SDK for Android

VMware Workspace ONE UEM

vmware®

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

Introduction to the VMware Workspace ONE SDK for Android

1

The Workspace ONE SDK for Android allows you to enhance your enterprise applications with MDM capabilities. You can use VMware Workspace ONE® UEM powered by AirWatch features that add a layer of security and business logic to your application.

The Workspace ONE SDK for Android has several components or library sets. The SDK component you use dictates what features you can add to your applications. For example, apps with basic MDM functionality, can use the Client SDK and omit importing the AWFramework or the AWNetworkLibrary. Whatever features used, your application must integrate the corresponding library.

SDK Library	Description	Available Features
Client SDK	The client SDK is a lightweight library for retrieving basic management and device information such as compromised status, environment info, and user information.	■ Enrollment Status ■ User Info ■ Partial SDK Profile ■ Compromised / Root Detection
AWFramework	The AWFramework includes an involved library for more advanced SDK functionality such as logging, restrictions, and encryption functions. The framework depends on the client SDK.	■ Authentication ■ Client-Side Single Sign On ■ Branding ■ Full SDK Profile Retrieval ■ Secure Storage ■ Encryption ■ Copy Restriction
AWNetworkLibrary	The AWNetworkLibrary provides advanced SDK functionality such as application proxy and tunneling and integrated authentication. It depends on the AWFramework.	■ Application Tunneling ■ NTLM and Basic Authentication ■ Certificate Authentication

This chapter includes the following topics:

Requirements

You must have the listed systems and knowledge to use the components of the VMware Workspace ONE SDK for Android v19.9.

- Android API level 16 or later
- Android Studio with the Gradle Android Build System (Gradle) 3.3.0 or later
- Workspace ONE Intelligent Hub v5.3 or later for Android or Workspace ONE 3.0 or later
- Workspace ONE UEM console v9.4 or later

You must migrate to AndroidX to use Workspace ONE SDK for Android v19.8. Refer to Migrating to AndroidX for steps to migrate to AndroidX.

Access the SDK

Download the SDK package from My Workspace ONE. You must have permissions to access this site. Speak with your Workspace ONE UEM representative for access.

SQLCipher Support and Migrating Databases

The Workspace ONE SDK for Android uses SQLCipher 4.2.0 by default. In an application upgrade scenario, the SDK handles database migration. However, if an application uses additional databases, you must migrate them.

To migrate and upgrade an existing database and preserve data and schemas, use the new default settings and use `PRAGMA cipher_migrate`. For more information, access SQLCipher 4.0.0 Release.

Emulators and Testing SDK-Built Applications

The SDK does not support testing in an emulator.

Required Libraries and Dependencies for Migration to the Latest SDK Version

Add the necessary libraries and add dependencies to Gradle when you migrate to the latest Workspace ONE SDK for Android. Also, ensure that the base classes have the latest code.

Always upgrade the dependent libraries with the SDK because the dependent libraries that are packaged with the SDK are upgraded with every new version of the SDK.

Table 1-1. Required Components by Version

Migrate From Version	Required Libraries and Dependencies
19.8	Workspace ONE SDK for Android offers a new method `getNightMode()` to support Dark Mode, also known as Night Mode. Dark Mode includes colors and style updates in the AWFramework. After migration, validate that your app's colors and styles are not having UI issues. If you need specific colors from AWFramework, and the migration to 19.9 changes those colors, you can revert the colors back. Refer to the 19.8 colors.xml file and override those colors in the app's colors.xml file. You can also revert styles back in the same way.
19.7	Workspace ONE SDK for Android v19.8 uses AndroidX support libraries. AndroidX replaces the original support library APIs with packages in the AndroidX namespace. Refer to Migrating to AndroidX for steps to migrate to AndroidX. You must migrate to AndroidX to use Workspace ONE SDK for Android v19.8.
17.x	The Workspace ONE SDK for Android does not require entries to migrate from 17.x.
16.10	Add the following entry to the build.gradle file. `android { defaultConfig { … ndk { abiFilters 'x86', 'x86_64', 'armeabi-v7a', 'arm64-v8a' } } }` The Workspace ONE SDK for Android 17.x includes updates to the AWNetworkLibrary. The updates remove the requirement to send an authentication token in an HTTP header. See Use the AWNetworkLibrary for updated requirements.

Table 1-1. Required Components by Version (continued)

Migrate From Version	Required Libraries and Dependencies
16.x	Select a migration process based on the use of a login module for initialization. ■ Login Module - To upgrade the master key manager, override the getPassword method in the application class. This override extends AWApplication to handle the upgrade. ■ No Login Module - Initialize your SDKContextManager and call the updateSDKForUpgrade() API.
Older than 16.x	■ Libraries ■ A total of 23 libraries including JAR and AAR files ■ SQLCipher library is an AAR file instead of a JAR file

```
@Override
public byte[] getPassword() {
    if
(SDKKeyManager.getSdkMasterKeyVersion(context
)
        !=
SDKKeyManager.SDK_MASTER_KEY_CURRENT_VERSION)
 {

        SDKKeyManager.newInstance(context);
    }
    return super.getPassword();
}
```

```
SDKContextHelper sdkContextHelper = new
SDKContextHelper();
SharedPreferences securePreferences =
SDKContextManager.getSDKContext().getSDKSecur
ePreferences();
int oldSDKVersion =
securePreferences.getInt(SDKSecurePreferences
Keys.CURRENT_FRAMEWORK_VERSION, 0);
SDKContextHelper.AWContextCallBack callBack
= new SDKContextHelper.AWContextCallBack() {
    @Override
    public void onSuccess(int requestCode,
Object result) {
        //success continue
    }

    @Override
    public void
onFailed(AirWatchSDKException e) {
        //failed
    }
};
try {
    sdkContextHelper.updateSDKForUpgrade(
                    0,
                    oldSDKVersion,
                    callBack);
} catch (AirWatchSDKException e){
        //handle exception
}
```

Table 1-1. Required Components by Version (continued)

Migrate From Version	Required Libraries and Dependencies
	■ Gradle Methods ■ For 16.02 – compile (name:'AWFramework 16.02',ext:'aar') ■ For 16.04 – compile (name:'AWFramework 16.04',ext:'aar') ■ For both – compile (name:'sqlcipher-3.5.2-2',ext:'aar') ■ Code If you are not using the login module for initialization, check the implementation of base classes.

Internally Deployed Applications

Upload an internal application to the Workspace ONE UEM console to establish trust between the SDK-built application and the console.

For applications that are deployed internally, either during production or testing, the system takes the following steps to establish trust.

Procedure

1 (Optional) Sign an APK file with the debug keystore of Android Studio.

This step allows the system to whitelist the app while debugging.

2 Upload the APK file to the Workspace ONE UEM console and assign an SDK profile to the application.

You must assign an SDK profile to the application in the Workspace ONE UEM console.

3 The Workspace ONE UEM console extracts the public signing key of the application.

4 The Workspace ONE UEM console whitelists the signing key with the Workspace ONE Intelligent Hub, Workspace ONE, or the Container.

5 The application calls the Workspace ONE SDK.

6 The Workspace ONE Intelligent Hub, Workspace ONE, or the Container validates the signing key by comparing it to the one uploaded in the Workspace ONE UEM console.

What to do next

After an application downloads and installs through the Workspace ONE Intelligent Hub, you can side-load the newer development versions signed with the same key.

Sideload a Debug Version of the Application for Developing

To help with developing, testing, and debugging your SDK-built applications, you can sideload them or push them from Android Studio.

The Workspace ONE UEM admin must have installed the same application with the same signature previously, with the Workspace ONE Intelligent Hub or Workspace ONE.

Procedure

1. The Workspace ONE UEM admin installs the SDK-built application as a signed APK with a debug key and pushes it to devices with either the Workspace ONE Intelligent Hub or Workspace ONE.
   - The admin must use the same signing key that you used for development.
   - The admin must publish the application as managed.
2. The developer side-loads and runs the debug versions of the application, that is signed with the same key, on the device.

   The signature of the console application and the side-loaded application must match.

The Workspace ONE Intelligent Hub or Workspace ONE trusts and runs the debug version on the device.

What to do next

If the application is uninstalled from the console or is uninstalled at the device, repeat this process to run a trusted debug version of the application.

Whitelist Publicly Deployed SDK-Built Apps

If you use SDK-built apps that are not uploaded to nor assigned through the Workspace ONE UEM console, then have the public signing key added to the Workspace ONE Intelligent Hub for Android v9.0+ by your Workspace ONE UEM support representative for whitelisting. Then, add the key value to the default SDK profiles for the Workspace ONE Intelligent Hub and the SDK-built app.

If your deployment meets outlined criteria, you do not need to perform this task.

Table 1-2. Whitelisting Criteria

Android Platform	SDK Version	Whitelist
Android Enterprise (Enterprise Play Store)	19.6 and newer	No **Note** Since you are not whitelisting the app, for security purposes, have the Workspace ONE UEM admin enable **Allow USB Debugging** and **Allow Non-Market** in the **Restrictions** payload of a device profile assigned to Android Enterprise devices.
Android Enterprise (Enterprise Play Store)	19.4 or older	Yes
Android (Legacy) (Play Store)	All SDK versions	Yes

If you deploy your SDK-built app from the Play Store, embed the app's public signing key in the Workspace ONE Intelligent Hub. This action sets trust between Workspace ONE UEM and the app.

Note Consider starting this process as early as possible, in parallel with your development process. The whitelisting process can take between two to six weeks, depending on Workspace ONE Intelligent Hub for Android and Workspace ONE SDK release cycles.

For applications that are deployed publicly through the Play Store, send the public signing key of the application to your Workspace ONE UEM support representative for whitelisting. Get the signing key value from the SDK-built application APK file. Add the signing key value as a custom setting in the SDK profile assigned to the Workspace ONE Intelligent Hub and in the SDK profile assigned to the app.

For apps that are from public independent software vendors (ISV) or for partner apps, access a list of ISV App Keys.

Procedure

1. Send the public signing key to your Workspace ONE UEM support representative for whitelisting.

   This action adds the key to releases of the Workspace ONE Intelligent Hub and the Workspace ONE SDK.

2. Extract the signing key value from the APK.

   a. Run a command-line interface application on a local machine with the APK file.

   b. Navigate to the directory containing the APK file.

   c. To display the signing key value, run the command `keytool –rfc –printcert –jarfile YourAppName.apk`.

   d. Copy and paste the key signing value into a text editor. Remove all line breaks and spaces so the value is one, long string value.

      The certificate value is the string between `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICAE-----`.

   Your Workspace ONE UEM admin adds this value to the SDK profile assigned to the Workspace ONE Intelligent Hub.

3 Have the Workspace ONE UEM admin add the key signing value to the SDK profile assigned to the Workspace ONE Intelligent Hub for Android.

   a Find the SDK profile assigned to the Workspace ONE Intelligent Hub by navigating in the Workspace ONE UEM console to **Groups & Settings > All Settings > Devices & Users > Android > Intelligent Hub Settings > SDK Profile** and note the profile assigned.

