

VMware AirWatch Android SDK Technical Implementation Guide

Empowering your enterprise applications with MDM capabilities using the AirWatch SDK for Android

AirWatch SDK v18.2

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

Copyright © 2018 VMware, Inc. All rights reserved. This product is protected by copyright and intellectual property laws in the United States and other countries as well as by international treaties. VMware products are covered by one or more patents listed at <http://www.vmware.com/go/patents>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

Table of Contents

Chapter 1: Overview	4
Introduction to the AirWatch SDK for Android	5
Compare Components	5
Requirements	6
Migrate to the Latest AirWatch SDK for Android	6
Chapter 2: Whitelist the Signing Key in AirWatch	9
Overview	10
Internally Deployed Applications	10
Publicly Deployed Applications	10
Chapter 3: Integrate the Client SDK	11
Overview	12
Import the Libraries	12
Set up Gradle	12
Implement the Client SDK Broadcast Receiver	13
Initialize the Client SDK	14
Chapter 4: Integrate the AWFramework	16
Overview	17
Import the Libraries and Set Up Gradle	17
Initialize the AWFramework	18
Run a Process Before Initialization, Optional	21
Use the AWFramework	22
APIs for Copy and Paste Restrictions	23
Chapter 5: Integrate the AWNetworkLibrary	25
Overview	26
Set up Gradle and Initialize the AWNetworkLibrary	26
Use the AWNetworkLibrary	27
Chapter 6: MAM Features With SDK Functions	29

MAM Functionality With Settings and Policies and the AirWatch SDK	30
Assign the Default or Custom Profile	30
Set the AirWatch Agent for Android	31
Supported Settings and Policies Options By Component and AirWatch App	31
Accessing Other Documents	34

Chapter 1:

Overview

- Introduction to the AirWatch SDK for Android5
- Compare Components5
- Requirements6
- Migrate to the Latest AirWatch SDK for Android6

Introduction to the AirWatch SDK for Android

The AirWatch Software Development Kit (SDK) for Android allows you to enhance your enterprise applications with MDM capabilities. You can use AirWatch features that add a layer of security and business logic to your application.

The Android SDK has several components or library sets.

SDK Library	Description
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.
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.
AWNNetworkLibrary	The AWNetworkLibrary provides advanced SDK functionality such as application proxy and tunneling and integrated authentication. It depends on the AWFramework.

Compare Components

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 Component	Available Features
Client SDK	<ul style="list-style-type: none"> • Enrollment Status • User Info • Partial SDK Profile • Compromised / Root Detection
AWFramework	<ul style="list-style-type: none"> • Authentication • Client-Side Single Sign On • Branding • Full SDK Profile Retrieval • Secure Storage • Encryption • Copy Restriction
AWNNetworkLibrary	<ul style="list-style-type: none"> • Application Tunneling • NTLM and Basic Authentication • Certificate Authentication

Requirements

You must have the following systems and knowledge to use the components of the AirWatch SDK for Android.

- Android 4.0.3+ / Ice Cream Sandwich
- Android API level 15-26
- Android Studio with the Android Build System (Gradle) 3.3.0+
- Android Plugin for Gradle 3.0.0+
- Knowledge in Android development
- AirWatch Agent v5.3+ for Android or Workspace ONE 3.0+
- AirWatch Console v8.0+

Emulators and Testing SDK-Built Applications

The SDK does not support testing in an emulator.

Migrate to the Latest AirWatch SDK for Android

When you migrate to the latest AirWatch SDK for Android, you must add the necessary libraries and add dependencies to Gradle. Also ensure that the base classes have the latest code.

Migrate to version 18.x

The SDK for Android does not require entries to migrate from 17.x.

Migrate to version 17.x

Add the following entry to the build.gradle file.

```
android {
    defaultConfig {
        ...
        ndk {
            abiFilters "x86", "armeabi-v7a", "armeabi"
        }
    }
}
```

Updates for the AWWNetworkLibrary

The AirWatch SDK for Android 17.x removes the requirement to send an authentication token in an HTTP header. See [Use the AWWNetworkLibrary on page 27](#) for updated requirements.

