Microsoft Intune App SDK for Android Developer Guide

> [!NOTE]
> You may wish to first read the Intune App SDK overview, which covers the current features of the SDK and describes how to prepare for integration on each supported platform.

The Microsoft Intune App SDK for Android allows you to incorporate Intune Mobile App Management (MAM) into your Android app. A MAM-enabled application is one integrated with the Intune App SDK, and allows IT administrators to deploy policies to your mobile app when the app is actively managed by Intune.

What's in the SDK

The Intune App SDK for Android is a standard Android library with no external dependencies. The SDK is composed of:

- **Microsoft.Intune.MAM.SDK.jar**: The interfaces necessary to enable MAM and interoperability with the Intune Company Portal app. Apps must specify it as an Android library reference.
- **Microsoft.Intune.MAM.SDK.Support.v4.jar**: The interfaces necessary to enable MAM in apps that leverage the Android v4 support library. Apps that need this support must reference the jar file directly.
- **Microsoft.Intune.MAM.SDK.Support.v7.jar**: The interfaces necessary to enable MAM in apps that leverage the Android v7 support library. Apps that need this support must reference the jar file directly.
- **The resource directory**: The resources (such as strings) on which the SDK relies.
- **Microsoft.Intune.MAM.SDK.aar**: The SDK components, with the exception of the Support.V4 and Support.V7 jars. This file can be used in place of the individual components if your build system supports AAR files.
- **AndroidManifest.xml**: The additional entry points and the library requirements.
- **THIRDPARTYNOTICES.TXT**: An attribution notice that acknowledges 3rd party and/or OSS code that will be compiled into your app.

Requirements

The Intune App SDK is a compiled Android project. As a result, it is largely agnostic to the version of Android the app uses for its minimum or target API versions. The SDK supports Android API 14 (Android 4.0+) to Android API 24.

The Intune App SDK for Android relies on the presence of the Company Portal app on the device for enabling MAM policies. For MAM without device enrollment, the user is **not** required to enroll the device using Company Portal.

How the Intune App SDK works

Entry Points

The Intune App SDK requires changes to an app's source code to enable Intune app management policies. This is done through the replacement of the Android base classes with equivalent Intune base classes, whose names have the prefix **MAM**. The SDK classes live between the Android base class and the app's own derived version of that class. Using an activity as an example, you end up with an inheritance hierarchy that looks like: Activity > MAMActivity > AppSpecificActivity.

For example, when **AppSpecificActivity** interacts with its parent (e.g. calling `super.onCreate()`), **MAMActivity** is the super class.

Typical Android apps have a single mode and can access the system through their Context object. Apps that have incorporated the Intune App SDK, on the other hand, have dual modes. These apps continue to access the system through the Context object, but, depending on the base Activity used, the Context object will either be provided by Android, or will intelligently multiplex between a restricted view of the system and the Android-provided Context.

When using an Android entry point that has been overridden by its MAM equivalent, the MAM version of the entry point's lifecycle must be used (with the exception of the class **MAMApplication**).

For example, when deriving from **MAMActivity**, instead of overriding `onCreate` and calling `super.onCreate`, the Activity must override `onMAMCreate` and call `super.onMAMCreate`. This allows Activity launch (among others) to be restricted by Intune in certain cases.

SDK Permissions

The Intune App SDK requires three Android system permissions on apps that integrate it:

- `android.permission.GET_ACCOUNTS` [requested at runtime if required]
- `android.permission.MANAGE_ACCOUNTS`
- `android.permission.USE_CREDENTIALS`

These permissions are required by the Azure Active Directory Authentication Library (ADAL) to perform brokered authentication. If these permissions are not granted to the app or are revoked by the user, authentication flows that require the broker (the Company Portal app) will be disabled.

Company Portal

Why is the Company Portal app required? When the Company Portal is installed onto the device and has retrieved application restriction policy for the user from the Intune service, the SDK entry points are initialized asynchronously. Initialization is only required when the process is initially created by Android. During initialization, a connection is established with Company Portal app, and policy is retrieved from the app.

> [!NOTE]
> When the Company Portal app is not present on the device, the behavior of the Intune-enabled app will not be altered, and it will behave as a normal app.

How to integrate with the Intune App SDK

As stated earlier, the SDK requires changes to the app's source code to enable app management policies. Here are the steps necessary to enable MAM in your app:

Replace classes, methods, and activities with their MAM equivalent

Android base classes must be replaced by their respective MAM equivalents. To do so, find all instances of the classes listed in the table below and replace them with the Intune App SDK equivalent.

Android base class	Intune App SDK replacement
android.app.Activity	MAMActivity
android.app.ActivityGroup	MAMActivityGroup
android.app.AliasActivity	MAMAliasActivity
android.app.Application	MAMApplication
android.app.DialogFragment	MAMDialogFragment
android.app.ExpandableListActivity	MAMExpandableListActivity
android.app.Fragment	MAMFragment
android.app.IntentService	MAMIntentService
android.app.LauncherActivity	MAMLauncherActivity
android.app.ListActivity	MAMListActivity
android.app.NativeActivity	MAMNativeActivity
android.app.PendingIntent	**MAMPendingIntent** *
android.app.Service	MAMService
android.app.TabActivity	MAMTabActivity
android.app.TaskStackBuilder	MAMTaskStackBuilder
android.app.backup.BackupAgent	MAMBackupAgent
android.app.backup.BackupAgentHelper	MAMBackupAgentHelper
android.app.backup.FileBackupHelper	MAMFileBackupHelper
android.app.backup.SharePreferencesBackupHelper	MAMSharedPreferencesBackupHelper
android.content.BroadcastReceiver	MAMBroadcastReceiver
android.content.ContentProvider	MAMContentProvider
android.os.Binder	**MAMBinder** (Only necessary if the Binder is not generated from an Android Interface Definition Language (AIDL) interface)
android.provider.DocumentsProvider	MAMDocumentsProvider
android.preference.PreferenceActivity	MAMPreferenceActivity

