access an internal resource through a non-standard port (a port that is not port 80 or 443), you must explicitly list the port number in the URL you enter in App Tunnel URLs. For example, if the resource URL is data.company.com and it is accessed through port 7777, you must add `data.company.com:7777` in the App Tunnel URLs field.

Prerequisites

You must have a valid Tunnel deployment. Access VMware Tunnel on Windows or VMware Tunnel on Linux for details.

Procedure

1. Navigate to Groups & Settings > All Settings > Settings & Policies > Security Policies > AirWatch App Tunnel.

2. Enable the setting.

3. Select an app tunnel mode, either VMware Tunnel - Proxy or VMware Tunnel.

4. In the App Tunnel URLs field, enter the URLs that you do not want to tunnel.
   - Enter no URLs and every URL goes through the VMware Tunnel.
   - Enter one or more URLs and the system splits the traffic. This configures split tunneling. The system does not send the URLs entered in this field through the VMware Tunnel. The system does send all other URLs through t he VMware Tunnel.

App Tunneling Known Limitations and Other Considerations

Due to platform and other technical limitations, only network traffic made from certain network classes can tunnel.

Table 1. Supported Network Classes

Network Class	Supported
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 except for CFSocketStream, which does not tunnel.

Table 2. Network Classes Not Supported

Network Class	Not Supported
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 supports tunnel only from iOS 17

Behavior of Copy and Paste for SDK-Built Applications

The copy and paste payloads, Enable Copy and Paste Out and Enable Copy and Paste Into, restrict actions when set to No. They allow actions when set to **Yes**.

- 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.

Limits of DLP Copy and Paste

The copy and paste payloads for the Workspace ONE SDK for iOS (Swift) are limited by parameters, out of process classes, SSO and DLP configurations, and keychain groups. There are specific limitations with certain UI classes.

- WKWebView

  - You cannot copy images in DOC and PDF files loaded in WKWebView due to a technical limitation.
  - Copy/Paste does not work in web applications like Microsoft Word/Excel etc.

- Out of Process Classes - The Workspace ONE 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, VMware Workspace ONE 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.

Set Up the Bundle and PLIST for Copy and Paste

To control the copy and paste interaction between your SDK-built applications and non-SDK-built applications, create a bundle and PLIST file, locally, and set the keys and values.

For details on creating the bundle and PLIST during initial setup, see SDK settings property list and bundle.

Procedure

1. Create a bundle named AWSDKDefaults if you did not create it during initial setup.
2. Create a PLIST named AWSDKDefaultSettings.plist and put it in the AWSDKDefaults bundle if you did not do this during initial setup.
3. In the PLIST, create a Boolean named AWClipboardEnabled and set it to YES.
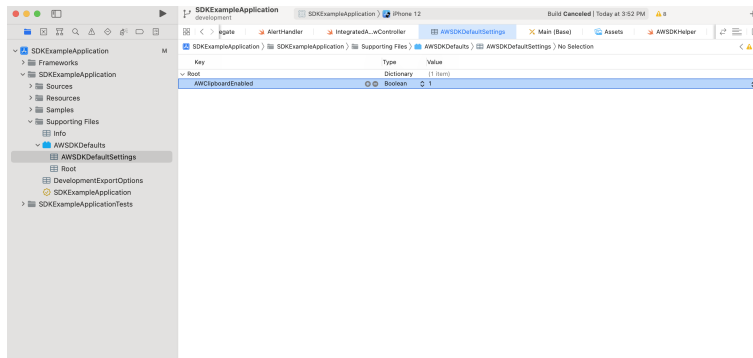
Screen Capture: Configuration for Copy and Paste in the application project

Results

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
- WKWebView

Note that the restriction isn't enforced in the following classes.

- UISearchTextField interactions won't be restricted.
- UISearchBar interactions won't be restricted.

Behavior of the Third-Party Keyboard Restriction

Run the third-party keyboard restriction by starting the AWController and configuring the data loss prevention setting in the Workspace ONE UEM console. This payload behaves depending on the most restrictive setting.

Request your Workspace ONE UEM admin to configure the data loss prevention (DLP) menu item. Find the console settings in **Groups & Settings > All Settings > Apps > Settings and Policies > Security Policies > Data Loss Prevention > Enable Third Party Keyboards**.

When this feature is set to **No**, any third party keyboards used in the application are automatically replaced with the native system keyboard.

SDK Behaves According to the Most Restrictive Implementation

If your application's code overrides the shouldAllowExtensionPointIdentifier delegate method, the Workspace ONE SDK for iOS (Swift) honors the more restrictive implementation.

For example, if the SDK setting allows third party keyboards but your application forcibly returns no to disallow custom keyboards, then custom keyboards are disallowed in the application. If the SDK setting does not allow third party keyboards then the third party keyboard is not allowed regardless of your applications implementation of the method.

Table 1. Third Party Keyboard Restriction Behavior Depends on Console Settings and Code

Data Loss Prevention Setting	Enable Third Party Keyboard Setting	Is shouldAllowExtensionPointIdentifier Implemented in the Application	Keyboard Behavior
Disabled	NA	Implemented	Third party keyboards behave depending on the implementation of the delegate method.
Enabled	Set to No.	Implementation does not matter.	Third party keyboards are not available.
Enabled	Set to Yes.	Implemented	Third party keyboards are available.
Enabled	Set to Yes.	Implemented and returns yes.	Third party keyboards are available.
Enabled	Set to Yes.	Implemented and returns no.	Third party keyboards are not available.

Run the Application to See Expected Behaviors

When the **Enable Third Party Keyboard** setting is configured in the console, the SDK does not enforce the restriction until the next time the user runs the application after the application retrieves the new SDK profile.

Use DLP to Control Links to Open in Workspace ONE Web and Workspace ONE Boxer

Configure applications built with the Workspace ONE SDK to open in the Workspace ONE Web and to compose emails in Workspace ONE Boxer. 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.

Procedure

1. 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.

2. Enable Links for Workspace ONE Boxer
   To enable the application to open MAILTO links in Workspace ONE Boxer, enable a few dictionary and PLIST flags.

   1. Work in the AWSDKDefaults bundle.
   2. Create a dictionary named **AWMailtoSchemeConfiguration** 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 Workspace ONE Boxer.
      - DLP Disabled – The app opens in the iOS Mail app.

3. Enable Links for Workspace ONE Web.
   To enable the application to open HTTP / HTTPS links in the Workspace ONE Web, 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 Workspace ONE Web.
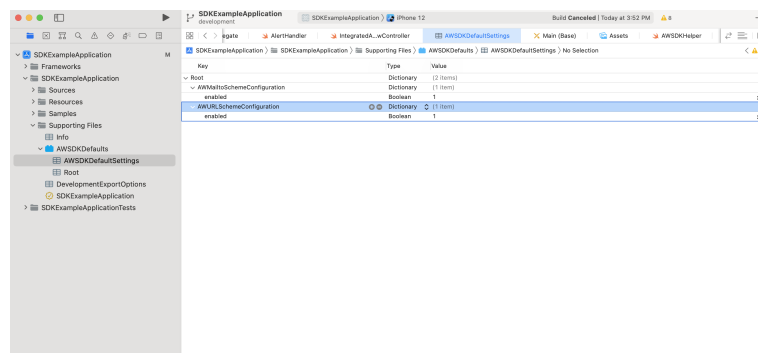
Screen Capture: Configuration of DLP to Control Links to Open in Workspace ONE app in the application project

4. Contain Data to Workspace ONE Web
   Use the data loss prevention, DLP, settings in the Workspace ONE UEM default SDK profile to enforce
   the application to use Workspace ONE Web andWorkspace ONE Boxer.
   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.

Disable the Default Blocker Screen

The Workspace ONE SDK displays a blocker screen to cover the application's content when the
application is not active. When the app is in the foreground, the Workspace ONE SDK closes the blocker
screen. You can disable this screen and use your own custom background blocker screen or use the
Workspace ONE SDK screen.