Migrate version 16.x to version 16.10

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.

```
@Override
public String getPassword() {
    if (SDKKeyManager.getSdkMasterKeyVersion(context) != SDKKeyManager.SDK_MASTER_KEY_CURRENT_
VERSION) {
        SDKKeyManager.newInstance(context);
    }
    return super.getPassword();
}
```

No Login Module

Initialize your `SDKContextManager` and call the `updateSDKForUpgrade()` API.

```
try {
    new SDKContextHelper().updateSDKForUpgrade(0,
        SDKContextManager.getSDKContext().getSDKSecurePreferences().getInt
        (SDKSecurePreferencesKeys.CURRENT_FRAMEWORK_VERSION, 0),
        new SDKContextHelper.AWContextCallBack() {
            @Override
            public void onSuccess(int requestCode, Object result) {
                //success continue
            }

            @Override
            public void onFailed(AirWatchSDKException e) {
                // failed
            }

        });
} catch (AirWatchSDKException e) {
    //handle exception
}
```

Migrate version 15.11 to version 16.02 or 16.04

- Libraries
 - A total of 23 libraries including JAR and AAR files
 - SQLCipher library is an AAR file instead of JAR file

- 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

Check the implementation of base classes if you are not using the login module for initialization.



For a list of the base classes that you migrate for the latest release of the AirWatch SDK for Android, see the following AirWatch Knowledge Base article: <https://support.air-watch.com/articles/115001676868>.

Chapter 2:

Whitelist the Signing Key in AirWatch

Overview	10
Internally Deployed Applications	10
Publicly Deployed Applications	10

Overview

Before you can begin using the SDK API, you must ensure that your application signing key is whitelisted with your AirWatch back end. There are a few ways to do this depending on your deployment scenario. The AirWatch SDK for Android offers features for apps you deployed internally or apps deployed through a public app store.

Internally Deployed Applications

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

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 AirWatch Console and assign an SDK profile to the application.
You must assign an SDK profile to the application in the AirWatch Console.
3. The AirWatch Console extracts the public signing key of the application.
4. The AirWatch Console whitelists the signing key with the AirWatch Agent or the AirWatch Container.
5. The application calls the AirWatch SDK.
6. The AirWatch Agent or the AirWatch Container validates the signing key by comparing it to the one uploaded in the AirWatch Console.

Side-Load Newer Versions for Development

After an application downloads and installs through the AirWatch Agent, then you can side-load the newer development versions signed with the same key.

Publicly Deployed Applications

For applications that are deployed publicly through the Play Store, send the public signing key of the application to AirWatch for whitelisting.

Note: Contact your professional services representative for the process of whitelisting the public signing key.

The AirWatch system follows the same process as the internally deployed applications process to establish trust.

Chapter 3:

Integrate the Client SDK

Overview	12
Import the Libraries	12
Set up Gradle	12
Implement the Client SDK Broadcast Receiver	13
Initialize the Client SDK	14

Overview

Follow the listed processes to prepare and use the Client SDK.

Import the Libraries

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

- AirWatchSDK aar
- gson jar

Multi-Dexing

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

To enable multi-dexing, follow the Android Developer guidelines, which you can find at the following location (as of February 2016), <http://developer.android.com/tools/building/multidex.html#mdex-gradle>.

Set up Gradle

1. Ensure your top-level build file has a classpath pointing to Gradle 3.0.0+.
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 {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile (name: 'AirWatchSDK-18.2', ext : 'aar')
}
```

An example of the dependencies block looks like the following:

```
android {
    compileSdkVersion 26
```

```

buildToolsVersion "26.0.1"
defaultConfig {
    // Replace your package name here
    applicationId "<packagename>"
    minSdkVersion 15
    targetSdkVersion 25
    versionCode 1
    versionName "1.0"
    multiDexEnabled true
}
}
// Add google repository
repositories {
    ...
    maven {
        url "https://maven.google.com"
    }
}

```

Implement the Client SDK Broadcast Receiver

The AirWatch SDK receives notifications from the AirWatch Console through the implementation of a class which extends the **AirWatchSDKBaseIntentService**.

1. Register the receiver.

In order for your SDK app to listen to these notifications, 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>
<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. Receive the callback methods.

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 onApplicationProfileReceived(Context context,
        String profileId, ApplicationProfile awAppProfile) {

```

3. Register the intent service in your manifest.

```

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

```

Initialize the Client SDK

The entry point into the client SDK is the **SDKManager** class.