Microsoft.Intune.MAM.SDK.Support.v4.jar:

Android Class Intune MAM	Intune App SDK replacement
android.support.v4.app.DialogFragment	MAMDialogFragment
android.support.v4.app.FragmentActivity	MAMFragmentActivity
android.support.v4.app.Fragment	MAMFragment
android.support.v4.app.TaskStackBuilder	MAMTaskStackBuilder
android.support.v4.content.FileProvider	MAMFileProvider

Microsoft.Intune.MAM.SDK.Support.v7.jar:

Android Class	Intune App SDK replacement
android.support.v7.app.ActionBarActivity	MAMActionBarActivity

> [!NOTE]
> Remember – when using an Android entry point that has been overridden by its MAM equivalent, the MAM version of the entry point's lifecycle must be used (with the exception of the class **MAMApplication**).
>
> For example, when deriving from **MAMActivity**, instead of overriding `onCreate` and calling `super.onCreate`, the Activity must override `onMAMCreate` and call `super.onMAMCreate`. This allows Activity launch (among others) to be restricted by Intune in certain cases.

PendingIntents and renamed methods

After deriving from one of the MAM entry points, it's safe to use the Context as you would normally – starting Activities, using its **PackageManager**, etc.

PendingIntents are an exception to this rule. When calling such classes, you need to change the class name. For example, instead of `PendingIntent.get*`, `MAMPendingIntents.get*` method must be used. After this, the resultant **PendingIntents** can be used per usual.

In some cases, a method available in the Android class has been marked as final in the MAM replacement class. In this case, the MAM replacement class provides a similarly named method (generally suffixed with "MAM") which should be overridden instead. For example, instead of overriding

`ContentProvider.query`, you would override `MAMContentProvider.queryMAM`. The Java compiler should enforce the final restrictions to prevent accidental override of the original method instead of the MAM equivalent.

Logging

Logging should be initialized early to get the most value. Depending on the app, `Application.onMAMCreate` is most likely the best place to initialize logging.

To receive MAM logs in your app, create a Java Handler and add it to the **MAMLogHandlerWrapper**. This will invoke `publish()` on the application handler for every log message.

```
/**
 * Global log handler that enables fine grained PII filtering within MAM
logs.
 *
 * To start using this you should build your own log handler and add it
via
 * MAMComponents.get(MAMLogHandlerWrapper.class).addHandler(myHandler,
false);
 *
 * You may also remove the handler entirely via
 * MAMComponents.get(MAMLogHandlerWrapper.class).removeHandler(myHandler);
 */
public interface MAMLogHandlerWrapper {
    /**
     * Add a handler, PII can be toggled.
     *
     * @param handler handler to add.
 * @param wantsPII if PII is desired in the logs.
 */
    void addHandler(final Handler handler, final boolean wantsPII);

    /**
 * Remove a handler.
 *
 * @param handler handler to remove.
 */
    void removeHandler(final Handler handler);
}
```

Enable features that require app participation

There are some policies the SDK cannot implement on its own. The app can control its behavior to achieve these features using several APIs you can find in the **AppPolicy** interface included below.

```
/**
 * External facing application policies.
 */
public interface AppPolicy {

/**
 * Restrict where an app can save personal data.
 * <strong>This function is now deprecated. Please use {@link
#getIsSaveToLocationAllowed(SaveLocation, String)} instead</strong>
 * @return True if the app is allowed to save to personal data stores;
false otherwise.
 */
@Deprecated
boolean getIsSaveToPersonalAllowed();

/**
 * Check if policy prohibits saving to a content provider location.
 *
 * @param location
 *            a content URI to check
 * @return True if location is not a content URI or if policy does not
prohibit saving to the content location.
 */
boolean getIsSaveToLocationAllowed(Uri location);

/**
 * Determines if the SaveLocation passed in can be saved to by the
username associated with the cloud service.
 *
 * @param service
 *           see {@link SaveLocation}.
 * @param username
 *           the username/email associated with the cloud service being
saved to. Use null if a mapping between the AAD username and the cloud
service username does not exist or the username is not known.
 * @return true if the location can be saved to by the identity, false if
otherwise.
 */
boolean getIsSaveToLocationAllowed(SaveLocation service, String username);

/**
 * Whether the SDK PIN prompt is enabled for the app.
 *
 * @return True if the PIN is enabled. False otherwise.
 */
boolean getIsPinRequired();
```

```
    /**
     * Whether the Intune Managed Browser is required to open web links.
     * @return True if the Managed Browser is required, false otherwise
     */
    boolean getIsManagedBrowserRequired();

    /**
     * Check if policy allows Contact sync to local contact list.
     *
     * @return True if Contact sync is allowed to save to local contact list;
    false otherwise.
     */
    boolean getIsContactSyncAllowed();

    /**
     * Return the policy in string format to the app.
     *
     * @return The string representing the policy.
     */
    String toString();

    }
```

Note: `MAMComponents.get(AppPolicy.class)` will always return a non-null App Policy, even if the device or app is not under an Intune management policy.