Procedure

1. Work in the AWSDKDefaults bundle.

2. Create a new Boolean entry with the key name as AWBlockerViewEnableKey in
   AWSDKDefaultSettings.plist.

   - If you set AWBlockerViewEnableKey to No, then the Workspace ONE SDK disables the blocker
     screen so that you can use your own blocker screen.
   - If you set AWBlockerViewEnableKey to Yes, then the Workspace ONE SDK uses its blocker screen.
   - If AWBlockerViewEnableKey is missing from the plist, then the Workspace ONE SDK displays its
     blocker screen.
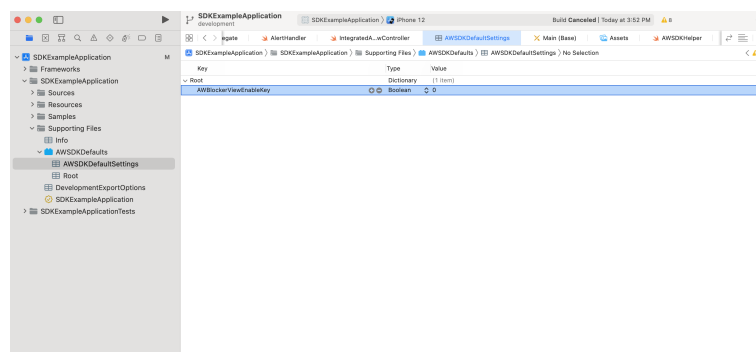
Screen Capture: Configuration of Blocker Screen in the application project

Restriction of Document Sharing

Workspace ONE data loss prevention supports the restriction of document sharing between mobile applications. Restricting document sharing is optional. If it is in use, a document file that is in one app can only be opened in another app if the other app is on a list of approved applications.

The list of approved applications is configured in the Workspace ONE Unified Endpoint Manager (UEM) console. Restriction of document sharing is imposed at run time by the Workspace ONE mobile Software Development Kit (SDK), if configured in the mobile application.

Console Configuration

The list of approved applications is configured in the management console. The configuration can be in an SDK profile, for example, and the profile can be assigned to your app. Administrator privileges in the enterprise Workspace ONE UEM console will be required to make the configuration.

The following instructions are an outline for guidance. Full documentation can be found in the online help.

1. Navigate to: Groups & Settings, All Settings, Apps, Settings and Policies, Security Policies.

   This opens the Security Policies configuration screen, on which a number of settings can be switched on and off, and configured.

2. For the Data Loss Prevention setting, select Enabled.

   When Enabled is selected, further controls will be displayed.

3. For the Limit Documents to Open Only in Approved Apps setting, select Yes.

   When Yes is selected, The Allowed Applications List control will be displayed.

4. Enter each of the approved applications in the Allowed Applications List text box.

5. Select Save to finalize the configuration.
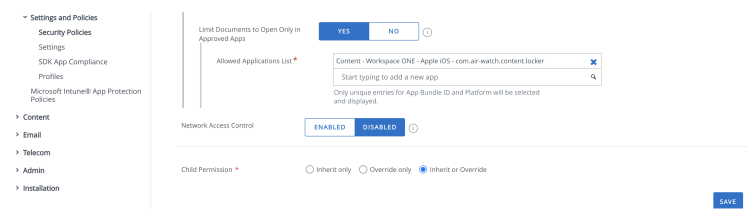
Screen Capture: Configuration for Restriction of Document Sharing in the management console

This concludes the console configuration.

Application Configuration

Integrate with document sharing restriction in your app configuration by adding property settings to the AWSDKDefaultSettings.plist file.

The following screen capture shows an example configuration.

Screen Capture: Configuration for Restriction of Document Sharing in the application project

The properties for integration are as follows.

EnableSecureDocumentController

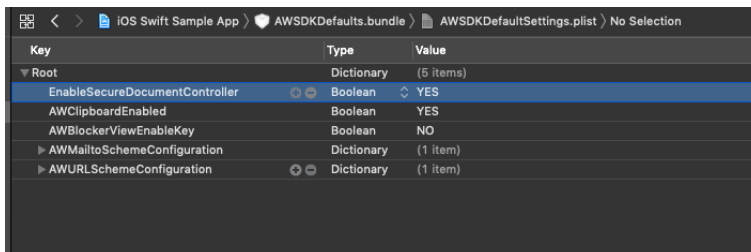The EnableSecureDocumentController property controls document sharing via the native `UIDocumentInteractionController` interface. The property takes a Boolean value.

If **EnableSecureDocumentController YES** is set then the following restrictions will be applied to a file sent to a UIDocumentInteractionController, if specified in the security policy.

- The file can be *copied to* an app that is on the approved list from the management console. In this case, copying means sending the file to the other app directly, not to an app extension. When a file is copied, the device user interface will flip to the other app.

- The file cannot be copied to an app that isn't on the approved list. Note that this applies equally to the VMWare Workspace ONE Productivity Apps suite.

- The file cannot be sent to an app extension. This applies even if the app that hosts the extension is on the approved list.

- Some *sharing actions* won't be available. Sharing actions means options, such as Copy and Print, that appear in the default document interaction user interface.

If **EnableSecureDocumentController NO** is set then the above restrictions won't be applied to a file sent to a UIDocumentInteractionController. The approved list from the management console will be ignored. This is the default.

DisableActivityViewController

The DisableActivityViewController property controls data sharing via the native `UIActivityViewController` interface. The property takes a Boolean value.

If **DisableActivityViewController YES** is set then data cannot be shared via a UIActivityViewController, if specified in the security policy. This option mustn't be used in the following cases.

- Don't set DisableActivityViewController YES if your application uses the UIDocumentInteractionController interface.

- Don't set DisableActivityViewController YES if you set EnableSecureDocumentController YES.

If **DisableActivityViewController NO** is set then data can be shared via a UIActivityViewController. This is the default.

Security Policy Application

The above application configuration directs the SDK to apply the security policy of the enterprise to parts of the native user interface. The security policy will be received at run time, for example in the SDK profile from the management console.

The policy mightn't specify data loss prevention, or might specify that document sharing isn't restricted. In that case, the SDK won't modify the behaviour of the native user interface.

Restriction of Printing

Workspace ONE data loss prevention supports the restriction of printing the documents. Restricting print is optional.

Restriction of print is imposed at run time by the Workspace ONE mobile Software Development Kit (SDK), if configured in the mobile application.

Console Configuration

Restriction of print is configured in the management console. The configuration can be in SDK profile, for example, and the profile can be assigned to your app. Administrator privileges in the enterprise Workspace ONE UEM console will be required to make the configuration.

The following instructions are an outline for guidance. Full documentation can be found in the online help.

1. Navigate to: Groups & Settings, All Settings, Apps, Settings and Policies, Security Policies.

   This opens the Security Policies configuration screen, on which a number of settings can be switched on and off, and configured.

2. For the Data Loss Prevention setting, select Enabled.

   When Enabled is selected, further controls will be displayed.

3. For restricting Print, select NO against Enable Printing

   For allowing Print, select YES against Enable Printing

4. Select Save to finalize the configuration.

Enable Location Services	YES	NO	ⓘ
Enable Printing	YES	NO	ⓘ
Enable Screenshot	YES	NO	ⓘ

Screen Capture: Configuration for Restriction of Printing in the management console

This concludes the console configuration.

Application Configuration

Integrate with Print restriction in your app configuration by adding property settings to the AWSDKDefaultSettings.plist file.

The following screen capture shows an example configuration.

EnableSecureDocumentController		Boolean	YES
DisableActivityViewController		Boolean	YES
EnableSecurePrintInteractionController	⊕ ⊖	Boolean ⇕	YES
› Branding	⊕ ⊖	Dictionary	(5 items)

Screen Capture: Configuration for Restriction of Printing in the application project

The properties for integration are as follows.

EnableSecurePrintInteractionController

The EnableSecurePrintInteractionController property controls printing via the native `UIPrintInteractionController` interface. The property takes a Boolean value.

If **EnableSecurePrintInteractionController YES** , printing would be restricted only if printing is restricted in the SDK DLP settings in UEM.

If **EnableSecurePrintInteractionController NO** , printing would not be restricted irrespective of the DLP settings in UEM. This is the default.

Set Up the DataSampler Module for Analytics

The DataSampler module samples detailed device data and reports it back to the Workspace ONE UEM console. Device details such as analytics and network adapters are all sampled with the DataSampler.

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 bitmask 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.

Modules Available for Sampling

These modules are available for sampling in the DataSampler.

- AWDataSamplerModuleSystem
- AWDataSamplerModuleAnalytics
- AWDataSamplerModuleNetworkData
- AWDataSamplerModuleNetworkAdapter
- AWDataSamplerModuleWLAN2Sample

Gather Telecom Data

Disable the AWDataSamplerModuleNetworkData mask if you gather telecom data using the Workspace ONE Intelligent Hub. If you enable this mask for the SDK, then you receive duplicate data from the Workspace ONE Intelligent Hub and from the SDK.

Use AnalyticsHelper

The AnalyticsHelper is a singleton with a property and a function. Send your custom analytics event from your application to the console with this process.

Procedure

1. Ask your admin to enable the Analytics setting in the SDK profile for the SDK-built application. This setting is in the console at Groups & Settings > All Settings > Apps > Settings and Policies > Settings > Analytics.