   In the next step you locate and edit this profile in the console.

   b Navigate to the **Custom Settings** payload of the SDK profile that is assigned to Workspace ONE Intelligent Hub. It is either the default profile or a custom profile.

   - Default SDK profile at **Groups & Settings > All Settings > Apps > Settings and Policies > Settings > Custom Settings**
   - A custom SDK profile at **Groups & Settings > All Settings > Apps > Settings and Policies > Profiles**, select the profile and the **Custom Settings** payload

   c To introduce the trusted key signing value in the **Custom Settings** text box, enter a code phrase .

```
{

"trustedKeys" :["key_value_1","key_value_2"]

}
```

   d Replace `["key_value_1","key_value_2"]` with the key signing value.

   Add multiple trusted keys and separate the values with a comma.

   e Save the profile.

4 Have the Workspace ONE UEM admin add the key signing value to the SDK profile assigned to the SDK-built application.

   a Find the SDK profile assigned to the SDK-built application in the Workspace ONE UEM console by navigating to **Apps & Books > Native > Public**, select the application, select **More > SDK > SDK Profile**, and note the profile assigned.

   b Navigate to the **Custom Settings** payload of the SDK profile that is assigned to the app. It is either the default profile or a custom profile.

   - Default SDK profile at **Groups & Settings > All Settings > Apps > Settings and Policies > Settings > Custom Settings**
   - A custom SDK profile at **Groups & Settings > All Settings > Apps > Settings and Policies > Profiles**, select the profile and the **Custom Settings** payload

   c To introduce the trusted key signing value in the **Custom Settings** text box, enter a code block .

```
{

"trustedKeys" :["key_value_1","key_value_2"]

}
```

d Replace `["key_value_1","key_value_2"]` with the key signing value.

Add multiple trusted keys and separate the values with a comma.

e Save the profile.

5 Update your app with the latest version of the SDK that includes the whitelisted signing key.

6 Update the device to the latest version of Workspace ONE Intelligent Hub that includes the whitelisted signing key.

7 Test the public application on a device enrolled with the Workspace ONE Intelligent Hub.

ISV App Keys

Cut and paste these app keys to the SDK profiles of the Workspace ONE Intelligent Hub and the app if you use apps that are from public independent software vendors (ISV) or partner apps.

Table 1-3. App Keys for Vendors

Vendors	App Keys
Webalo	MIIDdDCCAlygAwIBAgIETczrszANBgkqhkiG9w0BAQUFADB8 MQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEU MBIGA1UEBxMLTG9zIEFuZ2VsZXMxFTATBgNVBAoTDFdlYmFs bywgSW5jLjEUMBIGA1UECxMLRGV2ZWxvcG1lbnQxFTATBgNV BAMTDFdlYmFsbywgSW5jLjAeFw0xMTA1MTMwODI4MzVaFw0z ODA5MjgwODI4MzVaMHwxCzAJBgNVBAYTAlVTMRMwEQYDVQQI EwpDYWxpZm9ybmlhMRQwEgYDVQQHEwtMb3MgQW5nZWxlczEV MBMGA1UEChMMV2ViYWxvLCBJbmMuMRQwEgYDVQQLEwtEZXZl bG9wbWVudDEVMBMGA1UEAxMMV2ViYWxvLCBJbmMuMIIBIjAN BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAonjegLiAInGo faVq94A +lVA0OkDNjpNSTirUHw0sM4EMvyOKvgUGruYI1qKP0URiOqk xyeywMWuGXrQTYygiIaLgJFA3Az9Pgx/ oWFkYaxNMzbXZpy05EFgRRxJoEt7c5PA7Bk +syvOqm0cuob39wkM0wDK5opjqIhB6kSdwDCt4F6omjenTTz ydlWNpJxCjEqUE78F3Cydqkh8VAciIGXaco/ T9QXDZxHm4GAt0XicGBtnBEi1lGSWn0+le8NHy6e/H/ uxKZe3faiEDO8uwGxdYQRw+4hYhUsdFB +DN1XgO9RX7xgnUYuSFIXS6/950EqLVm8LBFt +vMZjOIfd7awIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQBosV L3vCn +dHqqwHSEnsOMuHjeacgB2nNNjRZ78fpL795BbM8DjLnUh1i alL32AC6t/VaVldnKoO6iFNSk+UOxn6I9wHn1ZGwLrSwO+G +uOCp1H23MVk3FuIzHeYRQc5csgFUkMsFgFWd9lkf0sRXLai 8NXEYYhZTxdG5BNixx2EbB5QqVPDaODomvT6WsCyeWzTKzd3 RcuOa9nnbJ5B5343QoEq78UxYZL+BbdaYWYx9jU5z/PUt/ ZxCQx4yBrhnBwYPUiUGtLL4zgvYEpHdK3qa7O8mJNkunFWx4 oNycFLPdzWo1q2nYL1FOCo0eXJVGUUgpShtaYieBuf6Tyydi
PrinterOn	MIICbzCCAdigAwIBAgIETYjhPzANBgkqhkiG9w0BAQUFADB8 MQswCQYDVQQGEwJDQTEQMA4GA1UECBMHT250YXJpbzESMBAG A1UEBxMJS2l0Y2hlbmVyMRYwFAYDVQQKEw1QcmludGVyT24g SW5jMRQwEgYDVQQLEwtEZXZlbG9wbWVudDEZMBcGA1UEAxMQ QW5ndXMgQ3VubmluZ2hhbTAeFw0xMTAzMjIxNzQ5NTFaFw0z ODA4MDcxNzQ5NTFaMHwxCzAJBgNVBAYTAkNBMRAwDgYDVQQI EwdPbnRhcmlvMRIwEAYDVQQHEwlLaXRjaGVuZXIxFjAUBgNV BAoTDVByaW50ZXJPbiBJbmMxFDASBgNVBAsTC0RldmVsb3Bt ZW50MRkwFwYDVQQDExBBbmd1cyBDdW5uaW5naGFtMIGfMA0G CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCRJ +Vt29LUH3x2A4lIS5xPBYBpsZculN9imd5fo/ambJigE/ a4KEhULpH7rvpYd41GgOfyQTFsVdszAjn5t8bWANvaFmZBNa YByRuVzq3XyflCodHQwLmF9rV3l44ug +e5Z9BOSJiAEtFdoAWiQjI5HzxDo +kfUtexw5QmU14cSwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBA ChFPQFlwmqiVBGECvRA79XnVZj1PGRc33FtkSYBQwNDj25NB / yisuFk01ArI8UNr5IqqidkhTjQq9EQatMveXxc5ro3SM7kWj 7VorXP9fbgGEt1L+3J0Me8ca7GjPSB1lzOq2si9dfoK3N+g +bL7OP1yeq6jEHmruHU2VhbZSuf

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
Capriza / Worksimple	MIIDbzCCAlegAwIBAgIEB65LmjANBgkqhkiG9w0BAQsFADBoMQswCQYDVQQGEwJJTDEPMA0GA1UECBMGSXNyYWVsMRUwEwYDVQQHEwxIb2QgSGFzaGFyb24xEDAOBgNVBAoTB0NhcHJpemExDDAKBgNVBAsTA1JuRDERMA8GA1UEAxMIRG9yIFR6dXIwHhcNMTQwNTE5MDc0NDI4WhcNNDExMDA0MDc0NDI4WjBoMQswCQYDVQQGEwJJTDEPMA0GA1UECBMGSXNyYWVsMRUwEwYDVQQHEwxIb2QgSGFzaGFyb24xEDAOBgNVBAoTB0NhcHJpemExDDAKBgNVBAsTA1JuRDERMA8GA1UEAxMIRG9yIFR6dXIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCF0AyIYm25y63pP5257bvWsm9QMt97LF4/CdACq9Btt39uyagI58dbc48WY4WCpQFX0tkxWkkxBlWNVSPkuAyxzAq46BJuYIT13oSEUJbvYVFtvoh9iJM9wldczPLbsMBO0KQ2iouJNU3qAIQGMBAoKW1IK9SHxW9XkHxXoew7sb9eLhnkMycHjLaJNs6HsqdXrDYQRFZCqOFdh/FI1bHHtUVzLcKsvLN09Z1nNvLe3WtiKvR7dcPmonv9khgjOTN9hNp3QGzUla87hfxNwzrvoyH3zdRBIEI4Y7jVC1foMepr7wDB3EHwkv+qx88aRSw7zxRoo86LaLybj5wrVCAfAgMBAAGjITAfMB0GA1UdDgQWBBSd7+aB2HEOxA30rE/qPU8FXTIWOzANBgkqhkiG9w0BAQsFAAOCAQEAOdnOSEXj3X0Fyypaelzc0ShBBtAZUPZkYS+FUblLxG058FG5Z3lhz1HENJFMY6GaBx9bzGJmlf8GB/IdrAGbK6aC7KTxa4CVa//QUtWkEN+8tTsbU2FjNXwhrkwCLGSYBHk5HeePPKG+ZLjib93mFuM2/q8AE2gpgijHKgag08Uvg6edR4w3wCUnFz4E+lRFo2lp9vmpsjkTkUoc+eessX2+mUJLEZqyztybtJXh52qyUxSjEvbmralLf7a5LRk20tyrMclimxBxkG3L91uTCJn4bILkklYX5cWLkxLwde7bacISiUjWrzVsYHxPE/6QuDESu1r7G+I9k+xyAZUTSg==
Varonis / DatAnywhere	MIICBTCCAW6gAwIBAgIEUVwUNDANBgkqhkiG9w0BAQUFADBHMQ8wDQYDVQQGEwZJc3JhZWwxEDAOBgNVBAoTB1Zhcm9uaXMxEDAOBgNVBAsTB1Zhcm9uaXMxEDAOBgNVBAMTB1Zhcm9uaXMwHhcNMTMwNDAzMTEzNjIwWhcNNDMwMzI3MTEzNjIwWjBHMQ8wDQYDVQQGEwZJc3JhZWwxEDAOBgNVBAoTB1Zhcm9uaXMxEDAOBgNVBAsTB1Zhcm9uaXMxEDAOBgNVBAMTB1Zhcm9uaXMwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAKG8QUrZAH8JtndsFArybDkvNjfT6HrMrytzZQGtXWkk063tIQZZm1g24UVGblbwReschTEStops6eD9WAk/gnosR+rjXiDGP6M6MyDJJRke08BuRgDFFdGZDrOVkAzBcf3ikkHVUw+6JpjtNpcC70DUyNyVeyvd4fuAlKXttYbVAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEANP3fmpRobzSONlxKB7ckmyK7PUbjmXh97Fanwtqfiw/a2MDAhkoDEG3Z7BPxfPWdjRQIxcxo3UaAUolXhszYKEtIS7t9z/G1tIlfIZtDWHgElZc243uvvHr/lMXU5OQBUPNwjOSeJivbFnYUDe0Omk+1NNFTCZQJuT9ses1n/Qs=