Example: Determining if PIN is required for the app

If the app has its own PIN user experience, you might want to disable it if the IT administrator has configured the SDK to prompt for an app PIN. To determine if the IT administrator has deployed the app PIN policy to this app, for the current end user, call the following method:

```
MAMComponents.get(AppPolicy.class).getIsPinRequired();
```

Example: Determining the primary Intune user

In addition to the APIs exposed in AppPolicy, the user principal name (**UPN**) is also exposed by the `getPrimaryUser()` API defined insode the **MAMUserInfo** interface. To get the UPN, call the following:

```
MAMUserInfo info = MAMComponents.get(MAMUserInfo.class);
       if (info != null) return info.getPrimaryUser();
```

The full definition of the MAMUserInfo interface is below:

```
/**
 * External facing user informations.
 *
 */
public interface MAMUserInfo {
	/**
	 * Get the primary user name.
	 *
	 * @return the primary user name or null if the device is not
enrolled.
	 */
	String getPrimaryUser();
}
```

Example: Determining if saving to device or cloud storage is permitted

Many apps implement features that allow the end user to save files locally or to a cloud storage service. The Intune App SDK allows IT administrators to protect against data leakage by applying policy restrictions as they see fit in their organization. One of the policies that IT can control is whether the end user can save to a "personal," unmanaged data store. This includes saving to a local location, SD card, or third-party backup services.

App participation is needed to enable the feature. If your app allows saving to personal or cloud locations directly from the app, you must implement this feature to ensure that the IT administrator can control whether saving to a location is allowed. The API below lets the app know whether saving to a personal store is allowed by the current Intune administrator's policy. The app can then enforce the policy, since it is aware of personal data store available to the end user through the app.

To determine if the policy is enforced, make the following call:

```
MAMComponents.get(AppPolicy.class).getIsSaveToLocationAllowed(
SaveLocation service, String username);
```

… where `service` is one of the following SaveLocations:

- SaveLocation.ONEDRIVE_FOR_BUSINESS
- SaveLocation.SHAREPOINT
- SaveLocation.BOX
- SaveLocation.DROPBOX
- SaveLocation.GOOGLE_DRIVE
- SaveLocation.LOCAL

- SaveLocation.OTHER

The previous method of determining whether a user's policy allowed them to save data to various locations was `getIsSaveToPersonalAllowed()` within the same **AppPolicy** class. This function is now **deprecated** and should not be used, the following invocation is equivalent to `getIsSaveToPersonalAllowed()`:

```
MAMComponents.get(AppPolicy.class).getIsSaveToLocationAllowed(SaveLocation
.LOCAL, userNameInQuestion);
```

Notes:

- `MAMComponents.get(AppPolicy.class)` will always return a non-null AppPolicy, even if the device or app is not under Intune management.
- Use SaveLocation.OTHER if the location in question is not listed in the SaveLocations enum.

Registering for notifications from the SDK

Overview

The Intune App SDK allows your app to control the behavior of certain policies, such as selective wipe, when they are deployed by the IT administrator. When an IT administrator deploys such a policy, the Intune service sends down a notification to the SDK.

Your app must register for notifications from the SDK by creating a **MAMNotificationReceiver** and registering it with **MAMNotificationReceiverRegistry**. This is done by providing the receiver and the type of notification desired in `App.onCreate`, as the example below illustrates:

```
@Override
public void onCreate() {
    super.onCreate();
    MAMComponents.get(MAMNotificationReceiverRegistry.class)
        .registerReceiver(
            new ToastNotificationReceiver(),
            MAMNotificationType.WIPE_USER_DATA);
    }
```

MAMNotificationReceiver

MAMNotificationReceiver simply receives notifications from the Intune service. Some notifications are handled by the SDK directly, while others require the app's participation. An app **must** return either true or false from a notification. It must always return true unless some action it tried to take as a result of the notification failed.

- This failure may be reported to the Intune service. An example of a scenario to report is if the app fails to wipe user data after the IT administrator initiates a wipe.

Note: It is safe to block in `MAMNotificationReceiver.onReceive` because its callback is not running on the UI thread.

The **MAMNotificationReceiver** interface is included below as defined in the SDK:

```
/**
 * The SDK is signaling that a WG event has occurred.
 *
 */
public interface MAMNotificationReceiver {

    /**
     * A notification was received.
     *
     * @param notification
     *            The notification that was received.
     * @return The receiver should return true if it handled the
     *   notification without error (or if it decided to ignore the
     *   notification). If the receiver tried to take some action in
     *   response to the notification but failed to complete that
     *   action it should return false.
     */
    boolean onReceive(MAMNotification notification);
}
```

Types of notifications

The following notifications are sent to the app and some of them may require app participation:

- **WIPE_USER_DATA**: This notification is sent in a **MAMUserNotification** class. When this notification is received, the app is expected to delete all data associated with the "corporate" identity passed with the **MAMUserNotification**. This notification is currently sent during Intune service un-enrollment. The user's primary name is typically specified during the enrollment process. If you register for this notification, your app must ensure that all the user's data has been deleted. If you don't register for it, the default selective wipe behavior will be performed.

- **WIPE_USER_AUXILIARY_DATA**: Apps can register for this notification if they'd like the Intune App SDK to perform the default selective wipe behavior, but would still like to remove some auxiliary

data when the wipe occurs.

- **REFRESH_POLICY**: This notification is sent in a **MAMUserNotification**. When this notification is received, any cached Intune policy must be invalidated and updated. This is generally handled by the SDK; however, it should be handled by the app if the policy is used in any persistent way.

Note: An app should never register for both WIPE_USER_DATA and WIPE_USER_AUXILIARY_DATA.

Configure Azure Active Directory Authentication Libraries (ADAL)

First, please read the ADAL integration guidelines found in the GitHub repo for ADAL.

The SDK relies on ADAL for its authentication and conditional launch scenarios, which require apps to be configured with Azure Active Directory. The configuration values are communicated to the SDK via AndroidManifest metadata.