2. In the application, call the recordEvent method on the singleton after the controllerDidFinishInitialCheck delegate callback returns.

```
func sendAnalytics() {
    let analytics = AnalyticsHandler.sharedInstance
    analytics.recordEvent(
        AWSDK.AnalyticsEvent.customEvent,
        eventName: "EVENT_NAME",
        eventValue: "EVENT_VALUE",
        valueType: AWSDK.AnalyticsEventValueType.string
    )
}
```

Results

After the system records the event, it saves the event in the SDK container for two hours. After the two hours passes, the SDK sends analytics recorded to disk to the console the application re-starts.

What to do next

Locate the data in the console in **Apps & Books** > **Applications** > **Logging** > **SDK Analytics**.

Branding

Branding colors and images can be applied to the SDK user interface. Branding configuration can be used to set

- the primary highlight color of the buttons on SDK authentication screens.
- the logo on SDK authentication screens.
- the logo on the SDK splash screen.

There are two sources of branding configuration:

- *Enterprise branding*, from the enterprise Workspace ONE UEM console with which the end user is enrolled.
- *Static app branding*, in the mobile app resources.

Enterprise branding will in general take precedence over static app branding.

Enterprise Branding

See the Workspace ONE administrator documentation for details of how to configure branding in the UEM console. Different branding can be configured for different organization groups.

The configuration in the console will be retrieved and applied by the SDK instance in your mobile app. If an administrator changes the configuration, the updated configuration will be retrieved and applied at run time. The app needn't be re-installed for branding changes to take effect, for example.

Access to Enterprise Branding Resources

Your app can access the enterprise branding configuration that has been retrieved by the SDK. The configuration could be used to, for example, reflect the enterprise brand in the app user interface.

The enterprise branding configuration will be available after invocation of the callback:
`controllerDidReceive(profiles: [Profile])`

Access the values from the console with code like the following.

```
let brandingPayload = AWController.clientInstance().sdkProfile()?.BrandingPayload
```

The values in AWBranding become set after `controllerDidFinishInitialCheck`. If a value isn't set in the console, then the corresponding property will be `nil`.
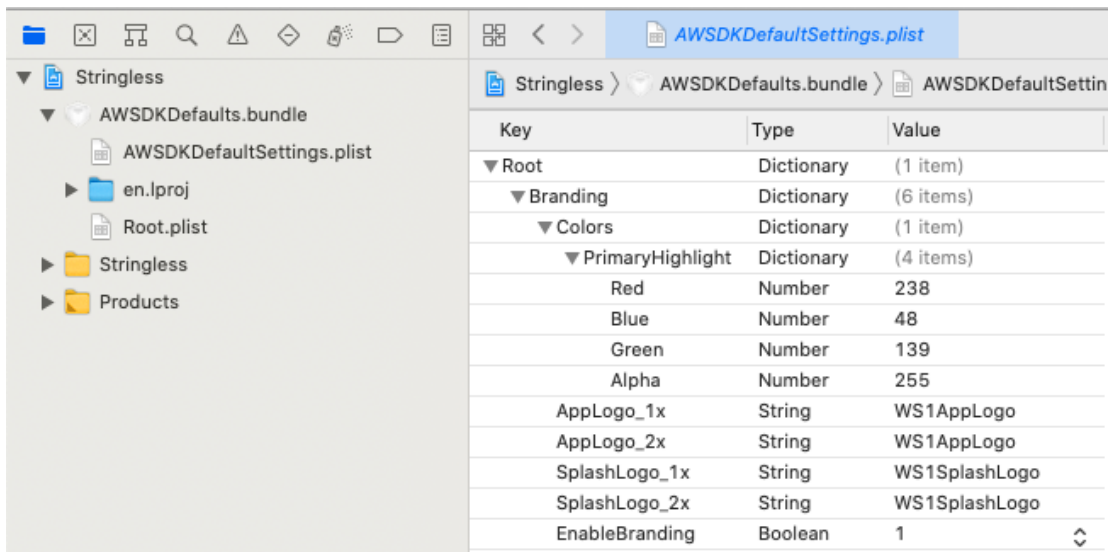
Static App Branding

You configure your static app branding in the SDK settings property list.

The following table lists the properties that relate to branding. The properties are structured in a dictionary hierarchy.

Property	Type	Value
Branding	Dictionary	Parent for all branding properties.
Branding, Colors	Dictionary	Parent for colors.
Branding, Colors, PrimaryHighlight	Dictionary	Primary highlight color.
, Red	Number, 0 to 255	Red component of a color.
, Blue	Number, 0 to 255	Blue component of a color.
, Green	Number, 0 to 255	Green component of a color.
, Alpha	Number, 0 to 255	Opacity of a color.
Branding, AppLogo_1x	String, image set name	App logo for smaller screens.
Branding, AppLogo_2x	String, image set name	App logo for larger screens.
Branding, SplashLogo_1x	String, image set name	Splash logo for smaller screens.
Branding, SplashLogo_2x	String, image set name	Splash logo for larger screens.
Branding, EnableBranding	Boolean	Overall switch.

Table: Branding properties and structure.

The following screen capture shows a sample branding property structure as it might appear in the integrated development environment (IDE).

Screen Capture: Branding keys and values in the SDK settings property list.

The EnableBranding flag can be used as a convenient toggle for the whole static app branding configuration. If EnableBranding is set to NO or zero, then the static configuration is ignored by the SDK.
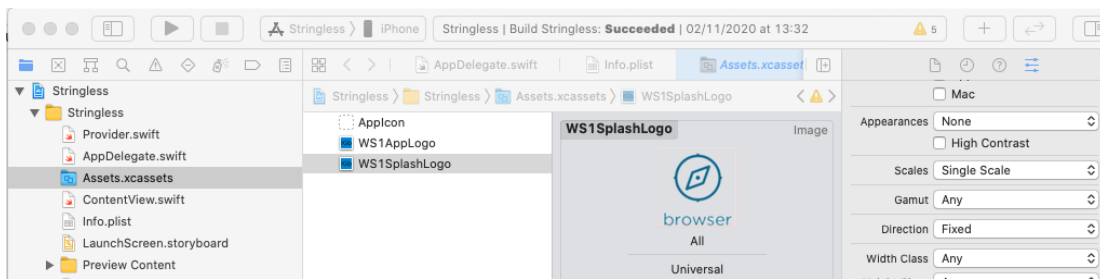
Branding Images

Branding images from the app will be utilized by the SDK as follows.

- **SplashLogo**, if specified, will appear on the SDK loading screen and on the second application login screen.
- **AppLogo**, if specified, will appear on SDK authentication screens.

Two sizes of each logo can be specified, 1x and 2x, for use on different device screen sizes. The SDK will select the more suitable at run time and scale as needed. For best presentation, use a larger image for your SplashLogo, and a smaller image for your AppLogo.

Image resources are specified in the logo properties as image set names. Image sets will be in the application assets catalog. The following screen capture shows how these could appear in the IDE.
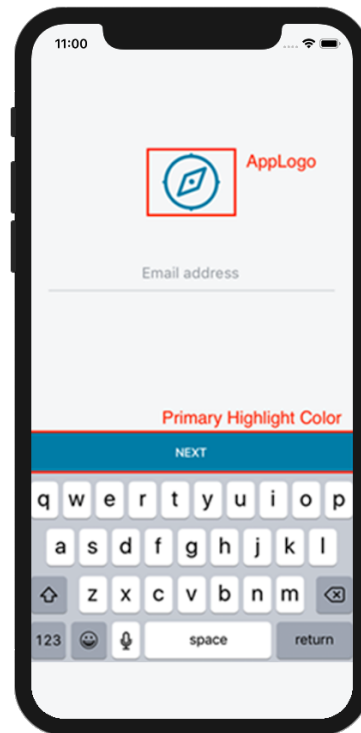

Screen Capture: Branding images in the assets catalog.

Application Screen Captures With Branding

The following screen captures show branding configuration in the SDK user interface.

Screen Capture: Branding on the splash screen

Screen Capture: Branding on an authentication screen

Note About Branding Images

- If the static branding images which include **SplashLogo** and **AppLogo** are not configured and if branding images part of Enterprise Branding are not configured / downloaded, then SDK would apply its default logo across the splash and authentication screens like shown below -

Screen Capture: Default Branding on splash screen
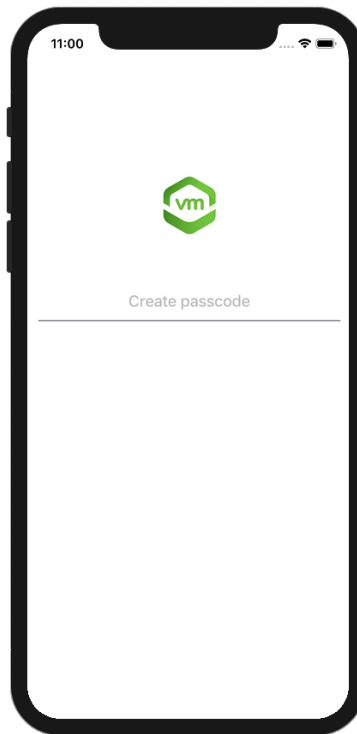
Screen Capture: Default Branding on authentication screen

- If **SplashLogo** and **AppLogo** are configured as part of Enterprise Branding, SDK would replace the default logo's with the Enterprise branding images after those images have been completely downloaded.