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

Note: Applications that also integrate the AW Framework do not need explicit SDKManager initialization. The AW Framework does this internally.

The code is an example 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 Agent is Installed";
            Toast.makeText(getApplicationContext(), reason,
            Toast.LENGTH_LONG).show();
        }
    });
}
}).start();

```

Once initialization completes, you can use the AirWatch Client SDK.

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



For more information on setting up the Android SDK for internal apps, see the following AirWatch Knowledge Base article: <https://support.air-watch.com/articles/115001676448>.

Chapter 4:

Integrate the AWFramework

Overview	17
Import the Libraries and Set Up Gradle	17
Initialize the AWFramework	18
Run a Process Before Initialization, Optional	21
Use the AWFramework	22
APIs for Copy and Paste Restrictions	23

Overview

Integrate the Client SDK and then follow the listed processes to prepare for and use the AWFramework.

Import the Libraries and Set Up Gradle

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

Set up Gradle

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

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.

```
def supportLibraryVersion = "26.0.2"

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:multidex:1.0.1'
    //Integrate with sdk client:
    compile (name: 'AirWatchSDK-18.2', ext: 'aar')
    //integrate with framework:
    compile (name: 'CredentialsExt-18.2', ext: 'aar')
    compile (name: 'AWFramework-18.2', ext: 'aar')
    compile 'com.google.android.gms:play-services-safetynet:11.4.2'
    compile "com.android.support:support-v13:${supportLibraryVersion}"
    compile "com.android.support:appcompat-v7:${supportLibraryVersion}"
    compile "com.android.support:cardview-v7:${supportLibraryVersion}"
    compile "com.android.support:recyclerview-v7:${supportLibraryVersion}"
    compile "com.android.support:design:${supportLibraryVersion}"
    compile "com.mixpanel.android:mixpanel-android:4.+"
    compile "com.android.support:preference-v14:${supportLibraryVersion}"
    compile 'net.zetetic:android-database-sqlcipher:3.5.7@aar'
}
```

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.

```
dexOptions {
    jumboMode = true
    preDexLibraries false
    javaMaxHeapSize "4g"
}
```

4. Set the compileSdk to 26, the build tools to 26.0.1, and the targetSdkVersion to 25, all of which reside in the defaultConfig block.

```
def compileSdk = 26
def buildTools = "26.0.1"
defaultConfig {
    minSdkVersion 15
    targetSdkVersion 25
    multiDexEnabled true
}
vectorDrawables.useSupportLibrary = true
//to force fw to be merged with app when the aar included
consumerProguardFiles file('proguard.cfg')
ndk {
    abiFilters "x86", "armeabi-v7a", "armeabi"
}
```

5. Sync your project with the Gradle files.

Initialize the AWFramework

The application can use application level authentication or not 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.

Application Level Authentication

Create a class which extends AWApplication and overrides applicable methods.

1. Create a class that extends the AWApplication class to pass configuration keys to the login module, and override the getMainActivityIntent() and getMainLauncherIntent() methods in the extended class. Move your onCreate() business logic to onPostCreate().

```

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(getApplicationContext(), SDKSplashActivity.class);
    }

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

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

```

Optional Methods to Override

Override the methods to enable AirWatch functionality in the AWApplication class.

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

2. In the manifest header file, declare tools.

```

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

```

3. 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:supportRtl = "true"
    android:theme = "@style/AppTheme"
    tools:replace = "android:label"
    >

```

4. 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>

```

See the example.

```

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

```

```

<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>

```

5. Add the 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>

```

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

Run a Process Before Initialization, Optional

To run a process before initialization in your SDK-built application, edit the AndroidManifest.xml file and customize an activity. Use this optional procedure to run processes like identifying users.

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.

Use the AWFramework

Add code and use APIs to configure AWFramework capabilities in your application.