To configure your app and enable proper authentication, add the following to the app node in AndroidManifest.xml. Some of these configurations are only required if your app uses ADAL for authentication in general; in that case, you will need the specific values your app uses to register itself with AAD. This is done to ensure that the end user does not get prompted for authentication twice, due to AAD recognizing two separate registration values: one from the app and one from the SDK.

```
<meta-data
    android:name="com.microsoft.intune.mam.aad.Authority"
    android:value="https://AAD authority/" />
<meta-data
    android:name="com.microsoft.intune.mam.aad.ClientID"
    android:value="your-client-ID-GUID" />
<meta-data
    android:name="com.microsoft.intune.mam.aad.NonBrokerRedirectURI"
    android:value="your-redirect-URI" />
<meta-data
    android:name="com.microsoft.intune.mam.aad.SkipBroker"
    android:value="[true | false]" />
```

ADAL metadata

- **Authority** is the current AAD authority in use. If present, you should use your own environment where AAD accounts have been configured. If this value is absent, an Intune default is used.
- **ClientID** is the AAD ClientID to be used. You should use your own app's ClientID if it is registered

with Azure AD. If this value is absent, an Intune default is used.

- **NonBrokerRedirectURI** is the AAD redirect URI to use in broker-less cases. If none is specified, a default value of `urn:ietf:wg:oauth:2.0:oob` is used.
    - If you have an AAD ClientID, you must set the value of NonBrokerRedirectURI as an acceptable redirect URI for that ClientID. For the Intune default ClientID, `urn:ietf:wg:oauth:2.0:oob` is already configured correctly.
- **SkipBroker** is used in case the ClientID has not been configured to use the broker redirect URI. The default value is "false."
    - For apps that **do not integrate ADAL** and **do not want to participate in device-wide brokered authentication/SSO**, this should be set to "true." When this value is "true," the only redirect URI that will be used is NonBrokerRedirectURI.
    - For apps that do support device-wide SSO brokering, this should be "false." When the value is "false," the SDK will select a broker between the result of `com.microsoft.aad.adal.AuthenticationContext.getRedirectUriForBroker()` and NonBrokerRedirectURI, based on the availability of the broker on the system. In general, the broker will be available from the COmpany Portal.

Common ADAL configurations

The following are common ways an app can be configured with ADAL. Find your app's configuration and make sure to set the ADAL metadata parameters (explained above) to the necessary values.

1. **App does not integrate ADAL:**

Required ADAL parameter	Value
Authority	Desired environment where AAD accounts have been configured
SkipBroker	True

2. **App integrates ADAL:**

Required ADAL parameter	Value
Authority	Desired environment where AAD accounts have been configured
ClientID	The app's ClientID (generated by Azure AD when the app is registered)
NonBrokerRedirectURI	A valid redirect URI for the app, or `urn:ietf:wg:oauth:2.0:oob` by default. Make sure to configure the value as an acceptable redirect URI for

Required ADAL parameter	your app's ClientID. **Value**
SkipBroker	False

3. **App integrates ADAL but does not support brokered authentication/device-wide SSO:**

Required ADAL parameter	**Value**
Authority	Desired environment where AAD accounts have been configured
ClientID	The app's ClientID (generated by Azure AD when the app is registered)
NonBrokerRedirectURI	A valid redirect URI for the app, or `urn:ietf:wg:oauth:2.0:oob` by default. Make sure to configure the value as an acceptable redirect URI for your app's ClientID.
SkipBroker	**True**

Multi-Identity (optional)

Overview

By default, the Intune App SDK will apply policy to the app as a whole. Multi-Identity is an optional Intune MAM feature which can be enabled to allow policy to be applied on a per-identity level. This requires significantly more app participation than other MAM features.

The app **must** inform the SDK when it intends to change the active identity, the SDK will also notify the app when an identity change is required. Currently, only one Intune managed identity is supported. Once the user enrolls the device or the app, the SDK registers this identity and considers it the primary Intune managed identity. Other users in the app will be treated as unmanaged, with unrestricted policy settings.

Note that an identity is simply defined as a string. Identities are **case-insensitive**, and requests to the SDK for an identity may not return the same casing that was originally used when setting the identity.

Enabling Multi-Identity

By default, all apps are considered to be single-identity apps. You can declare an app to be multi-identity

aware by placing the following metadata in AndroidManifest.xml.

```
<meta-data
    android:name="com.microsoft.intune.mam.MAMMultiIdentity"
    android:value="true" />
```

Setting the Identity

Developers can set the identity of the app on the following levels in descending priority:

1. Thread level
2. Context (generally Activity) level
3. Process level

An identity set at the thread level supersedes an identity set at the Context level, which supersedes an identity set at the process level. An identity set on a Context is only used in appropriate associated scenarios File IO operations, for example, do not have an associated Context. The following methods in **MAMPolicyManager** may be used to set the identity and retrieve the identity values previously set.

```
public static void setUIPolicyIdentity(final Context context, final String
identity, final MAMSetUIIdentityCallback mamSetUIIdentityCallback);

public static String getUIPolicyIdentity(final Context context);

public static MAMIdentitySwitchResult setProcessIdentity(final String
identity);

public static String getProcessIdentity();

public static MAMIdentitySwitchResult setCurrentThreadIdentity(final
String identity);

public static String getCurrentThreadIdentity();

/**
 * Get the currently applicable app policy. Same as
 * MAMComponents.get(AppPolicy.class). This method does
 * not take the context identity into account.
 */
public static AppPolicy getPolicy();

/**
 * Get the currently applicable app policy, taking the context
 * identity into account.
 */
```

```
public static AppPolicy getPolicy(final Context context);


public static AppPolicy getPolicyForIdentity(final String identity);

public static boolean getIsIdentityManaged(final String identity);
```

Important notes:

- You can clear the identity of the app by setting it to null.
- The empty string may be used as an identity which will never have policy restrictions.
- All the methods used to set the identity report back result values via `MAMIdentitySwitchResult`. There are four values that can be returned:

Return value	Scenario
SUCCEEDED	The identity change was successful.
NOT_ALLOWED	The identity change is not allowed. This occurs if an attempt is made to switch to a different managed user belonging to the same organization as the enrolled user. It also occurs if an attempt is made to set the UI (Context) identity when a different identity is set on the current thread.
CANCELLED	The user cancelled the identity change, generally by pressing the back button on a PIN or authentication prompt.
FAILED	The identity change failed for an unspecified reason.