- If branding images are configured as part of Enterprise Branding, SDK will **download** these **images asynchronously** i.e. SDK initialization will not wait for image downloads to complete.

Beacon Data Sent Upon Application Unlock or Sent Manually

The beacon is a regular update sent from the VMware Workspace ONE SDK for iOS (Swift) to the Workspace ONE UEM console. The SDK sends this data every time it is unlocked. You can also force the beacon when you want data.

Beacon Update Contents

The beacon update contains the listed information.

Table 1. Contents in the Beacon Update

Type of Information	Data
General	• Device name • Organizational group • Application bundle identifier
Platform	• Device operating system (Apple, iOS) • Device operating system version
User	• User email • User full name • User display name
Enrollment	• Device enrolled • Device unenrolled • Device wipe pending
Compliance	• Device compliance • Application compliance

Send the Beacon Manually

Use an API to send the beacon manually.

```
let beaconTransmitter = SDKBeaconTransmitter.sharedTransmitter

// To send immediately
beaconTransmitter.sendDeviceStatusBeacon(completion: SendBeaconCompletion?)
beaconTransmitter.sendBeacon(updatedAPNSToken: String, completion: SendBeaconCompletion?)

// To start a schedule of how frequently to send.
// (If given time interval is less than 60, frequency will default to 60)
public func startSendingDeviceStatusBeacon(transmitFrequency: TimeInterval = 60)

//To stop the sending the scheduled beacon
public func stopSendingDeviceStatusBeacon()
```

Certificate Pinning

Use certificate pinning to help prevent man-in-the-middle (MITM) attacks by enabling an additional layer of trust between listed hosts and devices.

Certificate pinning requires no code. Just enable SSL pinning in the Workspace ONE UEM console and upload your certificate.

Check the Compromised Status of Devices with Compromised Protection

Workspace ONE UEM detects jailbroken devices and can wipe compromised devices if enabled in the Workspace ONE UEM console.

Compromised protection requires no code unless you want to manually check the status of the device.

Check Compromised Protection Status

To check the status of a device directly in your application, whether the device is online or offline, call the refreshDeviceCompromisedStatus() API from the DeviceInformationController singleton class.

```
//Swift
DeviceInformationController.sharedController.refreshDeviceCompromisedStatus { isJailbroken, evaluationToken, identifier in
    DispatchQueue.main.async {
        if isJailbroken {
            AWLogInfo("Device is compromised")
        }
        else {
            AWLogInfo(""Device is not compromised"")
        }
    }
}
```

Dynamic Compromise Detection Requirements

Dynamic compromise detection for iOS sets SDK-built apps to securely update the compromise detection algorithm over-the-air. Apps that use this feature do not need to update or re-release after compromise detection rule updates. To configure this feature, update to the supported SDK version and ensure that devices can access specific URLs. To use dynamic compromise detection, update the SDK version and ensure that devices can access specific URLs.

- The SDK-built app must consume Workspace ONE SDK for iOS (Swift) v23.04 or later.
- To receive the latest compromise detection rules, ensure that devices can connect to the listed hosts -
  - api.na1.region.data.vmwservices.com
  - sdk-config.gov.data.vmwservices.com
  - discovery.awmdm.com
  - signing.awmdm.com
- If devices cannot access these URLs, they still get compromise detection but rules only update when the SDK-built app consumes the latest SDK. This lapse in rule updates might result in false positives.

Note: If you use VMware Workspace ONE Tunnel, ensure that your traffic rules are configured to allow devices to connect to the listed URLs.

Query Devices for MDM Information with DeviceInformationController

Use the DeviceInformationController singleton class to query devices for mobile device management (MDM) information.

The class returns the listed MDM information.

- Enrollment status
- Compliance status
- Managed status
- Management type
- Organizational group name
- Organizational group ID
- Device services URL
- Single sign on status
- Compromised status
- Console version
- Shared device status
- is App Supported

Requery Method

The method queries the console, and the console sends a query command to the device to collect certain types of device information.

SDK Logging APIs for 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 Workspace ONE UEM console. The SDK-built application collects logs over time and stores them locally on the device until another API or command is invoked to transmit the logs.

Note: When an enterprise wipe occurs, the console does not purge the log files. You can retrieve logs after a device re-enrolls to determine what issues occurred in the last enrollment session to cause the enterprise wipe.

Table 1. SDK Logging Level APIs and Level Descriptions

Level	Logging API	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.

SDK Logging APIs

The following API will allow you to send logs to UEM or alternatively save SDK log data.

The developer can manually trigger the transmission of SDK logs to the Workspace ONE UEM console with APIs. The Workspace ONE UEM admin can use the View Logs menu item to get logs for an application when they are transmitted. Alternatively, Workspace ONE SDK provides a way to retrieve persisted SDK logs. The retrieved logs are in **utf8** encoded **Data** object. The maximum size of the returned log object will be 5 MB.

Developer APIs

- iOS (Swift) - AWController

```
public func sendLogDataWithCompletion(
    completion: @escaping (success: Bool, _ error: NSError?) -> Void
)
```

- iOS (Swift) - WS1SDKDataProvider

```
do {
    let logProvider = WS1SDKDataProvider()
    let logChunks = try logProvider.fetchSDKLogChunks()

    logChunks.forEach { logChunk in
        // Each chunk data will be not more than 5 MB.
        let logData = logChunk.data

        // If a temporary file needs to be created with the log data, app can use the suggested file name.
        let suggestedFileName = logChunk.suggestedFileName
    }

    //Perform logic here with fetched log data
} catch let error {
        print("Failed to fetch SDK Log data with error: \(error)")
}
```

- iOS (Objective-C) - AWLog

```
- (void)sendApplicationLogsWithCompletion:
    (void(^)(BOOL success, NSError *error))completion;
- (BOOL)hasAWLogs;
```

SDK Log Types

Workspace ONE UEM displays logs for applications that report application failures and that report application-specific data. These logs integrate with the VMware Workspace ONE 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, ettings > 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 Workspace ONE UEM console without the need for extra code in the SDK application.

Configure Logging for the Default SDK Profile

Use Logging so the system records data for applications the use the VMware Workspace ONE SDK framework. The Workspace ONE UEM system collects logs until the log file size reaches 200 MB for SaaS environments. If the log size exceeds 200 MB, the system stops collecting logs. The Workspace ONE UEM console notifies you when your application log size reaches 75% of 200 MB. To act on the application log size, contact your Workspace ONE UEM Representative.

- Ask for an increase in your application log size.
- Ask for a purge of your application log. The system can purge logs older than two weeks.

Procedure

1. Navigate to **Groups & Settings > All Settings > Apps > Settings and Policies > Settings**.
2. Select **Enabled for Logging**.
3. Choose your **Logging Level** from a spectrum of recording frequency options.
4. Select **Send logs over Wi-Fi only** to prevent the transfer of data while roaming and to limit data charges.
5. **Save** your settings.

Request Application Logs for SDK-Built Apps

Request applications logs for your SDK-built applications from the device record in the console.

Procedure

1. Navigate to **Devices > List View** and select the device.
2. Select the **Apps** tab, select the SDK-built app, and choose **Request Logs**.
   The **Request Logs** button displays after you select the application.
3. Complete the settings in the **Request Logs** window. You can retrieve logs that are currently available or you can select to capture a log type for a duration of time.
4. To retrieve the logs, navigate to **Apps & Books > Applications > Logging > App Logs**.
5. Find the log for the application with the **App Name** column and download the file.

Configure View Logs for Internal Applications

Use the View Logs feature to access available log files pertaining to applications that use the Workspace ONE SDK framework. Log types include all logs, crash logs, and application logs. With this feature, you can download or delete logs.

Filter options using the **Log Type** and **Log Level** menus so that you can find the type or amount of information to research and troubleshoot applications that use the SDK framework.