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 AirWatch 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

Replace the Android classes in your application to the listed AirWatch APIs to use the AirWatch SDK copy restriction.

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`

You must call `return super.onTextContextMenuItem(id);` for the listed IDs.

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

APIs

Android Class	AirWatch SDK API
<code>android.support.v7.widget.AppCompatEditText</code>	<code>com.airwatch.ui.widget.AWEditText</code>
<code>android.support.v7.widget.AppCompatTextView</code>	<code>com.airwatch.ui.widget.AWTextView</code>
<code>android.support.v7.widget.AppCompatAutoCompleteTextView</code>	<code>com.airwatch.ui.widget.AWAutoCompleteTextView</code>

Android Class	AirWatch SDK API
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

Chapter 5:

Integrate the AWNetworkLibrary

Overview	26
Set up Gradle and Initialize the AWNetworkLibrary	26
Use the AWNetworkLibrary	27

Overview

AWNetworkLibrary has a dependency on AWFramework. Integrate the AWFramework and then follow the listed processes to add the network libraries.

Set up Gradle and Initialize the AWNetworkLibrary

Follow the listed process to add the AWNetworkLibrary to your project.

1. Set up Gradle.

Add all the dependency JARS and AARS from **libs > AWNetworkLibrary > Dependencies**. For each AAR file, add entry as below stating name and ext type.

```
dependencies {
... // In addition to AWFramework entries add the below library
compile (name:'AWNetworkLibrary-18.2', ext:'aar')
}
```