In the case of setting a Context identity, the result is reported asynchronously. If the Context is an Activity, the SDK doesn't know if the identity change succeeded until after conditional launch is performed – which may require the user to enter a PIN or corporate credentials. The app is expected to implement a `MAMSetUIIdentityCallback` to receive this result, you can pass null for this parameter.

```
public interface MAMSetUIIdentityCallback {
    void notifyIdentityResult(MAMIdentitySwitchResult
identitySwitchResult);
}
```

You can also set the identity of an activity directly through a method in **MAMActivity** instead of calling `MAMPolicyManager.setUIPolicyIdentity`. Use following method to do so:

```
public final void switchMAMIdentity(final String newIdentity);
```

You can also override a method in **MAMActivity** if you want the app to be notified of the result of attempts to change the identity of that activity.

```
public void onSwitchMAMIdentityComplete(final MAMIdentitySwitchResult
result);
```

Note: Switching the identity may require recreating the activity. In this case, the `onSwitchMAMIdentityComplete` callback will be delivered to the new instance of the activity.

Implicit Identity Changes

In addition to the app's ability to set the identity, a thread or a context's identity may change based on data ingress from another MAM-enlightened app (an app that has also integrated to Intune App SDK).

Examples:

1. If an activity is launched from an **Intent** sent by another MAM app, the activity's identity will be set based on the effective identity in the other app at the point the Intent was sent.
2. For services, the thread identity will be set similarly for the duration of an `onStart` or `onBind` call. Calls into the `Binder` returned from `onBind` will also temporarily set the thread identity.
3. Calls into a `ContentProvider` will similarly set the thread identity for their duration.

In addition, user interaction with an activity may cause an implicit identity switch.

Example: A user canceling out of an authorization prompt during `Resume` will result in an implicit switch to an empty identity.

The app is given an opportunity to be made aware of these changes, and, if it must, the app can forbid them. **MAMService** and **MAMContentProvider** expose the following method that subclasses may override:

```
public void onMAMIdentitySwitchRequired(final String identity,
    final AppIdentitySwitchResultCallback callback);
```

In the **MAMActivity** class , an additional parameter is present in the method:

```
public void onMAMIdentitySwitchRequired(final String identity,
    final AppIdentitySwitchReason reason,
    final AppIdentitySwitchResultCallback callback);
```

- The `AppIdentitySwitchReason` captures the source of the implicit switch, and can accept the values CREATE, RESUME_CANCELLED, and NEW_INTENT. The RESUME_CANCELLED reason is

used when activity resume causes PIN, authentication, or other compliance UI to be displayed and the user attempts to cancel out of that UI, generally though use of the back button.

- The `AppIdentitySwitchResultCallback` is as follows:

```
public interface AppIdentitySwitchResultCallback {
    /**
     * @param result
     *            whether the identity switch can proceed.
     */
    void reportIdentitySwitchResult(AppIdentitySwitchResult result);
}
```

  Where `AppIdentitySwitchResult` is either SUCCESS or FAILURE.

The method `onMAMIdentitySwitchRequired` is called for all implicit identity changes except for those made through a Binder returned from `MAMService.onMAMBind`. The default implementations of `onMAMIdentitySwitchRequired` immediately call:

- `reportIdentitySwitchResult(FAILURE)` when the reason is RESUME_CANCELLED.
- `reportIdentitySwitchResult(SUCCESS)` in all other cases.

It is not expected that most apps will need to block or delay an identity switch in a different manner, but if an app needs to do so, the following points must be considered:

- If an identity switch is blocked, the result is the same as if `Receive` sharing settings had prohibited the data ingress.
- If a Service is running on the main thread, `reportIdentitySwitchResult` **must** be called synchronously or the UI thread will hang.
- For **Activity** creation, `onMAMIdentitySwitchRequired` will be called before `onMAMCreate`. If the app must show UI to determine whether to allow the identity switch, that UI must be shown using a *different* activity.
- In an **Activity**, when a switch to the empty identity is requested with the reason as RESUME_CANCELLED, the app must modify the resumed activity to display data consistent with that identity switch. If this is not possible, the app should refuse the switch, and the user will be asked again to comply with policy for the resuming identity (e.g. by being presented with the app PIN entry screen).

Important note: A multi-identity app will always receive incoming data from both managed and unmanaged apps. It is the responsibility of the app to treat data from managed identities in a managed manner.

If a requested identity is managed (use `MAMPolicyManager.getIsIdentityManaged` to check), but the app is not able to use that account (e.g. because accounts, such as email accounts, must be set up in the app first) then the identity switch should be refused.