Procedure

1. Navigate to **Apps & Books > Applications > Native** and select the **Internal** tab.
2. Select the application and then select **More > View > Logs** option from the actions menu.
3. Select desired options depending on if you want to act on specific devices (selected) or to act on all devices (listed).

| Setting	Description
Download Selected	Download selected logs with information pertaining to applications that use the Workspace ONE SDK framework.
Download Listed	Download all logs in all pages with information pertaining to applications that use the Workspace ONE SDK framework.
Delete Selected	Delete selected logs with information about applications that use the Workspace ONE SDK framework.
Delete Listed	Delete all logs in all pages with information about applications that use the Workspace ONE SDK framework.

Restrictions

Offline Access

The offline access function allows access to the application when the device is not communicating with the network. It also allows access to Workspace ONE UEM applications that use the SSO feature while the device is offline.

The Workspace ONE 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 lock method of the AWSDKDelegate so your application can act locally.

Screenshot Restrictions

Admin can restrict taking screenshot via the DLP settings.
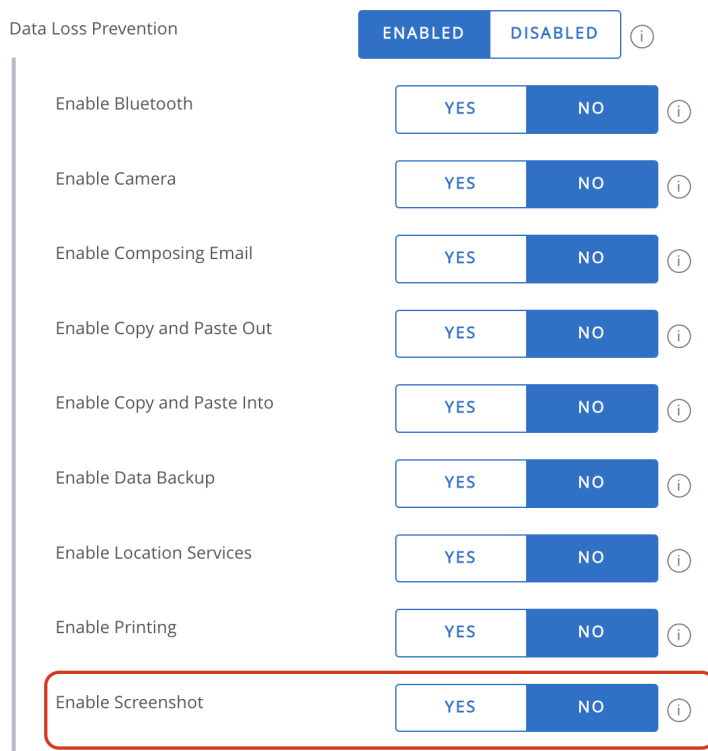
If taking screenshot is disabled and user captures a screenshot or starts screen recording, Workspace ONE SDK will present a blocker screen to the user describing that taking screenshot and screen recording has been restricted by the admin. The user will have to dismiss the screen by tapping on a "Got it" button. Please note that the blocker screen acts as hindrance to the user but it does not prevent the actual capture of the screenshot.

Note

Automatic enforcement of these restrictions would need AWController to be started by the application.

Custom Settings for the SDK

The VMware Workspace ONE SDK for iOS (Swift) allows you to define your own custom settings for your application using an SDK profile.

You can paste raw text in the custom settings section, and the SDK makes this content available inside the application using the AWCustomPayload object.

You can define an XML, JSON, key-value pairs, CSV, or plain text for your settings. Parse the raw text in the application once it is received.

Screen Capture: Configuration of Custom Settings in the management console

Method Usage

```
let customPayload = AWController.clientInstance().sdkProfile()?.customPayload
```

Encrypt Data on Devices

The VMware Workspace ONE SDK for iOS (Swift) offers the use of basic encrypt and decrypt methods to operate on raw data that the system encrypts using the SDK's internal encryption keys.

These methods are defined in the AWController. Important: Do not use these encryption methods on any mission critical data or data that you cannot recover. Examples of unrecoverable data include no backup on a server or if the data cannot be re-derived through other means. The encrypted key (and associated encrypted data) is lost in the event that an end user deletes the application or if an enterprise wipe.

Prequisites

Before you call the encryption methods, ensure the AWControllerDelegate receives no errors.

- Swift: Applications must ensure that AWControllerDelegate receives the `controllerDidFinishInitialCheck(error: NSError?)` callback with no errors before they call the encryption methods.

- Objective-C: The AWControllerDelegate callback method is `-(void)initialCheckDoneWithError:(NSError * _Nullable)error;`

Encryption Strength and Authentication Mode

The strength of the encryption depends on the enabling of the authentication mode.

If you set authentication passcode or username and password, then the system derives the key used for encryption from the passcode or username and passcode the user enters. The system keeps the key in device volatile memory for additional security.

If you disable authentication, the system randomly generates the encryption key and persists it in device storage.

Encrypt Data not Stored with Core Data

The Workspace ONE SDK for iOS (Swift) provides the ability to encrypt data that Core Data does not store. These methods take in the data input and return back either the encrypted or decrypted data. These methods are only used for the transformation of the data. The application developer is responsible for the storage of the encrypted data.

- Encryption Method: Swift

```
public func encrypt(_ data: Data) throws -> Data
public func decrypt(_ data: Data) throws -> Data
```

- Encryption Method: Objective-C

```
(NSData * _Nullable)encrypt:(NSData * _Nonnull)data
                    error:(NSError * _Nullable * _Nullable)error SWIFT_WARN_UNUSED_RESULT;

(NSData * _Nullable)decrypt:(NSData * _Nonnull)data
                    error:(NSError * _Nullable * _Nullable)error SWIFT_WARN_UNUSED_RESULT;
```

Error Codes Defined and Examples

The enum AWSDKCryptError defines the error codes for the error thrown by the methods.

- Encrypt

```
let controller = AWController.clientInstance()
let plainData: Data = .. //assign data to be encrypted
do {
    let encryptedData = try controller.encrypt(plainData)
    //save encryptedData for future use
    //...
} catch let error {
    print(" failed to encrypt data with error: \(String(describing: error))")
}
```

- Decrypt

```
let controller = AWController.clientInstance()
let encryptedData = ..//fetch data previously encrypted using Encrypt method above

do {
    let decryptedData = try controller.decrypt(encryptedData)
    //do something with decryptedData
    //...
} catch let error {
    print(" failed to encrypt data with error: \(String(describing: error))")
}
```

Enable and Code APNs in the Application

To use Apple push notifications in your SDK-built application and Workspace ONE UEM, enable the use of APNs and add code to support push notifications.

Setting a token value to AWController initiates the call to the console because it sends the beacon. Assign the token value to AWController only after the token value has changed.

Setting the token value to nil clears the token value from the console and you cannot use the token to send push notifications. Note: The sample code is for reference and can be adjusted per the app requirements. See the sample app for more examples of how the listed methods are used.

Procedure

1. Select Target and enable push notifications in capabilities. You see two checks in Push Notification.
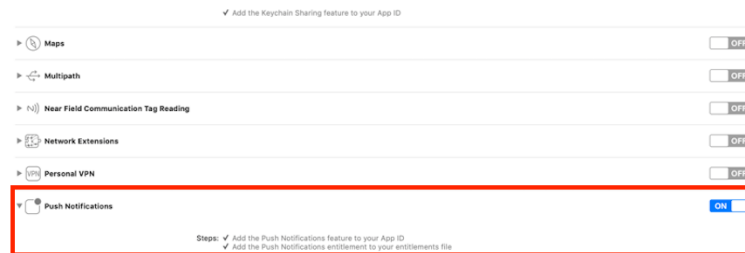
Push Notifications Example

2. Add `import UserNotifications` to the top of AppDelegate.swift.

3. Add applicable methods to the end of AppDelegate.swift.

```swift
func registerForPushNotifications() {
if #available(iOS 10.0, *){
    UNUserNotificationCenter.current()
        .requestAuthorization(options: [.alert, .sound, .badge]) {
            granted, error in
            print("Permission Granted\(granted)")
            guard granted else { return }
            self.getNotificationSettings()
        }
    }
    else {
        let notificationSettings = UIUserNotificationSettings(
            types: [.alert, .badge, .sound], categories: nil)
        DispatchQueue.main.async {
            UIApplication.shared.
                registerUserNotificationSettings(notificationSettings)
        }
    }
}


@available(iOS 10.0, *)
func getNotificationSettings() {
    UNUserNotificationCenter.current()
    .getNotificationSettings {settings in
        print("Notification settings\(settings)")
        guard settings.authorizationStatus == .authorized else { return }
        DispatchQueue.main.async {
            UIApplication.shared.registerForRemoteNotifications()
        }
    }
}

func application(
    _ application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data
) {
    let tokenParts = deviceToken.map {
        data in String(format: "%02.2hhx", data)
    }
    let token = tokenParts.joined()
    print("Device Token: \(token)")
    let beaconTransmitter = SDKBeaconTransmitter.sharedTransmitter
    beaconTransmitter.sendBeacon(updatedAPNSToken: token) { success, error in
        if (success) {
            AWLogInfo("Device token sent successfully")
        }
        else {
            AWLogError("failed to send device token: \(String(describing: error))")
        }
    }
}

func application(
    _ application: UIApplication,
    didFailToRegisterForRemoteNotificationsWithError error: Error
) {
    print("Failed to register: \(error)")
}
```

4. Add `registerForPushNotifications()` near the end of
`application(_:didFinishLaunchingWithOptions:)`, and before `return:`.