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
Palo Alto Networks / Traps	MIIDnTCCAoWgAwIBAgIEWn+4xjANBgkqhkiG9w0BAQsFADB/ MQswCQYDVQQGEwJVUzELMAkGA1UECBMCQ0ExFDASBgNVBAcT C1NhbnRhIENsYXJhMSAwHgYDVQQKExdQYWxvIEFsdG8gTmV0 d29ya3MsSW5jLjEOMAwGA1UECxMFVHJhcHMxGzAZBgNVBAMT ElBhbG8gQWx0byBOZXR3b3JrczAeFw0xNjA3MDYxMjQzNDha Fw00MzExMjIxMjQzNDhaMH8xCzAJBgNVBAYTAlVTMQswCQYD VQQIEwJDQTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExIDAeBgNV BAoTF1BhbG8gQWx0byBOZXR3b3JrcyxJbmMuMQ4wDAYDVQQL EwVUcmFwczEbMBkGA1UEAxMSUGFsbyBBbHRvIE5ldHdvcmtz MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgBhY 2iXNODtcvD1pv7LVOpfMhwvPJ2jNcuKNEz25pni7usgYHvGq 4fLD/ 8CYFY2tBgDEaSTZVZLabKvLOqTYM7MmT61jshA8+j2vImseW HG0dSgpgtTk696ox6XnB7QNuIeGDhtkIFqxdetiafDWTUJ70 s0RtdojegZJ7N8uOFQwmFyQTnoM/8ZrAk42mnfok8TJ/ B6JE7caYPGa9Icr6FqpfFSl927O7rEUS0JeOklthLLGQUKfW uS2FQbwgrNC5lcne81Kpi5WmeB/ olkYaA2sWXVNtUZaZKUIGsjgj/irn5+moawMyk/ ZLlWvmUP5W0h5KLwaotpksH8RniVKNQIDAQABoyEwHzAdBgN VHQ4EFgQUJdW6+XcHVipKSOhmlCh8peYHULQwDQYJKoZIhvc NAQELBQADggEBAG3YpZUE3RxDe9pak4NZZE/ H8EUBTkmyjd3xyjv7Oxo5Yn2n7Zxamyi7UIPEGH6UJGxHYag YYkUaNTFboLUYRGX6TCPbFh2klbWZNCcyx6BdDy7aDYDh63K 9PZvv/BQnNc8BEAyB8cOPyvxUL08RScJOOIj7l+Y7/ CPk13OTmcEZVXIHL8X4YkuFYVx2IhagtEBp7QywheknkPiUH 4uKOKKVktQISwcmsaU/ +JOEF0HxjF1iMyiy4tnOtmRf4baiCHoGxIEPOOUvmmNaB0Uo sZpiG9pG/P+PjD8zxpXow +x1kj4BQSdRd1YsgxYNFePMU5HrEyNaY4XaA/XwYtZRAn4=
Skycure / SEP Mobile	MIIDYDCCAkigAwIBAgIEUt6IkzANBgkqhkiG9w0BAQUFADBy MQswCQYDVQQGEwJJTDEQMA4GA1UECBMHVW5rbm93bjERMA8G A1UEBxMIVGVsIEF2aXYxFTATBgNVBAoTDFNreWN1cmUgTHRk LjEQMA4GA1UECxMHVW5rbm93bjEVMBMGA1UEAxMMU2t5Y3Vy ZSBMdGQuMB4XDTE0MDEyMTE0NDc0N1oXDTQxMDYwODE0NDc0 N1owcjELMAkGA1UEBhMCSUwxEDAOBgNVBAgTB1Vua25vd24x ETAPBgNVBAcTCFRlbCBBdml2MRUwEwYDVQQKEwxTa3ljdXJl IEx0ZC4xEDAOBgNVBAsTB1Vua25vd24xFTATBgNVBAMTDFNr eWN1cmUgTHRkLjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC AQoCggEBAJljcPl/ sxfglWKF8q3h3UiANwlfMsJbX0qAspYbF1R6wuARP1dfDMXn 545Gos1XO70cRN3mWE3LZDgkD3NJhMChO3Kn1Ys3yyx977F2 q6aLyRdqL8x62C12ydzo/gBscztW/ 1tiQCdRPDEMBkn1IUFSJ2jnBWmBXt8+m2tcPFW6mwnPEMykn pFm7JuiYS5eXsGUPchuURS89pOA08Vqpu/ r7XekhAUnFv8+BiUxOU9qlRjodW68VCItbPvNPzItQZAJai9 bRcsw +gq3fkO59knM2Mm6hyjflzWzmA3AlRwzUIFhbFevvTIngTXG ufrtFWS10CZfJm +2T1aEOOOEiycCAwEAATANBgkqhkiG9w0BAQUFAAOCAQEAZl dY29PRyjMRVpnlypXvC8r7edmdjE5n64v98GMvbDdazD7YSo 0y4GgACde1aVCxBrtoc2qYDB0+wpT6B+gpJrIkJI3H +qwfH7aP2ogcrRXlGdw5Ww5MXikByjiJUhqTJ9Lak0xypht +4wJA4EXdBUOhhZAbVsoAgDuFPNbyMtd0cxljbppfO+u2B/ GxpTuYRbbSIQT8cE4eu6/KSf+xIQ// kUqlDp4quTF4aqZBrnDJZtjHVt2QWDd/+3ZT/ 6Pe9Wh54eQiCRntVWOB/M7GRIaP2cfMofxCmKbd +iPU2l7B8L +WnMbOsZ2c973feOO8RRNNd3ycVrozAlv8psxI6Q==

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
Cogosense / bSafeMobile	MIIDzTCCArWgAwIBAgIEIw6v3zANBgkqhw0BAQsFADCBljELMAkGA1UEBhMCQ0ExGTAXBgNVBAgTEEJyaXRpc2ggQ29sdW1iaWExEzARBgNVBAcTClBvcnQgTW9vZHkxIjAgBgNVBAoTGUNvZ29zZW5zZSBUZWNobm9sb2d5IEluYy4xFDASBgNVBAsTC0VuZ2luZWVyaW5nMR0wGwYDVQQDExRDb2dvc2Vuc2UgVGVjaG5vbG9neTAeFw0xNjA1MjAwODQzMzZaFw00MzEwMDYwODQzMzZaMIGWMQswCQYDVQQGEwJDQTEZMBcGA1UECBMQQnJpdGlzaCBDb2x1bWJpYTETMBEGA1UEBxMKUG9ydCBNb29keTEiMCAGA1UEChMZQ29nb3NlbnNlIFRlY2hub2xvZ3kgSW5jLjEUMBIGA1UECxMLRW5naW5lZXJpbmcxHTAbBgNVBAMTFENvZ29zZW5zZSBUZWNobm9sb2d5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmumtaA86vRbZMoxuVWCUIzLW5+FAbXq1jkGyT6SIt+wOnL0cJ6CtizgPXAaCl2A77d1tofj8I4inp+1FPeJhSZS/UvZJK7mvWl8y0tyr1tJYtkwhYdCha/lXxkqBdAUAo+ +YRg08X9DF7sTaPZF6Hi30TVorYffJQ2vF2O9ofIHyAOIZ9D4GVImWggLJRc5FZI6iojGz0KtAxZj31HDBBbIpnTRXnfDXpuzE+jtVrbayoErSnLDp5GEtiwDv851K/3l8jj2PtdEQX1oLcAUw8Kq0lnNc1btj2TfGw+eg1h +TLQhYMjJb1DTXDV8lPz3WbRwRJJILuLri3gLZopVgBQIDAQABoyEwHzAdBgNVHQ4EFgQUZQfKgRF7ymdJSSKe2PTtSHo +64cwDQYJKoZIhvcNAQELBQADggEBADJvP8rKGPhc7soh5H5AE0cV7ECNeDPRGmz8JqxqXAjBOyqefyte7FjV7e28BBXtEw2POGxLwmdSOstxAKW4QwfzuhhXWlN1PfjI948UbFlPqbB22GtWnU7urLL5OD4pbZIrK// Z4EILJFrGN4o3xRKGC369DSrLRW1ietYDp0TC/bkpt17dcMjbpg9T3LwtRZk0vLEoCTsmOBi/Y9k37X4QRozX3oAP9zHrltA2RFJcHY8pG4RR7Mxg6B6H +ltL3EQMtK1PhPoBmWQfjeo3YxHPOZY3DBdpT/tYm/qSom26LeUTw274KHdJ5akLMsPKdVoMPKWeyTbKK2vpRPfAXaw=
9Folders / Nine Work	MIIBqTCCARKgAwIBAgIEUlIPdDANBgkqhkiG9w0BAQUFADAYMRYwFAYDVQQHEw05Rm9sZGVycyBJbmMuMCAXDTEzMTAwNzAxMzM0MFoYDzIwNjMwOTI1MDEzMzQwWjAYMRYwFAYDVQQHEw05Rm9sZGVycyBJbmMuMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCfvpIUArVlzNxWaHBV3wphJILBRx0oNs/ 6hI7ija2Zlft443IdGsLQ/ gZHCDaadPTKqh3WS2YrD4a53fGikrvFEYpGnyVVuAQgMZoHZ96fgbW2cOlVhYFtERnNEpPgyK1pLfFnehHhQXJSP43VtCvDLLG0gD6tlefzzazZHmdWBwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAIZHTlPHb3VRbaUNXXoxIjxzbrMR+YMnxOegn +HEsBbHmCocpF7yWEAIFRdIe71XpME7ECXsdsXdwboLO/ll +IQpnPBcM6Cxl0fqhM6ry2393MOeZazIIpt4KOuYjQb +bZv3ghtuq5XmmgLnMNk1+Y1tKu3a4yf2Xv8/UvxOtz9I