File Protection

Every file has an identity associated with it at the time of creation, based on thread and process identity. This identity will be used for both file encryption and selective wipe. Only files whose identity is managed and has policy requiring encryption will be encrypted. The SDK's default selective functionality wipe will only wipe files associated with the managed identity for which a wipe has been requested. The app may query or change a file's identity using the **MAMFileProtectionManager** class.

```
public final class MAMFileProtectionManager {

    /**
     * Protect a file. This will synchronously trigger whatever protection
is required for the file, and will tag the file for
     * future protection changes.
     *
     * @param identity
     *            Identity to set.
     * @param file
     *            File to protect.
     * @throws IOException
     *             If the file cannot be changed.
     */
    public static void protect(final File file, final String identity)
throws IOException;

    /**
     * Get the protection info on a file.
     *
     * @param file
     *            File or directory to get information on.
     * @return File protection info, or null if there is no protection
info.
     * @throws IOException
     *             If the file cannot be read or opened.
     */
    public static MAMFileProtectionInfo getProtectionInfo(final File file)
throws IOException;

    /**
     * Get the protection info on a file.
     *
     * @param file
     *            File to get information on.
     * @return File protection info, or null if there is no protection
info.
     * @throws IOException
     *             If the file cannot be read or opened.
```

```
     */
    public static MAMFileProtectionInfo getProtectionInfo(final
ParcelFileDescriptor file) throws IOException;

}

public interface MAMFileProtectionInfo {
String getIdentity();
}
```

Important note: File identity tagging is sensitive to offline mode. The following points should be taken into account:

- If the Company Portal is not installed, files cannot be identity-tagged.
- If the Company Portal is installed, but the app does not have Intune MAM policy, files cannot be reliably tagged with identity.
- When file identity tagging becomes available, all previously created files are treated as personal/unmanaged (belonging to the empty-string identity) unless the app was previously installed as a single-identity managed app in which case they are treated as belonging to the enrolled user.

Directory Protection

Directories may be protected using the same `protect` method used to protect files. Please note that directory protection applies recursively to all files and subdirectories contained in the directory, and to new files created within the directory. Because directory protection is applied recursively, the `protect` call can take some time to complete for very large directories. For that reason, apps applying protection to a directory that contains a large number of files might wish to run `protect` asynchronously on a background thread.

Data Protection

It is not possible to tag a file as belonging to multiple identities. Apps that must store data belonging to different users in the same file can do so manually, using the features provided by **MAMDataProtectionManager**. This allows the app to encrypt data and tie it to a particular user. The encrypted data is suitable for storing to disk in a file. You can query the data associated with the identity and the data can be unecrypted later.

```
public final class MAMDataProtectionManager {
    /**
     * Protect a stream. This will return a stream containing the
protected
 * input.
```

```
     *
     * @param identity
     *            Identity to set.
     * @param input
     *            Input data to protect, read sequentially. This function
     *           will change the position of the stream but may not have
     *            read the entire stream by the time it returns. The
     *           returned stream will wrap this one. Calls to read on the
     *            returned stream may cause further reads on the original
     *           input stream. Callers should not expect to read directly
     *            from the input stream after passing it to this method.
     *           Calling close on the returned stream will close this one.
     * @return Protected input data.
     * @throws IOException
     *             If the data could not be protected
     */
    public static InputStream protect(final InputStream input, final
String identity);

    /**
     * Protect a byte array. This will return protected bytes.
     *
     * @param identity
     *            Identity to set.
     * @param input
     *            Input data to protect.
     * @return Protected input data.
     * @throws IOException
     *             If the data could not be protected
     */
    public static byte[] protect(final byte[] input, final String
identity) throws IOException;

    /**
     * Unprotect a stream. This will return a stream containing the
     * unprotected input.
     *
     * @param input
     *            Input data to protect, read sequentially.
     * @return Protected input data.
     * @throws IOException
     *             If the data could not be unprotected
     */
    public static InputStream unprotect(final InputStream input) throws
IOException;

    /**
     * Unprotect a byte array. This will return unprotected bytes.
     *
     * @param input
```

```
     *            Input data to protect.
     * @return Protected input data.
     * @throws IOException
     *             If the data could not be unprotected
     */
    public static byte[] unprotect(final byte[] input) throws IOException;

    /**
     * Get the protection info on a stream.
     *
     * @param input
     *            Input stream to get information on. Either this input
 *            stream must have been returned by a previous call to
     *            protect OR input.markSupported() must return true.
 *            Otherwise it will be impossible to get protection info
 *            without advancing the stream position. The stream must be
 *            positioned at the beginning of the protected data.
     * @return Data protection info, or null if there is no protection
 *            info.
     * @throws IOException
     *             If the input cannot be read.
     */
    public static MAMDataProtectionInfo getProtectionInfo(final
InputStream input) throws IOException;

    /**
     * Get the protection info on a stream.
     *
     * @param input
     *            Input bytes to get information on. These must be bytes
 *            returned by a previous call to protect() or a copy of
 *            such bytes.
     * @return Data protection info, or null if there is no protection
 *            info.
     * @throws IOException
     *             If the input cannot be read.
     */
    public static MAMDataProtectionInfo getProtectionInfo(final byte[]
input) throws IOException;
}
```

Content Providers

If the app provides corporate data other than a **ParcelFileDescriptor** through a **ContentProvider**, the app must call the **MAMContentProvider** method `isProvideContentAllowed(String)`, passing the owner identity's UPN (user principal name) for the content. If this function returns false, the content *may*

not be returned to the caller. File descriptors returned through a content provider are handled automatically based on the file identity.

Selective Wipe

If an app registers for the **WIPE_USER_DATA** notification, it will not receive the benefit of the SDK's default selective wipe behavior. For multi-identity aware apps, this loss may be more significant since MAM default selective wipe will wipe only files whose identity is targeted by a wipe.

If a multi-identity aware application wishes MAM default selective wipe to be done ***and*** wishes to perform its own actions on wipe, it should register for **WIPE_USER_AUXILIARY_DATA** notifications. This notification will be sent immediately by the SDK before it performs the MAM default selective wipe. An app should never register for both WIPE_USER_DATA and WIPE_USER_AUXILIARY_DATA.