Enable APNs in the Console

Use SDK-built applications to send Apple push notifications to applicable devices. Enable the SDK-built app
to use APNs. This task assumes that the SDK-built app is already uploaded and managed in the Workspace
ONE UEM console. These apps are available in an app store and they use Production APNs certificates.

Prerequisites

Generate your production APNs certificates so you can upload the certificates to the Workspace ONE
UEM console. For details, visit the topic Registering Your App with APNs on the Apple Developer site.

Procedure

1. Navigate to **Apps & Books** > **Applications** > **SDK-built app** and choose **Edit**.
2. Select the **Files** tab and select **Yes** for **Application Supports APNs**.
3. Select **Production for APNs Certificate**.
4. Use **Upload** to add your certificates to the console as an **APNs Production Certificate**.
5. Select **Save & Assign**. Editing the assignment is optional and not necessary to finish this task. You can **Save and Publish** from the assignment module.

APIs to Use Custom Certificates for Your SDK-Built Apps

The Workspace ONE SDK for iOS (Swift) has APIs to evaluate server trust and verify configured certificates. API to Validate Server Trust

- Declaration

```
func validate(
    serverTrust: SecTrust,
    trustStore: CertificatesTrustStore,
    strictness: SSLTrustStrictness
) -> Bool
```

- The admin configures trusted certificates as Credentials in the SDK profile. When the SDK starts, it fetches custom anchors and SSL certificates configured by the admin and stores them securely as configured.
- While connecting to a network host, the app can receive a challenge. During this challenge, the app can use an API to validate the server trust and can decide to allow or cancel the connection.

- Parameter Explanations - ServerTrust, TrustStore, and SSLTrustStrictness

  - `ServerTrust`

    Retrieve the `SecTrust` object from the `ProtectionSpace` given to the app for authentication by the URLSession task.

    The API copies the certificate chain and policies for evaluation, so that the app can perform additional operations on the `SecTrust` in its original form.

  - `TrustStore`

    The API considers the `TrustStore` type while it evaluates the `ServerTrust`. The API supports only deviceAndCustom and custom types for `TrustStore`.

    If you configure the type as custom, the API uses only custom anchors or self-signed SSL certificates (those anchors or certificates configured by the admin in a **Credentials** payload) to evaluate trust. If your server uses intermediate certificate authorities, you must add the intermediate certificate authorities in the **Credentials** payload.

    If type is deviceAndCustom, the SDK uses system trust store combined with the configured certificates to evaluate the ServerTrust.

    Note:
    You can use self-signed SSL certificates with or without any CA certificates by adding them directly to the SDK Credentials payload.

  - `SSLTrustStrictness`

    The SDK uses `SSLTrustStrictness` to consider `recoverableTrustFailure SecTrustResultType` as an end result, to be trusted or not trusted.

    - If the value for strictness is `strict`, the `SecTrustResultType` end result is `recoverableTrustFailure` and is not trusted.
    - If the value for strictness is `ignore`, the `SecTrustResultType` end result is `recoverableTrustFailure` and is trusted.

If the `TrustStore` is custom, the SDK forms a complete chain with the certificates from the `SecTrust` and validates the chain. Validation is according to the policies set in the SecTrust. If `TrustStore` is deviceAndCustomThe, the SDK forms the chain up to a certificate that is in the trusted list.

- Certificates Considered for Server Trust Validation
  - Root CA certificates
  - Intermediate CA certificates
  - SSL certificates

  Upload public X509 certificates in DER or PEM format. The SDK does not consider certificates uploaded with a private key for server trust evaluation.

API to Retrieve Configured Certificates

- Declaration

```
func retrieveStoredPublicCertificates(
    completion: (
        _ certificateMap: [String: [PublicCertificate]]?,
        _ error: NSError?
    ) -> Void
)
```

- Parameter Explanation Completion

  The completion block is called with the configured certificates map. It returns an error if there is any problem while retrieving the certificates.

  The API returns a map. The keys are represented using the constants from AWCertificateUsageKey class. Corresponding values are array of Public Certificate Objects. You can query certain x509 attributes from the PublicCertificate objects and verify the configuration.

```
@objc(AWCertificateUsageKey)
public class CertificateUsageKey: NSObject {

    // Certificate of Usage key to reflect Integrated Authentication
    public static let integratedAuthIdentity: String

    // Certificate of Usage key to reflect Integrated Authentication
    public static let uncategorizedIdentity: String

    // Certificate of this usage are used for signing requests for MAG Proxy
    public static let magSigning: String

    // Certificate of this usage are used for signing requests for Tunnel Proxy
    public static let tunnelSigning: String

    // Certificates of type SSL
    public static let selfSignedSSLCerts: String

    // Certificates of type Custom Anchors
    public static let customTrustedAnchorCerts: String

    // SDK doesn't have specific usage for this type of certificates
    public static let others: String
}
```

VMware Workspace ONE SDK for iOS (Swift) and the Apple App Review

Deploy apps that use the Workspace ONE SDK for iOS (Swift) to the App Store without dependency on other Workspace ONE UEMcomponents. The SDK includes a mode for your application for use during the Apple App Review process.

This app review mode removes dependencies on the broker applications such as the Workspace ONE Intelligent Hub for iOS, Container, and the Workspace ONE application. It also enables the app reviewer to access the application without enrolling with Workspace ONE UEM.

Explanation of the Process

Build your application and incorporate the Workspace ONE SDK for iOS (Swift). Then, build a test environment in Workspace ONE UEM and prepare the application for submission to the app review process. For general steps in the process, see Steps to Configure App Review Mode.

Build a Test Environment in Workspace ONE UEM

Create a test environment in Workspace ONE UEM that you use only for this app review process. For details on how to create this environment and how to upload your application to it, see Configure an App Review Mode Testing Environment in the Workspace ONE UEM Console.

Identify the Server URL and Group ID

To help your application work for the review process without dependencies on other Workspace ONE UEM components, follow the procedure in Declare the App Review Server and Group ID in the SDK PLIST.

Steps to Configure App Review Mode

Deploy apps that use the VMware Workspace ONE SDK for iOS (Swift) to the App Store without dependency on other Workspace ONE UEM components. The SDK includes a mode for your application for use during the Apple App Review process.

This app review mode removes dependencies on the broker applications such as the Workspace ONE Intelligent Hub for iOS, VMware Container, and the Workspace ONE application. It also enables the app reviewer to access the application without enrolling with Workspace ONE UEM.

Important: Use this work flow only on applications built with the Workspace ONE SDK that you submit to the App Store for review. Do not use this work flow for any other application development processes. Also, do not use the process in a production environment. This process is only supported for use in a test environment for applications you submit to Apple's App Review.

Procedure

1. Integrate the SDK with your application.

2. Configure the app review mode testing environment in the Workspace ONE UEM console, upload the application IPA file, assign it an SDK profile, and deploy it to the test environment.

   See Configure an App Review Mode Testing Environment in the Workspace ONE UEM Console.

3. Assign an app review mode server and a group ID to the SDK PLIST.

   See Declare the App Review Server and Group ID in the SDK PLIST.

4. Test the IPA in the test environment.

   See Test the App Review Mode Testing Environment in the Workspace ONE UEM Console.

5. Run the app store build script.

   See [Build Script Information for App Store Submission].

6. Submit your application for review to the Apple App Store ensuring to add the app review mode server, group ID, and user credentials from the test environment to the submission.

Configure an App Review Mode Testing Environment in the Workspace ONE UEM Console

With help from your admin, configure a testing environment in the Workspace ONE UEM console. Upload your application to this environment so that the app reviewer can review your application without dependencies on other Workspace ONE UEM components.

Prerequisites

- Integrate the Workspace ONE SDK for iOS (Swift) with your application.
- You need Workspace ONE UEM system admin permissions to configure these components. If you do not have these permissions, ask your Workspace ONE UEM Admin for help.
- Ensure that you create a testing environment that hosts no production applications and components. Use this app review mode environment only for the app review process.
- Configure all options in the app review organization group.