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
Appthority	MIIDpTCCAo2gAwIBAgIEZnL5UjANBgkqhkiG9w0BAQsFADCB gTELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWEx FjAUBgNVBAcTDVNhbiBGcmFuY2lzY28xFzAVBgNVBAoTDkFw cHRob3JpdHkgSW5jMRUwEwYDVQQLEwxBbmRyb2lkIFRlYW0x FTATBgNVBAMTDEFua2l0IERldmFuaTAgFw0xNjAxMDUxNjI4 NDBaGA8yMjg5MTAyMDE2Mjg0MFowgYExCzAJBgNVBAYTAlVT MRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1TYW4g RnJhbmNpc2NvMRcwFQYDVQQKEw5BcHB0aG9yaXR5IEluYzEV MBMGA1UECxMMQW5kcm9pZCBUZWFtMRUwEwYDVQQDEwxBbmtp dCBEZXZhbmkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK AoIBAQCiDD7a1lPf00zEOD3EgQcdsmJSHoZ+XU +Fg2KhuuAlkC0dHNafqDaXYc4fSaWgsofYDrw7Ppx3yr+J +qLLhgfYJNX751gr+ +rYKECFwMAlAOL0BwoNXMpXPj5KTw9vAwa8kbszN98Narw8r Ka5tzjJpk+URSXm3fWXbuAy3D5DY54QuX/ 2q3PLii6kl7DV7wqJOo2r0c6IMYyBOgXagPaHMWJeXyVg4e2 3jLhPpoU7k9Dv1Og5bfNLUjwEyU2dUIoZ/krN1iM1pq +iQQoYCnWVA+096kM40FA0fQ/ 4J4ZA4sZAsFcpr0yHEUyj3BPEKNCwkPC6A8ig7LaQeShOpC +bAgMBAAGjITAfMB0GA1UdDgQWBBSfnTsTxAZQ8+4OjrwNGG ijadgHzTANBgkqhkiG9w0BAQsFAAOCAQEAfj341f4vAXjp1K S9s7V7/mbzK59/CI5HNaI96MLe3T/ pIqbOUerijNurTCpr1Z7WRyghi70/ V80otVgJn356d2FfBzwcsfkYw36f44mUOEolo2XQuE28ElWL iqE7uqClyFTbD9+2QULvSpb4/V +7hygNq1faX97OoSpJFXz5igE6e8zaKc +81AuTEnHLiJHWo6o1TjAerMEnz +eIwejt8q8VBcbQ6t4SDYJMhm+u/ edOKiQBNxJ13/3KWdGqc4PoQroFwcJahKhjUZhDAu+3P +zCWqlLY4NYANSeN4LPV29sSv7ZK5ZEsOh8XOW6P/ JqLlgPOuDx45L2UTm3QrFUXQ==

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
Movius	MIIDjjCCAnagAwIBAgIEUKqV8jANBgkqhkiG9w0BAQUFADCB iDELMAkGA1UEBhMCVVMxEDAOBgNVBAgTB0dlb3JnaWExFDAS BgNVBAcTC0pvaG5zIENyZWVrMScwJQYDVQQKEx5Nb3ZpdXMg SW50ZXJhY3RpdmUgQ29ycG9yYXRpb24xEjAQBgNVBAsTCU1h cmtldGluZzEUMBIGA1UEAxMLSm9zZSBSb21lcm8wHhcNMTIx MTE5MjAyNjI2WhcNNDAwNDA2MjAyNjI2WjCBiDELMAkGA1UE BhMCVVMxEDAOBgNVBAgTB0dlb3JnaWExFDASBgNVBAcTC0pv aG5zIENyZWVrMScwJQYDVQQKEx5Nb3ZpdXMgSW50ZXJhY3Rp dmUgQ29ycG9yYXRpb24xEjAQBgNVBAsTCU1hcmtldGluZzEU MBIGA1UEAxMLSm9zZSBSb21lcm8wggEiMA0GCSqGSIb3DQEB AQUAA4IBDwAwggEKAoIBAQDiigQgehRMGOUKfZILdgHonkkA OmySVFOkzNO7Fv+oMWB +qVvlj2ZQdNqyiTkj5nhxGjArj5MobDTc9zH42anpYcfU7Ky Mj6vpMFZfpyfROUyFXdTWEL0b3lXd1QZAZwzBg5yDmklNqHH J1z8moA18PLg+X0b2W92CFxbkkacG1zJnZ6GjJkfMPfQ/ sQcayK+hXdr3CaavduDUsIdQoa3IfkiLGHw62zHmOhcPeV/ lwDwhe+lXPew/ VNapOuBZgNPnm22pLYrpPW5MMk2Oq3lcOspCZJDXlp/ zfhiE4xH91CIsKBStP5c6buQJd9cwrPpqX +fobDOzdOeAZt5Z9E25AgMBAAEwDQYJKoZIhvcNAQEFBQADg gEBAK72k97hFpHHhuTfJiRAAy/TsU54MyDawvVMCqTd/ kQDAODDA/ jQDv6U380wIG0uB5rn5UyoKbH4ZN0hizhiAAdIsIBR9B97mn l9zZGlk2x4sxECHjs/YcXkyLL8lcALVMoph +WGCPR7RzPlWbvqFOFecqk/cCNcNGLVI2Qm/ TuKMvCvdIjGlRgY2Car9w3NbQ7y4Avflv2mzuzHI4zLaxgGs 7n6Jw0jiDOruzL1NbiuldYABkM +XCLtNtTTqpeN32dFrZOH6d4cJkAjTqh2iYd +HcYDX3t6N9pA9ADAunKJbfOMgjW0rrcvSc6VkhpMH4Tk2Wy AdYdZCpFkSWQnt+M=

Table 1-3. App Keys for Vendors (continued)

Vendors	App Keys
NetDocuments	MIICzTCCAbWgAwIBAgIEXS/TXzANBgkqhkiG9w0BAQsFADAXMRUwEwYDVQQKEwxOZXREb2N1bWVudHMwHhcNMTcwNjI2MTAwOTQ1WhcNNDIwNjIwMTAwOTQ1WjAXMRUwEwYDVQQKEwxOZXREb2N1bWVudHMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDTSWa+GjVADWgH3Wj+ZGeoxVEcY4uFZTIR6f9MWjEGUNNPHf9Ljs8CUcvlu6xXmWOtrAP8U/++0kkq7izkanT9of3T6bS/4DKLVeOzISg42xQzodGIz+zZhQ2+2F7x2KlV59F7IthZPbRTn86D2YhKs6YM0aFotMnX13lpWpdPEnuN4n7CKsfKwbj4VHMkMUTAjpml+TWLtVx55BoY+XLxr26G9/osp6Q3Hr8iHGFwCop72tE5vpdMyhD7+zuW0VueFriijpGFnr5R5zTHAlgBks9HFW5tjbnXIBp44ytzFE8ju1aQ/zSZy9SgnAdJS6HEHnuwV18BgBZEDIJZJXNpAgMBAAGjITAfMB0GA1UdDgQWBBTyt4Alh+gNKyyYvk5dHy9A4PVv4TANBgkqhkiG9w0BAQsFAAOCAQEAs2DCK/12wC0bwoAECpZWfEfTorZFyM+6EkrhdY8WNdztZCyIBqfBTrtbhXQVRQ7gZta8mfoaUoNma2ZXYdzZ3RHnkY+4jVdPgIJPwlBFjiuiQlOayoc/Bqu25OAJ2f2j0ygfB/5ncg5BHuPBQRMzUXTi60/43WkOMf0MwBCBoTwwaHoZTaLCaoAOh2kkIIDnoaTN0paxC2r7Hkcr1foui4LKsO7hOM6oO8C8fJXBZdO320+NtfrXSCU0hnHT8xbYLUwqWLqb0LtOTnFZbtuWEdES38b7a06oQxd1ngYCAXNhpsd8J3AfykBhMCGoWFLUcshHIH1W/oD73QD7pd+UlA==
IBM	MIICqDCCAmagAwIBAgIETNrtMjALBgcqhkjOOAQDBQAwNzELMAkGA1UEBhMCVVMxDjAMBgNVBAsTBUxvdHVzMRgwFgYDVQQDEw9JQk0gQ29ycG9yYXRpb24wHhcNMTAxMTEwMTkwNjI2WhcNMzgwMzI4MTkwNjI2WjA3MQswCQYDVQQGEwJVUzEOMAwGA1UECxMFTG90dXMxGDAWBgNVBAMTD0lCTSBDb3Jwb3JhdGlvbjCCAbgwggEsBgcqhkjOOAQBMIIBHwKBgQD9f1OBHXUSKVLfSpwu7OTn9hG3UjzvRADDHj+AtlEmaUVdQCJR+1k9jVj6v8X1ujD2y5tVbNeBO4AdNG/yZmC3a5lQpaSfn+gEexAiwk+7qdf+t8Yb+DtX58aophUPBPuD9tPFHsMCNVQTWhaRMvZ1864rYdcq7/IiAxmd0UgBxwIVAJdgUI8VIwvMspK5gqLrhAvwWBz1AoGBAPfhoIXWmz3ey7yrXDa4V7l5lK+7+jrqgvlXTAs9B4JnUVlXjrrUWU/mcQcQgYC0SRZxI+hMKBYTt88JMozIpuE8FnqLVHyNKOCjrh4rs6Z1kW6jfwv6ITVi8ftiegEkO8yk8b6oUZCJqIPf4VrlnwaSi2ZegHtVJWQBTDv+z0kqA4GFAAKBgQCpzsGYWkP3Ab7mgl8gdEGnBH91U+6R/3UJ0lB5AyRlokcuTwzA7aMhj34tU7x2SgrBhwYYfGmlsoNx5Ad41Qr4KwAuY9/6wlgKrHSVZoO/xVHwT4pyjof/W+MC7n9FacDz+kvVBCWAXqyIMG5QK/u3dTT6ew/qH0WVGwt4JBTPFDALBgcqhkjOOAQDBQADLwAwLAIUILyh9fOFfHEOtYvYYnGVbAVhCg0CFCoLbVTdLhFwn6H1i427vvKu4Qsh

SDK Profile Refresh Interval

The SDK profile applies SDK functionality to an SDK-built app. The SDK retrieves updates from the console at varying intervals based on the app's activity status.

Purpose of the SDK Profile

You enable an SDK-built app with SDK functionality using code and settings in the Workspace ONE UEM console. The SDK profile is what carries the console configurations to the application. You configure default profiles in the **Groups & Settings > All Settings > Apps > Settings and Policies** and then select from **Security Policies**, **Settings**, or **SDK App Compliance**. You configure custom profiles in **Groups & Settings > All Settings > Apps > Settings and Policies > Profiles**. After you configure the profile, you assign it to the application when uploading it to the console.

The refresh interval is the schedule the Workspace ONE UEM console sends changes and updates to the SDK profile assigned to the app.

Refresh Interval When App is in the Background

The SDK updates the SDK default or custom profile when it has been 30 minutes since the last profile retrieval for these app activity statuses.

- The app transitions from the background to the foreground.
- The app wakes from doze mode.

You can change the time to retrieval with the API `AWSDKApplicationDelegate.getScheduleSdkFetchTime()`.

Refresh Interval When App is in the Foreground

The SDK updates the SDK default or custom profile when it has been four hours since the last profile retrieval for this app activity status.

- The app is in the foreground and remains there for an extend period.

You cannot change this value.

Libraries to Integrate the Client SDK 2

View a brief list of the libraries to import for the client SDK and view information on multidexing.

Import the Libraries

In your project file directory, ensure that the listed files are in the libs folder.

- AirWatchSDK AAR
- GSON JAR

Multidex

When including the Workspace ONE SDK, it is possible your app method count may exceed 65k due to the library dependencies. In this case, enable multidex to manage the additional DEX files and the code they contain.

To enable multidex, follow the Android Developer guidelines, which you can find at the following location (as of June 2019), http://developer.android.com/tools/building/multidex.html#mdex-gradle.