MAM Service Enrollment (new!)

Overview

Intune MAM without device enrollment, also known as MAM-WE or MDM-less MAM, allows apps to be managed by Intune without device enrollment with Intune MDM. The Company Portal is still required to be installed on the device, but the user does not need to sign into the Company Portal and enroll the device.

MAM Service enrollment is done on a per-app basis. The app should enroll with the MAM service after authenticating the user, when it's determined the user is corporate/managed. The app should block the user from accessing protected, managed content until the enrollment process completes.

If the user is not licensed for the Intune MAM service, or if enrollment completes successfully, the user can be allowed to access protected content. If the user is licensed but the enrollment process fails, the user should be blocked from accessing protected content until enrollment successfully completes.

The app should un-enroll with the MAM service when removing the managed user.

MAMEnrollmentManager

The app can enroll in the Intune Mobile Application Management service by calling one of the **MAMEnrollmentManager**'s `enrollApplication()` methods:

```
package com.microsoft.intune.mam.policy;
```

```
public interface MAMEnrollmentManager {
    public enum Result {
        AUTHORIZATION_NEEDED,
        NOT_LICENSED,
        ENROLLMENT_SUCCEEDED,
        ENROLLMENT_FAILED,
        WRONG_USER,
        MDM_ENROLLED,
        UNENROLLMENT_SUCCEEDED,
        UNENROLLMENT_FAILED,
        PENDING,
        COMPANY_PORTAL_REQUIRED;
    }

    Result enrollApplication(String identity);
    Result enrollApplication(String identity, String refreshToken);
    Result unenrollApplication(String identity);
    boolean isApplicationEnrolled(String identity);

    void registerADALConnectionDetails(String identity,
ADALConnectionDetails adalDetails);

}
```

A reference to the **MAMEnrollmentManager** interface can be obtained using the code below:

```
MAMEnrollmentManager mgr = MAMComponents.get(MAMEnrollmentManager.class);

// make use of mgr
```

The **MAMEnrollmentManager** instance returned will never be null. If the Company Portal app is not installed on the device, the instance will be an offline version that can check whether the user is Intune-licensed before prompting to install the Company Portal for full enrollment functionality.

Enroll:

Once the **MAMEnrollmentManager** instance is retrieved, the app can enroll by calling one of its `enrollApplication()` methods. The identity argument to these methods is the displayable ID that ADAL stores in the **AuthenticationResult** it returns. The version that takes a single string for the identity of the enrolling user will internally attempt to get the user's ADAL refresh token from the ADAL cache.

- If the calling app provides its own implementation of ADAL's `ITokenCacheStore` interface, then it may not be possible for the **MAMEnrollmentManager** to obtain the refresh token from ADAL. In this case, the calling app must use the second version of the `enrollApplication()` method that takes both the user identity and the user's refresh token.

- **Note:** a valid `refreshToken` must be used if `enrollApplication(..., refreshToken)` is

called.

- The `enrollApplication()` methods are otherwise interchangeable, and the app can call the second version of to provide the refresh token if you prefer.

The `enrollApplication()` methods are usually asynchronous. It's possible sometimes for the result to be produced synchronously, so you should be prepared to **receive the result in either manner**.

- In the asynchronous case, the methods returns a Result code of **PENDING** and an SDK Notification will be received, as detailed below, when the enrollment process is complete.
- In case of successful enrollment, if the Intune policy is refreshed, the **REFRESH_POLICY** notification will be sent before the notification of the enrollment success.
- If the app already enrolled for the given user, **ENROLLMENT_SUCCEEDED** will be returned via notification. If the app is already enrolled for a *different user*, **WRONG_USER** will be returned.
- **Note:** It is **not** currently possible for the app to be enrolled for more than one user at a time, and all managed apps must be enrolled for the same user (the only managed identity allowed on the device). In order to successfully enroll as a new user, any apps currently enrolled for the previous user must be unenrolled first.

Unenroll:

To unenroll from the Intune MAM service, you must call the `unenrollApplication()` method. This method is asynchronous, due in part to the need to synchronously send a wipe notification back from the Company Portal app during unenrollment.

- The `unenrollApplication()` method will immediately return **PENDING**, and the final result will be delivered via an enrollment notification (UNENROLLMENT_SUCCEEDED or UNENROLLMENT_FAILED).
- **Note:** It's possible for an app to receive an unenrollment notification without calling `unenrollApplication()` due to a wipe command, MDM-unenroll, or enroll of a different user.

Determine if app is enrolled for given user: Call the `isApplicationEnrolled()` method and pass in the user's UPN.

Result codes

Result code	Explanation
AUTHORIZATION_NEEDED	Unable to obtain the MAMService token due to an invalid ADAL refresh token. The app should prompt the user for credentials to obtain a valid refresh token and then try again.
NOT_LICENSED	The user is not licensed for the MAMService, or the attempt to contact the location/licensing service failed. The app should

Result code	Explanation
	continue as if it were not managed by Intune.
ENROLLMENT_SUCCEEDED	The enrollment attempt succeeded, or the user was already enrolled. In the case of a successful enrollment, a policy refresh notification will be sent before this notification.
ENROLLMENT_FAILED	The enrollment attempt failed. Further details can be found in the device logs. The app should not allow access to protected content in this state, since it was previously determined that the user is licensed for the MAMService.
WRONG_USER	Only one user per device can enroll apps. In order to enroll successfully as a different user, all enrolled apps must be unenrolled first. Otherwise, this app must enroll as the primary user. This check happens after the license check, so the user should be blocked from accessing protected content until the app is successfully enrolled.
UNENROLLMENT_SUCCEEDED	Unenrollment was successful.
UNENROLLMENT_FAILED	The unenrollment request failed. Further details can be found in the device logs.
PENDING	An asynchronous operation has been successfully scheduled. The final result of the operation will be sent via notification.
COMPANY_PORTAL_REQUIRED	The user is licensed for MAMService, but the app cannot be enrolled until the Company Portal app is installed on the device.