Procedure

1. Configure a special organization group for app review mode in the Workspace ONE UEM console. Record the group ID for later entry to the SDK PLIST.
2. Configure an app review mode user with credentials in the Workspace ONE UEM console. You give these credentials to the app reviewer so record the credentials.
3. Create a smart group and add the user to the group. Workspace ONE UEM deploys applications based on assignment groups, specifically the smart group type.
4. Configure the SDK profile. Use the default SDK profile or a custom SDK profile. Whatever SDK profile you use, ensure that the desired SDK features are enabled. Features to review are the Authentication Type, Single Sign On, and the App Tunnel Mode.
5. Upload the application binary (IPA) to the internal application area or the public application area of the Workspace ONE UEM console. Ensure that you assign the SDK profile to the application and assign the test smart group to the application. The bundle identifier must match the application submitted to the App Review process.
6. Disable the requirement for MDM enrollment so the app reviewer can access the application without enrolling with MDM. Although the setting are nested under the Content Locker, it applies to all applications. Improvements to the user interface are planned for the future.
   - Ensure you are in the app review mode organization group.
   - Navigate to Groups & Settings > All Settings > Content > Applications > Content Locker.
   - In the General area, disable Require MDM Enrollment.
   - Select Save.

Declare the App Review Server and Group ID in the SDK PLIST

To prepare to submit your application to the Apple App Review process, add the app review mode server URL and the group ID. These strings allow the reviewer to review your application without the need for other Workspace ONE UEM components.

Procedure

1. If you have not done so, in your Xcode project, create a bundle named AWSDKDefaults.

2. If the AWSDKDefaults bundle does not have a PLIST named AWSDKDefaultSettings.plist, create this PLIST in the bundle.

3. Create a key in the PLIST with the data type string. Name this key com.vmware.air-watch.enrollment.test-server-url.

   This name is case sensitive.

4. Set the value of this key to the server URL of the Workspace ONE UEM environment you set up in Configure an App Review Mode Testing Environment in the Workspace ONE UEM Console.

   Ensure to meet these requirements for the URL.

   - Include https:// before the URL.
   - Ensure the URL is the exact device services server URL. Do not use the console or API server URL.
   - Do not include /deviceservices at the end of the URL. The SDK appends this automatically.

5. Create another key in the PLIST with the data type string. Name this key com.vmware.air-watch.enrollment.test-org-group-id.

   This name is case sensitive.

6. Set the value of this key to the group ID of the app review group you setup in Configure an App Review Mode Testing Environment in the Workspace ONE UEM Console.

Test the App Review Mode Testing Environment in the Workspace ONE UEM Console

Test that the IPA file, server URL, group ID, and user credentials work before you submit the application for review.

Procedure

1. Attempt to run the app on a device without any previous app data.

   This action ensures that stale URL and device information is not present on the device. It also ensures there are no other Workspace ONE UEM apps on the device.

2. Enter the server URL and group ID when the app prompt for these options.

3. Enter the user credentials when prompted.

Results

If the SDK permits you to continue without error and controllerDidFinishInitialCheck is called, the test environment and components are successful.

Migrate the Objective-C Version to the Swift Version

To migrate to a version of the Workspace ONE SDK for iOS (Swift), remove the old SDK and add the current one to your environment.

See Component Changes in the Workspace ONE SDK for iOS (Swift) for changes to make to your project to prevent build errors.

Share Your Keychain

Share your keychain between the SDK applications so you can use all the SDK capabilities. See Keychain Access Group Entitlements.

Remove the Objective-C Version of the SDK

Delete the listed Workspace ONE SDK for iOS (Swift)frameworks and libraries to remove the SDK.

Procedure

1. On the General tab in your project, delete the AWSDK.framework from both the Embedded Binaries and Link Framework and Libraries areas.
2. Open the Build Phases tab in the project settings of your application.
3. Delete AWKit from your project.
4. Delete AWlocalization from your project.

Add the Swift Version of the SDK

Add Workspace ONE SDK for iOS (Swift) frameworks and edit the locations of the listed calls to migrate SDK behaviors to the current version. If you do not edit the listed call locations, the UI behavior is inconsistent with the previous SDK version.

Procedure

1. Drag and drop the current AirWatchSDK framework and the AWCMWrapper file into your Link Binary with Libraries step in the build phase section of your project settings.
2. Change the location of your StartSDK call. Call it in the didFinishLaunchingWithOptions method that is inside your application delegate class. In versions before the Workspace ONE SDK v17.x, you called awcontroller.start() within the applicationDidBecomeActive method.
3. Build your project.
4. Resolve naming differences and API differences that changed in the new SDK causing build errors.

Component Changes in the Workspace ONE SDK for iOS (Swift)

If you migrate an older version of the SDK to install it, review the list of changed components. Update names and locations of components to prevent or resolve build errors caused by the differences between SDK versions.

Samples present the old version of the code followed by the current code.

Component	Sample Code

AWController start
 	
In the previous SDK you called **awcontroller.start()** within the **applicationDidBecomeActive** method.
 	
In the current SDK, start the SDK within the **didFinishLaunchingWithOptions** method inside your application delegate class.
	`///5.9.X Implementation func applicationDidBecomeActive(_ application: UIApplication) let awc = AWController.clientInstance() awc.delegate = self awc.callbackScheme = "myAppName" awc.start() }`
	
	`///Swift version Implementation func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool { let awc = AWController.clientInstance() awc.delegate = self awc.callbackScheme = "myAppName" awc.start() return true }`

CanhandleProtectionSpace (Integrated Authentication)
 	
Update the code for authentication challenges and chain validation.
	`///5.9.X Implementation try AWController.clientInstance().canHandle(challenge.protectionsSpace)`
	
	`///Swift version Implementation try AWController.clientInstance().canHandle(protectionsSpace: challenge.protectionsSpace )`

AWLog singleton (Logging)
 	
Use this instead of the AWController to send logs.
	`///5.9.X Implementation AWLog.sharedInstance().sendApplicationLogs(success, errorName)`
	
	`///Swift version Implementation AWController.clientInstance().sendLogDataWithCompletion { (success, error) }`

Network Status
 	
Update the front of the enum to AWSDK.
	`///5.9.X Implementation AWNetworkActivityStatus`
	
	`///Swift version Implementation AWSDK.NetworkActivityStatus`

Profiles and profile payloads
 	
Drop the AW from the front of profiles.
	`///5.9.X Implementation AWProfile`
	
	`///Swift version Implementation Profile`

Custom Settings
 	
Access custom settings through **AWController** instead of **AWCommanManager**
	`///5.9.X Implementation AWCommandManager().sdkProfile().customPayload`
	
	`///Swift version Implementation AWController.clientInstance().sdkProfile()?.customPayload`

Account object
 	
The account object is now a property on **AWController** instead of an accessor method.
	`///5.9.X Implementation AWController.clientInstance().account()`
	
	`///Swift version Implementation AWController.clientInstance().account`

User credentials	`///5.9.X Implementation AWController.clientInstance().updateUserCredentials(completions: {`

Component	Sample Code
| | ```
(success, error) in { ... } )

///Swift version Implementation
AWController.clientInstance().updateUserCredentials(with: {
(success, error) in { ... })
``` |
| **OpenInURL calls** | ```
///5.9.X Implementation
AWController.clientInstance().handleOpen(url, fromApplication:
sourceApplication)

///Swift version Implementation
AWController.clientInstance().handleOpenURL(url, fromApplication:
sourceApplication)
``` |
| **DeviceInformationController**

Replace **MDMInformationController** with **DeviceInformationController** | NA |
| **Manually load commands**<br><br>Use an API on AWController to force commands to reload instead of using the command manager. | ```
///5.9.X Implementation
AWCommandHandler.sharedHandler().loadCommands()

///Swift version Implementation
AWController.clientInstance().loadCommands()
``` |

Multitasking Split View Support

The VMware Workspace ONE SDK supports split view on iPad for multitasking.

The following actions whenever triggered by any of the SDK applications, all the other SDK applications in foreground will detect the same action and perform the required operation.

1. Unenroll
2. Logout
3. Lock and Unlock: In this case the action will be detected only by the SDK applications from same keychain cluster.

Important: For compatibility, Hub or other anchor application should integrate the VMware Workspace ONE SDK version 21.10 or later. Also, Login event is not supported for the feature and user would need to terminate and relaunch the application after logging in Hub or other anchor applications.

for information on split view see the Apple webSite

Fetch Application Status and Device Information

Workspace ONE SDK fetches the Application Status and takes appropriate action. Applications can programatically check the status of the application or display the details in UI using SDK provided API

```
//Swift
DeviceInformationController.sharedController.fetchApplicationAssignmentStatus { appStatus, deviceInformation in

}
```

Applications can also programatically get the details like organization group and the UEM host to which the device is enrolled.