This chapter includes the following topics:

Set Up Gradle for the Client SDK

Set up Gradle in Android Studio for build automation. Find out where to create the dependencies block in your Gradle file and view sample code.

Procedure

1 Ensure that your top-level build file has a classpath pointing to Gradle 3.4.2+.

2 Add a repositories block.

```
repositories{
    flatDir{
        dirs 'libs'
    }
}
```

3 To link the SDK AAR and the appropriate support library, create the dependencies block in your app-level Gradle file like the following block.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation (name: 'AirWatchSDK-19.9',ext : 'aar')
}
```

The dependencies block looks like the following example.

```
android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"
    defaultConfig {
        //Replace your package name here
        applicationId "<packagename>"
        minSdkVersion 16
        targetSdkVersion 27
        versionCode 1
        versionName "1.0"
        multiDexEnabled true
    }
}
// Add Google repository
repositories {
...
    maven {
    url “https://maven.google.com”
    }
}
```

Implement the Client SDK Broadcast Receiver

Extend the AirWatchSDKBaseIntentService class to set the SDK within the application to receive notifications from the Workspace ONE UEM console.

The VMware Workspace ONE SDK receives commands from the Workspace ONE UEM console through the implementation of a class which extends the **AirWatchSDKBaseIntentService**.

Procedure

1 For your SDK app to listen for these commands, register the receiver in your Android Manifest file. You can do that by adding the following excerpt to your manifest.

```
<uses-permission android:name="com.airwatch.sdk.BROADCAST" />
<receiver
    android:name="com.airwatch.sdk.AirWatchSDKBroadcastReceiver"
    android:permission="com.airwatch.sdk.BROADCAST" >
    <intent-filter>
        <!-- Replace your app package name here -->
        <action android:name="<packagename>.airwatchsdk.BROADCAST" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.airwatch.intent.action.APPLICATION_CONFIGURATION_CHANGED" />
        <data android:scheme="app" android:host=<packagename> />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.PACKAGE_ADDED" />
        <action android:name="android.intent.action.PACKAGE_REMOVED" />
        <action android:name="android.intent.action.PACKAGE_REPLACED" />
        <action android:name="android.intent.action.PACKAGE_CHANGED" />
        <action android:name="android.intent.action.PACKAGE_RESTARTED" />
        <data android:scheme="package" />
    </intent-filter>
</receiver>
```

2 Create a class named **AirWatchSDKIntentService** which extends **AirWatchSDKBaseIntentService** in the app package path to receive the callback methods.

```
package com.sample.airwatchsdk;

import ...

public class AirWatchSDKIntentService extends AirWatchSDKBaseIntentService {

    @Override
    protected void onApplicationConfigurationChange(Bundle applicationConfiguration) {
    }

    @Override
    protected void onApplicationProfileReceived(Context context, String profileId,
           ApplicationProfile awAppProfile) {
    }

    @Override
    protected void onClearAppDataCommandReceived(Context context, ClearReasonCode reasonCode) {
    }

    @Override
    protected void onAnchorAppStatusReceived(Context context, AnchorAppStatus awAppStatus) {
    }
```

```
    @Override
    protected void onAnchorAppUpgrade(Context context, boolean isUpgrade) {
    }
}
```

3 Register the intent service in your manifest.

```
<service android:name="<packagepath>.AirWatchSDKIntentService" />
```

Initialize the Client SDK

Initialize the SDKManager class with the application context on a background thread.

The entry point into the client SDK is the **SDKManager** class. Applications that also integrate the AW Framework do not need explicit SDKManager initialization. If you integrate the AWFramework, skip this task.

Important It must initialize with the application context on a background thread.

The code is an example of initialization. You can also access the sample app provided with the SDK for examples of initialization.

```
new Thread(new Runnable() {
    public void run() {
       try {
            awSDKManager = SDKManager.init(getApplicationContext());
        } catch (AirWatchSDKException e) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    String reason = "AirWatch SDK Connection Problem. " +
                                    "Please make sure AirWatch MDM Hub is Installed";
                    Toast.makeText(getApplicationContext(), reason, Toast.LENGTH_LONG).show();
                }
            });
        }
    }
}).start();
```

Once initialization completes, you can use the Client SDK.

Note Reference the Javadoc for more in-depth information on what APIs are available.

Import the Libraries and Set Up Gradle for AWFramework

3

To begin integrating the AWFramework, import the needed libraries and add dependencies to the Gradle build file.

To import the custom, packaged libraries, use the libraries provided in the SDK package. These libraries are custom and packaged to work with the SDK. The SDK does not work as designed if you replace the libraries with publicly available ones.

Inside the SDK zip folder, move all the files located in the **Libs > AWFramework > Dependencies** folder into the libs folder for your application project.

Add the dependencies in your app-level Gradle build file. View the sample application for examples of an SDK file built with Gradle.

Procedure

1 Add the JAR and AAR files to the dependencies section, ensuring to change the names to match the names and versions of the library files.

```
dependencies {
implementation fileTree(dir: ‘libs’, include: [‘*.jar’])
implementation "androidx.multidex:multidex:2.0.0"
//Integrate with ClientSDK:
implementation (name:‘AirWatchSDK-19.9’, ext:‘aar’)
//Integrate with AWFramework:
implementation (name:‘ CredentialsExt-101.1.0’, ext:‘aar’)
implementation (name:‘AWFramework-19.9’, ext:‘aar’)
implementation (name:‘SCEPClient-1.0.7’, ext:‘aar’)
implementation (name:'AWComplianceLibrary-2.2.0', ext:'aar')
implementation (name:‘VisionUx-1.0.0’, ext:‘aar’)
implementation "com.google.android.gms:play-services-safetynet:16.0.0"
implementation "androidx.legacy:legacy-support-v13:1.0.0"
implementation 'androidx.appcompat:appcompat:1.1.0'
implementation "androidx.cardview:cardview:1.0.0"
implementation "androidx.recyclerview:recyclerview:1.0.0"
implementation "com.google.android.material:material:1.0.0"
implementation “androidx.appcompat:appcompat:1.1.0”
implementation("androidx.legacy:legacy-preference-v14:1.0.0") {
    exclude group: 'androidx.legacy', module: 'legacy-support-v4'
    exclude group: 'androidx.appcompat', module: 'appcompat'
    exclude group: 'androidx.annotation', module: 'annotation'
    exclude group: 'androidx.recyclerview', module: 'recyclerview'
}
implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
```

```
implementation "org.jetbrains.kotlin:kotlin-stdlib:1.2.71"
implementation "org.jetbrains.kotlin:kotlin-reflect:1.2.71"
implementation ‘org.jetbrains.kotlinx:kotlinx-coroutines-core:1.1.1’
implementation ‘org.jetbrains.kotlinx:kotlinx-coroutines-android:1.1.1’
implementation 'org.koin:koin-core:2.0.1'
implementation 'org.koin:koin-android:2.0.1'
implementation 'org.koin:koin-java:2.0.1'
implementation 'net.zetetic:android-database-sqlcipher:4.2.0@aar'
implementation 'com.mixpanel.android:mixpanel-android:4.9.8'
}
```

2 Add a packagingOptions block with these exclusions.

```
packagingOptions {
    exclude 'META-INF/LICENSE.txt'
    exclude 'META-INF/NOTICE.txt'
}
```

3 Add a dexOptions block with these values.

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

4 Set `compileSDK`, `buildTools`, and `targetSDKVersion`, all of which reside in the defaultConfig block.

```
def compileSdk = 28
def buildTools = "28.0.3"
defaultConfig {
minSdkVersion 16
targetSdkVersion 27
multiDexEnabled true
vectorDrawables.useSupportLibrary = true
    ndk {
        abiFilters 'x86', 'x86_64', 'armeabi-v7a', 'arm64-v8a'
    }
}
```

5 Sync your project with the Gradle files.

Run a Process Before Initialization, Optional

Use this optional procedure to run processes that analyze the environment into which the application is deployed. For example, run a process to determine if your application needs to start the SDK in a specific environment.

To run a process before initialization in your SDK-built application, edit the AndroidManifest.xml file and customize an activity.

Procedure

1. In the AndroidManifest.xml file, remove the launcher tag `<category android:name="android.intent.category.LAUNCHER" />`.

   You add the tag in the placeholder activity you create to run your process.

2. Create an activity and register it in the AndroidManifest.xml file.

   This activity runs the desired process.

3. Add the intent filter to the activity you just created in the manifest.

   The filter resembles the example.

   ```
   <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
   ```

4. Call `startActivity(new Intent(this, SDKSplashActivity.class))` after the process completes.

Application Delegation Options

After importing the libraries and before initialization, you need to extend an application class. Select from several paths available for extension depending on the programming language and if you can extend AWApplication.

Extend AWApplication Class

If you can extend the AWApplication class, see Initialize the AWFramework to extend it and initialize the AWFramework.

Delegate Method in Java

If you cannot extend the AWApplication class and you want to use Java, see Delegate Method for Java to extend the AWSDKApplicationDelegate method.

Delegate Method in Kotlin

If you cannot extend the AWApplication class and you want to use Kotlin, see Delegate Method for Kotlin to extend the AWSDKApplicationDelegate method.

Delegate Method for Java

Use AWSDKApplicationDelegate if your SDK-built application cannot extend AWApplication class for applications written in Java. If you can extend AWApplication class, skip this task and go to the initialization topic.

If you are using the automatic implement feature from Android Studio to implement methods, you might see `@org.jetbrains.annotations.Nullable`. If so, replace it with the `@Nullable` from the android.support.annotation.Nullable package.

Procedure

1 In the application's Application class, implement the AWSDKApplication interface.

```
public class <myApplication_class> extends <some_other_class> implements AWSDKApplication
```

2 Implement specific methods, getDelegate, onCreate, onPostCreate, and getSystemService.

```
private final AWSDKApplicationDelegate delegate = new AWSDKApplicationDelegate();

@NonNull
@Override
public AWSDKApplication getDelegate() {
    return delegate;
}

@Override
public void onCreate() {
    super.onCreate();
    this.onCreate(this);
}

@Override
public void onPostCreate() {
    //add your onCreate code here.
}

@Override
public Object getSystemService(String name) {
    return getAWSystemService(name, super.getSystemService(name));
}
```

3 You must implement all the other methods in the AWSDKApplication interface. Use the following pattern to implement the methods.

```

@Override
public <SomeReturnType> <SomeMethodName> (<some-arguments...>) {
    return delegate.<SomeMethodName>(<some-arguments...>);
}

@Override
public void <SomeMethodName> (<some-arguments...>) {
    delegate.<SomeMethodName>(<some-arguments...>);
}
```

What to do next

See Initialize the AWFramework.

Delegate Method for Kotlin

Use AWSDKApplicationDelegate if your SDK-built application cannot extend AWApplication class for applications written in Kotlin. If you can extend AWApplication class, skip this task and go to the initialization topic.

Procedure

1 In your Application class, override the listed methods in `class <yourApplicationClass>: <YourParentClass>, AWSDKApplication by AWSDKApplicationDelegate()`.