2. Initialize the AWNetworkLibrary.

- a. Follow the steps outlined in [Initialize the AWFramework on page 18](#).
- b. In the extended AWApplication class, override getMagCertificateEnable() and return true to fetch the certificate for the AirWatch Tunnel.

```
/**
 * This method should be override 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>
```

```

        </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 AirWatch-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 AirWatch provided HTTP Client and WebView classes do not require any changes.

Application Tunneling

Use any HTTP clients or the default WebView for tunneling application traffic through the VMware Tunnel Proxy.

Capability	Previous Requirements	Updated Requirement
Application Tunneling		
Tunnel HTTP request with the VMware Tunnel Proxy.	Use AWHttpClient, AWURLConnection, or AWOkHttpClient.	Use any HTTP client.
Tunnel WebView requests with the VMware Tunnel Proxy.	Use AWWebView.	Use the default WebView.

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 AirWatch provided-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.

Capability	Previous Requirements	Updated Requirement
Integrated Authentication - NTLM		
Add support for NTLM integrated authentication for HTTP clients.	Use AWHttpClient or NTLMURLConnection as wrapper classes.	<ul style="list-style-type: none"> For Apache HttpClient register an NTLM AuthScheme and add an AWAAuthInterceptor request interceptor. For UrlConnection, there is no change. Use NTLMURLConnection. For OkHttpClient, set an instance of AWOkHttpAuthenticator as an authenticator.
Add support for NTLM integrated authentication for WebViews.	Use AWWebView or AWWWebViewClient.	<p>No change. Use one of the listed methods.</p> <ul style="list-style-type: none"> Set an instance of AWWWebViewClient as the WebViewClient for the WebView. Extend the AWWWebViewClient class and customize it for several methods.
Integrated Authentication - SSL Client Certificate		
Add support for SSL client certificate authentication for HTTP clients.	Use AWHttpClient and AURLConnection.	<p>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.</p> <p>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.</p>
Add support for SSL client certificate authentication with WebViews.	Use AWWWebViewClient.	No change. Use AWWWebViewClient. Extend the class and override unneeded behaviors.

Chapter 6:

MAM Features With SDK Functions

- MAM Functionality With Settings and Policies and the AirWatch SDK30
- Assign the Default or Custom Profile30
- Set the AirWatch Agent for Android31
- Supported Settings and Policies Options By Component and AirWatch App31

MAM Functionality With Settings and Policies and the AirWatch SDK

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

You can apply these SDK features to applications built with the AirWatch SDK, to supported AirWatch applications, and to applications wrapped by the AirWatch app wrapping engine because the AirWatch SDK framework processes the functionality.

Types of Options for SDK Settings

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

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

Default Settings

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

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

Custom Settings

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

Assign the Default or Custom Profile

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

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

Changes to Default and Custom Profiles

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

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

Set the AirWatch Agent for Android

Configure the AirWatch Agent for Android to use the correct default profile to apply SDK functionality.

Your configurations in Settings And Policies do not work on devices if you do not set the AirWatch Agent to apply the configurations.

1. Navigate to **Groups & Settings > All Settings > Devices & Users > Android > Agent Settings**.
2. Set the **SDK Profile V2** option in the **SDK PROFILE** section to the default profile by selecting **Android Default Settings @ <Organization Group>**.
3. **Save** your settings.

Supported Settings and Policies Options By Component and AirWatch App

Use the default settings profile to apply an AirWatch SDK feature to an SDK application, an AirWatch application, or a wrapped application by setting the configurations in **Policies and Settings** and then applying the profile. View compatibility information to know what features AirWatch supports for your application.

Scope of Matrices

The data in these tables describes the behaviors and support of the specific application.

Settings and Policies Supported Options for SDK and App Wrapping

UI Label	
	Android
Passcode: Authentication Timeout	✓
Passcode: Maximum Number Of Failed Attempts	✓
Passcode: Passcode Mode Numeric	✓
Passcode: Passcode Mode Alphanumeric	✓
Passcode: Allow Simple Value	✓
Passcode: Minimum Passcode Length	✓
Passcode: Minimum Number Complex Characters	✓
Passcode: Maximum Passcode Age	✓

UI Label	
	Android
Passcode: Passcode History	✓
Passcode: Biometric Mode	✓
Username and Password: Authentication Timeout	✓
Username and Password: Maximum Number of Failed Attempts	✓
Single Sign On: Enable	✓
Integrated Authentication: Enable Kerberos	X
Integrated Authentication: Use Enrollment Credentials	✓
Integrated Authentication: Use Certificate	✓
Integrated Authentication: Use NAPPS Authentication	X
Offline Access: Enable	✓
Compromised Detection: Enable	✓
AirWatch App Tunnel: Mode	✓
AirWatch App Tunnel: URLs (Domains)	✓
Geofencing: Area	X
DLP: Bluetooth	X
DLP: Camera	✓
DLP: Composing Email	X
DLP: Copy and Paste Out	✓
DLP: Copy and Paste Into	✓
DLP: Data Backup	X
DLP: Location Services	X
DLP: Printing	X
DLP: Screenshot	✓
DLP: Third Party Keyboards	✓
DLP: Watermark	X
DLP: Limit Documents to Open Only in Approved Applications	✓

UI Label	
	Android
DLP: Allowed Applications List	✓
NAC: Cellular Connection	X
NAC: Wi-Fi Connection	X
NAC: Allowed SSIDs	X
Branding: Toolbar Color	X
Branding: Toolbar Text Color	X
Branding: Primary Color	X
Branding: Primary Text Color	X
Branding: Secondary Color	X
Branding: Secondary Text Color	X
Branding: Organization Name	X
Branding: Background Image iPhone and iPhone (Retina)	X
Branding: Background Image iPhone 5 (Retina)	X
Branding: Background Image iPad and iPad (Retina)	X
Branding: Background Small, Medium, Large, and XLarge	X
Branding: Company Logo Phone	X
Branding: Company Logo Phone High Res	X
Branding: Company Logo Tablet	X
Branding: Company Logo Tablet High Res	X
Logging: Logging Level	✓
Logging: Send Logs Over Wi-Fi	✓
Analytics: Enable	✓
Custom Settings	✓

Accessing Other Documents

While reading this documentation you may encounter references to documents that are not included here.

The quickest and easiest way to find a particular document is to navigate to https://my.air-watch.com/help/9.2/en/Content/Release_Notes/Doc_List_PDFs.htm and search for the document you need. Each release-specific document has a link to its PDF copy on AirWatch Resources.

Alternatively, you can navigate to AirWatch Resources on myAirWatch (resources.air-watch.com) and search. When searching for documentation on Resources, be sure to select your AirWatch version. You can use the filters to sort by PDF file type and AirWatch SDK v18.2.