```
//Swift
let awController = AWController.clientInstance()
let organizationGroup = awController.enrollmentInfo.organizationGroup
let host = awController.enrollmentInfo.hostName
```

Note

In order to use these APIs, application would need AWController instance to be started.

Application Attestation

The Apple app attestation service will be used to verify the bundle identifier of custom SDK apps during enrolment.

- Every app must set the teamID property of the AWController instance before starting the SDK.
- The service must be reachable during enrolment. See the advice from Apple about use of their products on enterprise networks, for example under App validation here https://support.apple.com/en-us/HT210060.

Support for Tunnel with WKWebView

Workspace ONE SDK supports Tunneling with WKWebView from iOS 17 onwards.

Requirements

1. Xcode 15+

2. iOS 17+

Enabling Tunnel with WKWebView

```
#if swift(>=5.9)
if #available(iOS 17.0, *) {
    do {
        // Enabling Tunnel on an instance of WKWebView.
        // WorkSpace One SDK needs to be intialised before enabling Tunnel.
        try self.wkWebView?.enableTunnel()
    } catch let error {
        print("Failed to enable Tunnel - \(error)")
    }
}
#endif
```

WorkspaceOne SDK Error lists

Error Code	Error Name	Possible reasons	Error Description
0	internalError	Internal error occurred within the SDK. Returns when failed to enroll or unknown server error occur.	Internal error occurred within the SDK.
1	stopSDKRequested	Call to stop AWSDK from app before initialization could be done.	Call to stop AWSDK from app before initializations could be done.
2	registeringApplicationBlocked	When attempting to register the application, the console has blocked us from registering.	When attempting to register the application, the console has blocked us from registering.
3	enrollmentInformationSetup	When failed to fetch Enrollment status or profile.	Enrollment status could not be completed.
4	emptyProfiles	No SDK profiles were retrieved, and there are no saved profiles.	No profiles have been downloaded, and there were no saved profiles.
5	proxyFailedToStart	Failed to enforce proxy configuration.	Proxy server failed to start with provided configuration.
6	integratedAuthenticationCertificatesNotDownloaded	Invalid certificate Data received, certificate expiry or server responded with empty certificate details.	Integrated Authentication certificates could not be downloaded.
7	applicationIdentityNotSet	When user authentication or device registration fails.	Failed to verify application status from server.
8	failedToFetchPinningCertificate	Invalid certificate Data received, certificate expiry or server	Failed to download certificates to

Error Code	Error Name	Possible reasons	Error Description
		responded with empty certificate details.	do certificate pinning.
9	failedToFetchEnvironmentInformationFromAnchor	App Attestation failed, server not reachable.	Failed to fetch environment information for app from HUB.
10	callBackSchemeNotConfigured	Callback scheme has not been set on AWController's instance. Please set your app's custom URL scheme.	Callback scheme has not been set on AWController's instance. Please set your app's custom URL scheme.
11	airWatchApplicationSchemeNotInAllowedLists	No AirWatch application URLs schemes have been whitelisted in app's info.plist. Please whitelist AirWatch app's custom URL scheme.	No AirWatch application URLs schemes have been allowed in app's info.plist. Please add AirWatch app's custom URL scheme.
12	anchorRequiredForThirdPartyApplictionBootstrap	Attempted to run a third party application as standalone when a container app is not installed.	Attempted to run a third party application as standalone when a container app is not installed.
13	failedToReportUnenrollmentStatus	Server not reachable.	Failed to report unenrollment status to the console.
14	userReachedMaximumAllowedUnlockAttempts	User Failed to Unlock App in Maximum Allowed Attempts.	User Failed to Unlock Protected Data in Maximum Allowed Attempts.
15	deviceIsCompromised	Device is jailbroken.	Device is compromised.
16	consoleVersionNotCompatible	Console version is not compatible with current version of SDK.	Console version is not compatible

Error Code	Error Name	Possible reasons	Error Description
			with current version of SDK.
17	invalidKeyWrappingConfiguration	Return when invalid AWSDKConfiguration provided	Disable Strict Key Wrapping is not allowed.
18	protectedDataIsUnavailable	Fail to read from keychain as key might have updated.	SDK cannot proceed as protected data is unavailable.
19	tunnelFailedToStart	Failed to enforce Tunnel configuration.	Tunnel was not able to start using the profile.
20	missingExpectedRequirements	When SDK requirements like callback schema is not set.	AWController is expected to be provided with proper requirements.
21	dataMigrationFailure	Failed to move legacy data in upgrade scenario.	Failed to migrate SDK data.
22	deviceNotEnrolled	Current device got deleted from console or device never enrolled in provided environment.	Device is not Enrolled with any WS1 UEM Console. Cannot complete operation.
23	failedToSetupAccessControl	When SDK failed to unlock or SDK security setup failure.	Device failed to setup access control.
24	applicationNotAssigned	When app is not assigned.	It appears this app is not assigned to your device.
25	credentialsFetchFailed	When SDK failed to fetch certificates from Server.	Error while fetching configured certificates.
26	deviceAlreadyEnrolled	Returns when SDK already has enrollment info.	Device is already enrolled.
27	missingRequiredInformation	OG or server URL details not found.	Missing Required Information.

Error Code	Error Name	Possible reasons	Error Description
28	nonAirWatchConsole	When response header doesn't contain a console-version.	Non AirWatch Console.
29	enrollmentBlockedThroughExtension	Trying to enroll SDK with app extension.	Enrollment blocked through app extension.
30	enrollmentBlocked	When customer app tries to register without enrolling through HUB.	When customer app tries to register without enrolling through HUB.
31	sharedDeviceNotCheckedOut	Trying to access app without user login in hub.	Shared Device required checkout before being used.
32	serverIsNotReachableForRequiredSetup	When SDK failed to determine user CICO status due to network issue.	Server is not reachable for required setup.
33	policySigningCertFetchFailed	SDK failed to fetch NAIP policy signing certificate from console.	Policy Signing Certificate Fetch Failed.
34	applicationNotManaged	When application does not contain managed settings.	Application not managed.
35	enrollmentInfoNotMatched	Enrollment Info mismatched.	Enrollment info not matched.
36	failedToFetchCrossClusterInfoFromAnchor	App Attestation failed, server not reachable.	Failed to fetch cross cluster information for app.
37	crossClusterMisconfigurationFailure	Two keychain sharing apps should not contain different cross cluster configuration.	Cross cluster mode invalid setup.
38	userCancelledEnrollment	When user cancels the enrollment like dismissing the enter server URL screen.	User cancelled Enrollment.
39	invalidServerDetails	When server details are not proper.	Invalid server details.

Document Information

Revision History

30jun2020 Final for 20.6 SDK.
15sep2020 Update for 20.9 SDK.
16oct2020 Update for 20.10 SDK.
18nov2020 Update for 20.11 SDK.
21jan2021 Update for 21.1 SDK.
23feb2021 Update for 21.2 SDK.
25mar2021 Update for 21.3 SDK.
13apr2021 Update for 21.4 SDK.
17jun2021 Update for 21.6 SDK.
15jul2021 Update for 21.7 SDK.
17aug2021 Update for 21.8 SDK.
27sep2021 Update for 21.9 SDK.
26Oct2021 Update for 21.10 SDK.
26Nov2021 Update for 21.11 SDK.
19Jan2022 Update for 22.1 SDK.
23Feb2022 Update for 22.2 SDK.
23Mar2022 Update for 22.3 SDK.
20Apr2022 Update for 22.4 SDK.
31May2022 Update for 22.5 SDK.
06Jul2022 Update for 22.6 SDK.
08Aug2022 Update for 22.7 SDK.
09Sep2022 Update for 22.9 SDK.
18Oct2022 Update for 22.10 SDK. API correction for retrieving stored certificates
06Dec2022 Update for 22.11 SDK. Use of App Attestation
27Jan2023 Update for 23.1 SDK.
16Mar2023 Update for 23.3 SDK.
18Apr2023 Update for 23.04 SDK. Screen recording restriction and new endpoint to get compromised device detection rules
08Jun2023 Update for 23.06 SDK.
19Jul2023 Update for 23.07 SDK. New API to fetch SDK log chunks
11Sep2023 Update for 23.09 SDK. Restriction of Print based on DLP
31Oct2023 Update for 23.10 SDK.
14Dec2023 Update for 23.12 SDK.
29Jan2024 Update for 24.01 SDK.
15May2024 Update for 24.04 SDK.
01Jul2024 Update for 24.06 SDK. Support for Tunnel with WKWebView
06Aug2024 Update for 24.07 SDK.
30Sep2024 Update for 24.09 SDK. MAG Decommission

Legal