```
override fun onCreate() {
    super.onCreate()
    onCreate(this)
}

override fun getSystemService(name: String): Any? {
    return getAWSystemService(name, super.getSystemService(name))
}

 override fun onPostCreate() {
      //add your onCreate code here.
 }
```

2 Use `getDelegate()` to call AWFramework implementation for methods in the delegate from your application class.

What to do next

See Initialize the AWFramework.

Initialize the AWFramework

The SDK-built application can use application level authentication for initialization with the AWFramework. Latest versions of the SDK automatically initialize both the context and gateway so you do not have to manually initialize the VMware Tunnel.

Create a class which extends AWApplication and overrides applicable methods.

If you need alternatives to extending the AWApplication class, see Delegate Method for Java or Delegate Method for Kotlin. After you extend the AWSDKApplicationDelegate, go to step 2.

Procedure

1 Create a class that extends the AWApplication class to pass configuration keys to the login module.

2 Override getMainActivityIntent() and getMainLauncherIntent() methods in the extended class. Move your onCreate() business logic to onPostCreate().

The onSSLPinningValidationFailure() and onSSLPinningRequestFailure() callbacks are called when SSL pinning validation fails for the back-end server (denoted by "host"). If the application does not use SSL pinning, you can leave these callbacks empty.

```
public class AirWatchSDKSampleApp extends AWApplication {

    /**
    * This method must be overridden by application.
    * This method should return Intent of your application main Activity
    *
    * @return your application's main activity(Launcher Activity Intent)
    */
    @Override
    public Intent getMainLauncherIntent() {
        return new Intent(getAppContext(), SDKSplashActivity.class);
    }

    @Override
    protected Intent getMainActivityIntent() {
        Intent intent =  new Intent(getApplicationContext(), MainActivity.class);
        return intent;
    }

    @Override
    public void onPostCreate() {
         super.onPostCreate();
         // App code here
    }

    @Override
    public void onSSLPinningValidationFailure(String host, @Nullable X509Certificate
x509Certificate) {
    }

    @Override
    public void onSSLPinningRequestFailure(String host, @Nullable X509Certificate
x509Certificate) {
    }
}
```

3 If you want to enable Workspace ONE UEM functionality in the AWApplication class, override optional methods.

- `getScheduleSdkFetchTime()`- Override this method to change when the login module fetches updates to SDK settings from the Workspace ONE UEM console.
- `getKeyManager()` – Override this method to a value rather than null so that the login module initializes another key manager and not its own.

4 In the manifest header file, declare tools.

```
<?xml version  = "1.0" encoding = "utf-8"?>
<manifest xmls:android = http://schemas.android.com/apk/res/android
    package = "<your app package name>"
    xmlns:tools = "http://schemas.android.com/tools">
```

5 Declare the tools:replace flag in the application tag that is in the manifest.

```
<application
    android:name  = ".AirWatchSDKSampleApp"
    android:allowBackup = "true"
    android:icon = "@mipmap/ic_launcher"
    android:label "@string/app_name"
    android:suportRtl = "true"
    android:theme = "@style/AppTheme"
    tools:replace = "android:label">
```

6 Set the `SDKSplashActivity` as your main launching activity in the application tag.

```
<activity
    android:name="com.airwatch.login.ui.activity.SDKSplashActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

7 In the AndroidManifest, add the certificate pinning meta tags under the opening application tag.

```
    <meta-data android:name="com.airwatch.certpinning.refresh.interval" android:value="1"/>
    <meta-data android:name="com.airwatch.certpinning.refresh.interval.unit"
android:value="DAYS"/>
```

Here is an example of the launching activity and the certificate pinning code.

```
<application
    android:name=".AirWatchSDKExampleApp"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    tools:replace = "android:label">

    <meta-data android:name="com.airwatch.certpinning.refresh.interval" android:value="1"/>
    <meta-data android:name="com.airwatch.certpinning.refresh.interval.unit"
android:value="DAYS"/>

    <receiver
        android:name="com.airwatch.sdk.AirWatchSDKBroadcastReceiver"
        android:permission="com.airwatch.sdk.BROADCAST">
        <intent-filter>
            <action android:name="<packagename>.airwatchsdk.BROADCAST" />
```

```
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.PACKAGE_ADDED" />
            <action android:name="android.intent.action.PACKAGE_REMOVED" />
            <action android:name="android.intent.action.PACKAGE_REPLACED" />
            <action android:name="android.intent.action.PACKAGE_CHANGED" />
            <action android:name="android.intent.action.PACKAGE_RESTARTED" />
            <data android:scheme="package" />
        </intent-filter>
    </receiver>

    <activity
        android:name="com.airwatch.login.ui.activity.SDKSplashActivity "
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name=".MainActivity" />

</application>
```

8 Add `sdkBranding` to the application theme. The system displays this logo on the login screen. You can also use your own icon located in the mipmap directory.

```
<style name="SDKBaseTheme" parent="Theme.AppCompat.Light">
    // Replace with your own app specific resources to have branding
    <item name="awsdkSplashBrandingIcon">@drawable/awsdk_test_icon_unit_test</item>
    <item name="awsdkLoginBrandingIcon">@drawable/awsdk_test_icon_unit_test</item>
    <item name="awsdkApplicationColorPrimary">@color/color_awsdk_login_primary</item>
</style>
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

What to do next

If you need the SDK authentication, DLP, and timeout behavior, app activities should extend from `SDKBaseActivity`. These activities allow the application to handle the life cycle correctly and to manage the state of the SDK.

The SDKContext and Custom Data Encryption

To encrypt custom data and to secure storage data, use the data encryption API set to encrypt and decrypt data.

Retrieve the SDK Profile

Once the SDKContext is in the configured state, you can call the SDKContext getSDKConfiguration() method to retrieve the SDK profile. The SDK must be finished with its configuration otherwise calling getSDKConfiguration() returns with an empty value.

```
if (sdkContext.getCurrentState() == SDKContext.State.CONFIGURED) {
    String sdkProfileString = SDKContextManager.getSDKContext().
    getSDKConfiguration().toString();
}
```

Encrypt Custom Data

Once the SDKContext is in the initialized state, you can call the data encryption API set. This set of functions uses the Workspace ONE SDK's intrinsic key management framework to encrypt and decrypt any data you feed in.

Use the MasterKeyManager API set when the SDK is in an initialized or configured state. See the example of how you can encrypt and decrypt a string value.

```
if (SDKContextManager.getSDKContext().getCurrentState()!=SDKContext.State.IDLE) {
    MasterKeyManager masterKeyManager = SDKContextManager.getSDKContext().getKeyManager();
    String encryptedString = masterKeyManager.encryptAndEncodeString("HelloWorld");
    String decryptedString = masterKeyManager.decodeAndDecryptString(encryptedString);
}
```

Secure Storage Data

After the SDKContext is in the initialized state, you can call the secure storage API set. This set of functions stores key value pairs in encrypted storage.

```
if (SDKContextManager.getSDKContext().getCurrentState()!=SDKContext.State.IDLE) {
    SecurePreferences pref = SDKContextManager.getSDKContext().getSDKSecurePreferences();
    pref.edit().putString(<KEY_NAME>, <VALUE>).commit(); // to store value
    Object value = pref.getString(<KEY_NAME>, <Default_Value>);
}
```

APIs for Copy and Paste Restrictions

To use the Workspace ONE SDK copy restriction, replace the Android classes in your application to the listed Workspace ONE APIs.

Examples

If Java class `XYZ` extends `TextView{...}`, change it to extend `AWTextView{...}`.

If you override the method `onTextContextMenuItem(int id)`, do not process the listed IDs. You must call `return super.onTextContextMenuItem(id);` for the listed IDs.

- `android.R.id.cut`

- `android.R.id.copy`
- `android.R.id.paste`
- `android.R.id.shareText`

Change `<TextView>` in all Layout XML or View XML to `<com.airwatch.ui.widget.AWTextView.../>`.

APIs

Android Class	Workspace ONE SDK API
android.support.v7.widget. AppCompatEditText	com.airwatch.ui.widget. AWEditText
android.support.v7.widget. AppCompatTextView	com.airwatch.ui.widget. AWTextView
android.support.v7.widget. AppCompatAutoCompleteTextView	com.airwatch.ui.widget. AWAutoCompleteTextView
android.support.design.widget. TextInputEditText	com.airwatch.ui.widget. AWTextInputEditText
android.support.v7.widget. SearchView.SearchAutoComplete	com.airwatch.ui.widget. AWSearchAutoComplete
android.webkit.WebView	com.airwatch.ui.widget.CopyEnabledWebView

Configure Dynamic Branding in the Console

Use dynamic branding for applications that serve multiple brands with a single login page. This feature requires settings in the Workspace ONE UEM console and code changes in the application.

Request your Workspace ONE UEM admin to set the listed values in the console.

Procedure

1. In Workspace ONE UEM console navigate to **Groups & Settings > All Settings > Apps > Settings and Policies > Settings > Branding**.
2. Enable **Branding**.
3. Add a color in the **Primary Color** field.

   The application uses this color for the background of splash and loading pages and for action buttons.

What to do next

Add code to the application to complete dynamic branding. See Configure Dynamic Branding in the Code.

Configure Dynamic Branding in the Code

Change the code to add a company logo and to implement your branding scheme.

Prerequisites

Have the Workspace ONE UEM admin enable branding in the Workspace ONE UEM console. See Configure Dynamic Branding in the Console.

Procedure

1 **Company logo** - To add a company logo for dynamic branding, override **isInputLogoBrandable()** in the **AWApplication** class.

```
/**
* Returns if the company logos are brandable.
* @return true if branded logos needs to be used on the SDK UI screens, and false if not
*/
protected boolean isInputLogoBrandable() {
    return true;
}
```

2 **Branding scheme** - Add your branding scheme to **BrandingManager** and inject it into the **BrandingProvider** class.

```
public abstract class AWApplication extends MultiDexApplication implements BrandingProvider {

    private BrandingManager brandingManager;

    @Override
    public final void onCreate() {
        super.onCreate();
        brandingManager = new DefaultBrandingManager(
            SDKContextManager.getSDKContext().getSDKConfiguration(),
            new SDKDataModelImpl(getApplicationContext()),
            getApplicationContext());
    }

    @Override
    public BrandingManager getBrandingManager() {
        return brandingManager;
    }
}
```

3 Change the notification icon.

The SDK uses the notification icon specified in the AndroidManifest file. To change the notification icon, edit the icon value in the application component list of the manifest file. If you do not have a value for the notification icon in the manifest file, the SDK uses a default Workspace ONE UEM icon.

```
<application

    android:icon="@drawable/icon">
</application>
```

Configure Dark Mode

AWFramework has a new method `getNightMode()` to support Dark Mode, also known as Night Mode. Work with it in the `AWSDKApplicationDelegate` class.

The `getNightMode()` method, by default, does not support Dark Mode. To enable or disable Dark Mode, override `getNightMode()` in the `AWSDKApplicationDelegate` class and return it with one of the listed values.

- AppCompatDelegate.MODE_NIGHT_AUTO
- AppCompatDelegate.MODE_NIGHT_FOLLOW_SYSTEM
- AppCompatDelegate.MODE_NIGHT_NO
- AppCompatDelegate.MODE_NIGHT_YES

Examples

Kotlin

```
override fun getNightMode(): Int {
        return AppCompatDelegate.MODE_NIGHT_FOLLOW_SYSTEM
    }
```

Java

```
    @Override
    public int getNightMode() {
        return AppCompatDelegate.MODE_NIGHT_FOLLOW_SYSTEM;
    }
```

Code Apps to Use the SDK to Send Push Notifications with FCM

If you use Firebase Cloud Messaging (FCM), use two Workspace ONE SDK for Android APIs in your SDK-built, internal application to integrate your FCM solution with Workspace ONE UEM. This feature replaces sending push notifications with GCM.

The Workspace ONE SDK for Android handles two tasks with APIs.

- It accepts the registration token and sends it to the console.

```
/**
* API to register the Notification Token with the WS1 Console.
*/
PushNotificationManager::registerToken(token: String)
```

- It checks for and recieves notifications from the application based on the notification parameters.

```
/**
* API to notify the SDK that a message has been received.
*
* @param title: Notification Title
* @param body: Notification Message
* @param payload: (Optional) Generic container to downstream multiple actions, data, etc.
*/
PushNotificationManager::processMessage(title: String, body: String, payload: Map<String, Any>?)
```

Prerequisites

Devices must have an internal, SDK-built application deployed to them that supports push notifications for this to work. Also, your application needs to handle getting the registration token and the sending of push notifications not delegated to the SDK. The Workspace ONE SDK for Android does not handle token registration and non-delegated push notifications for the FCM model.

Procedure

1 Use the register token API to register the FCM token with the SDK.

```
FirebaseInstanceId.getInstance().getInstanceId().addOnCompleteListener(task -> {
            if (task.getResult() != null && !TextUtils.isEmpty(task.getResult().getToken())) {
                String token = task.getResult().getToken();
                regId.setText(String.format(getString(R.string.registration_id), token));

PushNotificationManager.getInstance(this.getApplicationContext()).registerToken(token);
            } else if (task.getException() != null) {
                regId.setText(String.format(getString(R.string.error_while_registering),
task.getException().getMessage()));
            }
        }
});
```

Workspace ONE SDK for Android sends the token to the Workspace ONE UEM console.

2 Use the process message API to define parameters for the Workspace ONE SDK for Android to consume the push notification. Currently, the SDK checks for command actions.

```
class FCMMessagingService : FirebaseMessagingService() {
    private val TAG = "FCMMessagingService"
    override fun onMessageReceived(message: RemoteMessage?) {
        val context = SDKContextManager.getSDKContext().context

PushNotificationManager.getInstance(context).processMessage(message?.notification?.title?: "",
                message?.notification?.body ?: "You got a message", null)
    }


    override fun onNewToken(token: String?) {
        Logger.d(TAG, "onNewToken() called with $token")
        super.onNewToken(token)
```

```
        val context = SDKContextManager.getSDKContext().context
        if (token != null)
            PushNotificationManager.getInstance(context).registerToken(token)
    }
}
```

Configure Push Notifications in the Console

Use internal, SDK-built applications to send push notifications integrated with systems like FCM.

Procedure

1. Upload the application to Workspace ONE UEM.
2. Navigate to **Apps & Books > Applications > Native > Internal** and select the application from the list view.
3. Select the **Devices** tab.
4. Select all the devices to which you want to send the notification.
5. Select **Send Message to All** or **Send**.
6. Select **Push Notification** for the **Message Type**.
7. Select the SDK-built application in the **Application Name** field the system uses to send push notifications to selected devices.
8. Enter the message in the **Message Body**, complete the rest of the procedure, and send it to devices.

Send Push Notifications Through Internal Applications with GCM and Workspace ONE UEM

If you use Google Cloud Messaging (GCM), you can send push notifications from applications uploaded to the Workspace ONE UEM console to Android devices through GCM. Devices must have an SDK-built application on it that supports push notifications.

Firebase Cloud Messaging (FCM) replaces GCM. This outlined process does not work for applications that use FCM.

Prerequisites

Google has deprecated GCM as of April 10, 2018. See Code Apps to Use the SDK to Send Push Notifications with FCM.

This feature requires Workspace ONE UEM console v9.5+.

Procedure

1 To send notifications using GCM, configure GCM and get a sender ID and an API server key.

Option	Description
Sender ID	The GCM system generates this value when you configure an GCM API project. The system uses the value to identify the package ID for the application.
API Server Key	This key gets saved on application server to authorize the server to access Google services. Enter this value when you upload the internal application to the Workspace ONE UEM console.

These two values enable GCM, Workspace ONE UEM, the client, and the application to communicate

The Client Gets a Registration Token from GCM. Call the registration API on the client to register with GCM. The client then sends the registration token to the Workspace ONE UEM console.Each client has its own registration token. The console uses the registration token to identify the client for which push notifications are sent through GCM servers.

2 Register the application with GCM using the directions at https://developers.google.com/cloud-messaging/android/client.

3 Retrieve the sender ID and code it in the application class.

4 Retrieve the API server key and enter it in the console while uploading the app.

5 Call Register API on the client side to register the device with the GCM server and to retrieve the registration token (GCM token).

6 The SDK beacon sends the token to the Workspace ONE UEM console. If you want to force sending the token immediately, stop and restart the application.

7 Upload the Application to Workspace ONE UEM.

8 Send push notifications with Workspace ONE UEM from the Workspace ONE UEM console.

a Navigate to **Apps & Books > Applications > Native > Internal** and select the application from the list view.

b Select the **Devices** tab.

c Select all the devices to which you want to send the notification.

d Select **Send Message to All** or **Send**.

e Select **Push Notification** for the **Message Type**.

f Select the SDK-built application in the **Application Name** field the system uses to send push notifications to selected devices.

g Enter the message in the **Message Body**, complete the rest of the procedure, and send it to devices.

Code Push Notifications in the Application

To use push notifications in your application and Workspace ONE UEM, code the support of push notifications in the AWFramework module in the application class.

Use the **PushNotificationContext** Interface.

The AWFramework module provides an abstract AWApplication class that runs the **PushNotificationContext** with default behaviors. Your application can override some of the methods to intercept callbacks for additional actions or if it does not use the AWApplication class.

Follow this procedure to code support for push notifications if your application does not use the AWApplication class.

Prerequisites

Google has deprecated GCM as of April 10, 2018. See Code Apps to Use the SDK to Send Push Notifications with FCM.

- Code the sender ID from GCM into the application class for push notifications to work.
- Add the API server key from GCM to the Workspace ONE UEM console when the admin uploads the application.

Procedure

1. Override the **registerForPushNotifications()** method in the **PushNotificationContext** class and call `GCMManager.registerForPushNotifications(getApplicationContext());`.

   A call to **registerForPushNotifications()** initiates the GCM registration process.
2. Override **getSenderID()** and return your application's corresponding sender ID.
3. To intercept GCM registration and GCM push notification events, use the callbacks in **IPushNotificationReceiver**.
4. If your application sends other push notifications besides those for Workspace ONE UEM, override **AWPushNotificationReceiver** implementation and call super.

   For details about specific Workspace ONE UEM actions, review **AWPushNotificationReceiver**. Actions include uploading the registration token and checking for secure messages in the console.

   If the application class overrides AWApplication and does not use push notifications for anything else, do not override **getPushNotificationReceiver()**.
5. To code that the application received a notification and to not use AWApplication's default behavior, override **onSecureMessageReceived(String message)** with the desired behavior.

   The default behavior is to display a notification in the notification bar with a Workspace ONE UEM icon.

APIs to Use Custom Certificates for Your SDK-Built Apps

The Workspace ONE SDK for Android provides APIs to retrieve custom certificate authority (CA) certificates configured in the SDK profile and to perform trust validations based on these custom CA 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 CA Certificates

```
List<X509Certificate> caCerts =
SDKContextManager.getSDKContext().getKeyStore().getCACertificates();
```

API to Perform Trust Validations

Use the previous API with the following SDK provided API to perform trust validations on any X509 certificate.

- Declaration

  `public boolean isTrusted(X509Certificate[] inputCerts, List<X509Certificate> trustedCerts)` in `CertificateUtils` class.

- Parameter Explanation

  The parameter `inputCerts` is the list of the `X509Certificate` chain that must be validated for trust.

  The parameter `trustedCerts` is the list of CA certificates to compare against. If the `inputCerts` chain, up to any of the CA certificates, is `trustCerts`, then this API returns `true`.

Build Custom X509TrustManager Objects

You can build custom X509TrustManager objects with the two APIs. You can use them with any networking libraries (`URLConnection`, `OkHttpClient`) that require TLS/SSL trust.

```
X509TrustManager customTrustManager = new X509TrustManager() {

    @Override
    public void checkServerTrusted(X509Certificate[] chain, String authType) throws
CertificateException {

        List<X509Certificate> caCerts =
SDKContextManager.getSDKContext().getKeyStore().getCACertificates();

        if(!new CertificateUtils().isTrusted(chain, caCerts)) {
            throw new CertificateException("Certificate chain not trusted");
        }
    }
};
```

API to Perform Certificate Classification

The SDK provides a new API to retrieve certificates based on the certificate use. It classifies all the certificates configured in the SDK profile into four types.

- Authentication
- CA
- Signing
- Encryption

The API classifies based on the `Key Usage` and `Extended Key Usage` attributes present in the certificates.

- Declaration

  ```
  KeyStore authCerts =
  SDKContextManager.getSDKContext().getKeyStore().getKeyStoreByUsage(SDKKeyStore.Certificate
  Usage.AUTHENTICATION_CERTIFICATE);
  ```

- Parameter Explanation

  This API returns a `java.security.KeyStore` object which contains all key-pair entries classified as authentication certificates.

Set Up Gradle and Initialize the AWNetworkLibrary

4

To integrate the AWNetworkLibrary, add the necessary dependencies to the Gradle project and initialize the AWNetworkLibrary.

Procedure

1 Set up Gradle. Add all the dependency JARS and AARS from **libs > AWNetworkLibrary > Dependencies**. For each AAR file, add an entry stating the name and EXT type.

```
dependencies {
    ... // In addition to AWFramework entries add the listed library
    implementation (name:'AWNetworkLibrary-19.9', ext:'aar')
}
```

2 Initialize the AWNetworkLibrary.

a Follow the steps outlined in Initialize the AWFramework.

b In the extended AWApplication class, override getMagCertificateEnable() and return true to fetch the certificate for the VMware Tunnel.

```
/**
* This method is overriden if your application supports fetch mag certificate during login
process.
*
* @return true if app supports fetch mag certificate.
*/
@Override
public boolean getMagCertificateEnable() {
    return true;
}
```

c Set GatewaySplashActivity as your main launching activity instead of SDKSplashActivity.

```
<activity
    android:name="com.airwatch.gateway.ui.GatewaySplashActivity"
    android:label="@string/app_name"  >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

d Extend GatewayBaseActivity at the activity level to support network features like tunneling and integrated authentication in addition to AWFramework features.

Use the AWNetworkLibrary

The AWNetworkLibrary provides SDK capabilities like application tunneling and integrated authentication using NTLM and SSL/TLS client certificates for various HTTP clients.

The AWNetworkLibrary does not require the use of Workspace ONE UEM-provided HTTP clients or WebView to tunnel requests through the VMware Tunnel Proxy. Use common HTTP clients or the default WebView. The AWNetworkLibrary also provides new APIs for NTLM or SSL/TLS integrated authentication that you can use with available HTTP client APIs. Review the tables to see what methods have changed to configure these features.

Applications using the existing provided HTTP Client and WebView classes do not require any changes.

Application Tunneling

Use Workspace ONE SDK provided HTTP clients to tunnel application traffic through the VMware Workspace ONE Tunnel-Proxy.

Important SDK-built applications that use the VMware Tunnel-Proxy feature in the AWNetworkLibrary must use the latest Workspace ONE SDK provided HTTP clients - AW wrapper classes. Use the proveded clients to overcome a platform limitation introduced in Android Q. Android Q has removed access to platform APIs used by the AWNetworkLibrary to authenticate requests from HTTP clients. Due to this removal, authentication moved to the wrapper classes provided by the AWNetworkLibrary. Authentication requires the wrapper classes.

Table 4-1. HTTP Clients for App Tunneling

HTTP Client	Notes on Use	Replaces
AWWebView	Use the provided client directly, rather than WebView.	NA
AWWebViewClient	Use the provided client to set WebViewClient.	NA

Table 4-1. HTTP Clients for App Tunneling (continued)

HTTP Client	Notes on Use	Replaces
AWURLConnection	Use the provided client to get HttpsURLConnection for certificates and VMware Tunnel authentication. URLConnection urlConnection = AWUrlConnection.openConnection(url)	AWHttpClient This client is deprecated.
AWOkHttpClient	Use the request `AWOkHttpClient.newCall(client:OkHttpClient,request:Request).execute()` to run.	NA

Integrated Authentication

Use APIs with various HTTP clients for integrated authentication using the NTLM method or SSL/TLS client certificates. These APIs eliminate the need to provide HTTP clients in some cases.

Note Developers that use the existing APIs to achieve integrated authentication functionality do not require any changes.

Note Find code samples that use the integrated authentication APIs in the IntegratedAuthActivity.java file in the sample application.

Table 4-2. Comparison of Previous and Current Requirements for Integrated Authentication - NTLM

Capability	Previous Requirements	Updated Requirement
Add support for NTLM integrated authentication for HTTP clients.	Use AWHttpClient or NTLMHttpUrlConnection as wrapper classes.	■ For Apache HttpClient register an NTLM AuthScheme and add an AWAuthInterceptor request interceptor. ■ For UrlConnection, there is no change. Use NTLMHttpUrlConnection. ■ For OkHttpClient, set an instance of AWOkHttpAuthenticator as an authenticator.
Add support for NTLM integrated authentication for WebViews.	Use AWWebView or AWWWebViewclient.	No change. Use one of the listed methods. ■ Set an instance of AWWebViewClient as the WebViewClient for the WebView. ■ Extend the AWWebViewClient class and customize it for several methods.

Table 4-3. Comparison of Previous and Current Requirements for Integrated Authentication - SSL Client Certificate

Capability	Previous Requirements	Updated Requirement
Add support for SSL client certificate authentication for HTTP clients.	Use AWHttpClient and AWUrlConnection.	Use the API called AWCertAuthUtil. It provides methods to construct an SSLContext instance with the required certificates for authentication. You can then plug it into various HTTP clients like Apache HTTP Client, URLConnection, and OKHttpClient. For example, for HttpClient, retrieve a list of KeyManagers from the API AWCertAuthUtil.getCertAuthKeyManagers(). Use this list to construct an instance of SSLContext. Obtain an instance of SSLSocketFactory from the SSLContext instance and use it in the HttpClient.
Add support for SSL client certificate authentication with WebViews.	Use AWWebViewClient.	No change. Use AWWWebViewClient. Extend the class and override unneeded behaviors.

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. Additionally, you must enable a feature flag in the SDK to retrieve SCEP certificates. To enable this feature flag, set the `<meta-data>` attribute under the `<application>` element in the application's AndroidManifest.xml.

```
<meta-data android:name="scepEnable" android:value="true"/>
```

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
- GUID
- otherName
- x400Address

APIs for a Pending Status from the SCEP Certificate Authority

Use the SCEPCertificateFetcher method to modify SCEP certificate fetches to poll the SCEP server and to refetch certificates when the server returns a pending status.

When using SCEP, some configurations set the SCEP certificate authority not to issue the certificate until the request is approved. In this scenario, the authority returns a pending status to the SDK.

The Workspace ONE SDK for Android fetches SCEP certificates when you enable integrated authentication and have SCEP configured as a certificate authority. However, if you want your application to handle the listed scenarios, you must modify code.

- You want the application to handle a pending status result for a certification fetch.
- You want the application to know the result of the fetch and to act based on the result.

Ensure the Certificate Authority server handles retry requests. The 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 the server is not configured to handle retry requests caused by the pending status, the fetch never finishes.

To handle the pending result and to poll the server to refetch, modify the SCEPCertificateFetcher method.

Procedure

1 Set the retry interval and maximum retry count.

```
/**
 * Sets the maximum number of retry attempts. Once this is elapsed, the pending status is
 * cleared, polling stops and the next fetch results in a new SCEP request for a new
```

```
 * certificate.
 */
void setMaxRetryCount(int maxRetryCount);

/**
 * Sets the retry interval between fetch attempts (in seconds).
 */
void setRetryIntervalSeconds(int retryIntervalSeconds);
```

2 Pause and resume the polling mechanism.

Use the Activity Lifecycle methods so that the polling mechanism does not run when the application is not in the foreground.

```
/**
 * Triggers/resumes scheduled polling for fetching "Pending" certificates. To be called
 * when the application is brought to foreground(in any Activity's onResume()).
 */
void triggerPolling();

/**
 * Pause polling for SCEP pending certificates. To be called when the application is
 * sent to background (in any Activity's onPause()).
 */
boolean pausePolling();

/**
 * Returns true if a SCEP certificate fetch is pending. This will be reset to false when the
 * SCEP certificate polling results in a success or failure or if maximum number of attempts
 * is exceeded.
 */
boolean isSCEPCertificatePending();
```

3 Register listeners to identify the SCEP certificate fetch result.

```
/**
 * Register an implementation of {@link SCEPCertificateFetchListener} to listen for fetch
 * results. Ensures that one instance is added only once. Unregister the listeners using
 * {@link #unregisterFetchListener(SCEPCertificateFetchListener)} when callbacks are no longer
 * required in order to prevent memory leaks of the listener implementation.
 */
void registerFetchListener(SCEPCertificateFetchListener statusListener);

/**
 * Unregister the registered listener when callbacks are no longer necessary. This ensures
 * that the listeners are not leaked.
 */
void unregisterFetchListener(SCEPCertificateFetchListener statusListener);
```

4 Obtain an instance of the SCEPCertificateFetcher using `SCEPContext.getInstance().getSCEPCertificateFetcher()`.

Example

Example of an Activity to Poll and Listen for SCEP Fetch Results

```
public class IntegratedAuthActivity extends AppBaseActivity implements View.OnClickListener{

    private static final String TAG = "IntegratedAuthActivity";

    private static final String SCEP_MAX_COUNT_KEY = "max_count";
    private static final String SCEP_RETRY_INTERVAL_KEY = "retry_interval";

    private SCEPCertificateFetchListener certStatusListener = new SCEPCertificateFetchListener() {
        @Override
        public void onResult(CertificateFetchResult certificateFetchResult) {
            handleResult(certificateFetchResult);
        }
    };

    private SCEPCertificateFetcher certificateFetcher =
SCEPContext.getInstance().getSCEPCertificateFetcher();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ntlm_and_basic_auth);

        certificateFetcher.registerFetchListener(certStatusListener);

        final SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(this);

        //These values are fetched from SharedPreferences.
        int maxCount = pref.getInt(SCEP_MAX_COUNT_KEY,
SCEPCertificateFetcher.DEFAULT_MAX_RETRY_COUNT);
        int retryInterval = pref.getInt(SCEP_RETRY_INTERVAL_KEY,
SCEPCertificateFetcher.DEFAULT_RETRY_INTERVAL_SECONDS);

        certificateFetcher.setMaxRetryCount(maxCount);
        certificateFetcher.setRetryIntervalSeconds(retryInterval);
    }

    @Override
    protected void onResume() {
        super.onResume();

        if(certificateFetcher.isSCEPCertificatePending()){
            certificateFetcher.triggerPolling();
        }
    }

    @Override
    protected void onPause() {
        super.onPause();

        pauseSCEPCertificatePolling();
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();

        certificateFetcher.unregisterFetchListener(certStatusListener);
    }

    private void handleResult(CertificateFetchResult result) {
        switch (result.getStatus()){
            case SUCCESS:
                showSnackbar("SCEP certificate fetch succeeded");
                break;
            case FAILURE:
                String errorString = getErrorString(result.getErrorCode());
                showSnackbar("SCEP certificate fetch failed. " + errorString);
                break;
            case PENDING:
                String messageString = getErrorString(result.getErrorCode());
                PendingRetryDataModel retryDataModel = result.getPendingRetryDataModel();
                String retryMessage = "Attempts remaining: " +
retryDataModel.getRetryAttemptsRemaining() +
                        ". Time remaining for next retry: " +
retryDataModel.getTimeRemainingForNextRetryAttempt();
                showSnackbar(messageString + " " + retryMessage);
                break;
        }
    }
}
```

Run SDK-Built Applications on Huawei Devices

5

End users must enable certain Huawei devices to run applications built with the Workspace ONE SDK. This action allows the SDK-built application to communicate with Workspace ONE Intelligent Hub or with Workspace ONE as part of the application's startup process.

If this setting is not enabled on certain Huawei devices, then the SDK-built application crashes when launched if Workspace ONE Intelligent Hub or Workspace ONE are not actively running.

Procedure

1. On the device, launch the Tablet Manager or Phone Manager app.
2. Select the **Auto-Launch** option and **Secondary Launch Management**.
3. Enable Workspace ONE Intelligent Hub or Workspace for **Secondary Launch**.