Dynamic ADAL configuration (optional)

Normally, the AAD configuration for Intune MAM to use is given in Android Manifest as discussed above. Some applications use multiple AAD authorities and therefore require dynamic configuration.

If your app requires dynamic configuration, you must use the `registerADALConnectionDetails` method in **MAMEnrollmentManager** before calling `enrollApplication`. For example:

```
enrollmentManager.registerADALConnectionDetails(userToEnroll,
    new ADALConnectionDetails(aadAuthority, clientID,
    nonBrokerRedirectURI, skipBroker))
```

Company Portal requirement prompt override (optional)

If the **COMPANY_PORTAL_REQUIRED** result is received, Intune will block use of activities that use the identity for which enrollment was requested. The SDK will cause those activities to instead display a

prompt to download the Company Portal. If you want to prevent this behavior in your app, activities may implement `MAMActivity.onMAMCompanyPortalRequired`.

This method is called before the SDK displays its default blocking UI. If the app changes the activity identity or unenrolls the user who attempted to enroll, the SDK will not block the activity. In this situation, it is up to the app to avoid leaking corporate data.

Notifications

A new type of **MAMNotification** has been added in order to inform the app that the enrollment request has completed. The **MAMEnrollmentNotification** will be received through the **MAMNotificationReceiver** interface as described in the Registering for notifications from the SDK section.

```
public interface MAMEnrollmentNotification extends MAMUserNotification {
    MAMEnrollmentManager.Result getEnrollmentResult();
}
```

The `getEnrollmentResult()` method returns the result of the enrollment request. Since **MAMEnrollmentNotification** extends **MAMUserNotification**, the identity of the user for whom the enrollment was attempted is also available. The app must implement the **MAMNotificationReceiver** interface to receive these notifications, detailed in the SDK Notifications section.

App requirements

For authentication and licensing checks, as well as the periodic check-in operations, the app must provide the ADAL parameters as specified in the ADAL section. In particular, the **ClientID** and **Authority** values are required.

Since the MAMEnrollmentManager has a dependency on ADAL for authentication, the app must use a compatible version of the ADAL library. This feature cannot be fully supported with versions earlier than 1.1.18.

The app is expected to call the `enrollApplication` method periodically even though it may be already enrolled. Why?

1. If the app is not currently enrolled because the user was not licensed, the app should call `enrollApplication` to determine if the user's tenant has been onboarded in the past 24 hours.
2. If the app has successfully enrolled, subsequent calls to `enrollApplication` will update the user's refresh token. It is necessary to keep the refresh token up-to-date in order for the Company Portal to perform background check-in operations and pick up any policy updates from the Intune MAM Service. Therefore `enrollApplication` should be called every time the user's refresh token is updated, **or every 24 hours at the least**.

3. If attempting to use a background service to perform this periodic enrollment, please be aware that `enrollApplication` calls can lead to UI elements appearing on screen. Take that into consideration and test heavily if you decide to do so – it may lead to strange UX elements.

Protecting Backup data

As of Android Marshmallow (API 23), Android has two ways for an app to back up its data. Each option is available to your app and requires different steps to ensure that Intune data protection is correctly implemented. You can review the table below on corresponding actions required for correct data protection behavior. You can read more about the backup methods in the Android API guide.

Auto Backup for Apps

Android began offering automatic full backups to Google Drive for apps on Android Marshmallow devices, regardless of the app's target API. In your AndroidManifest.xml, if you explicitly set `android:allowBackup` to **false**, then your app will never be queued for backups by Android and "corporate" data will stay within the app. In this case, no further action is necessary.

However, by default the `android:allowBackup` attribute is set to true, even if `android:allowBackup` isn't specified in the manifest file. This means all app data is automatically backed up to the user's Google Drive account, a default behavior that poses a **data leak risk**. Therefore, the SDK requires the changes outlined below to ensure that data protection is applied. It is important to follow the guidelines below to protect customer data properly if you want your app to run on Android Marshmallow devices.

Intune allows you to utilize all the Auto Backup features available from Android, including the ability to define custom rules in XML, but you must follow the steps below to secure your data:

1. If your app does **not** use its own custom BackupAgent, use the default MAMBackupAgent to allow for automatic full backups that are Intune policy compliant. If you do this, you can ignore the `android:fullBackupOnly` manifest attribute, as it's not applicable for our backup agent. Place the following in the app manifest:

   ```
   android:backupAgent="com.microsoft.intune.mam.client.app.backup.MAMDefaultBackupAgent"
   ```

2. **[Optional]** If you implemented an optional custom BackupAgent, you need to make sure to use MAMBackupAgent or MAMBackupAgentHelper. See the following sections. Consider switching to using Intune's **MAMDefaultFullBackupAgent** (described in step 1) which provides easy back up on Android M and above.

3. When you decide which type of full backup your app should receive (unfiltered, filtered, or none)

you'll need to set the attribute `android:fullBackupContent` to true, false, or an XML resource in your app.

4. Then, you ***must*** copy whatever you put into `android:fullBackupContent` into a metadata tag named `com.microsoft.intune.mam.FullBackupContent` in the manifest.

   Example 1: If you want your app to have full backups without exclusions, set both the `android:fullBackupContent` attribute and `com.microsoft.intune.mam.FullBackupContent` metadata tag to **true**:

   ```
   android:fullBackupContent="true"
   ...
   <meta-data android:name="com.microsoft.intune.mam.FullBackupContent"
   android:value="true" />
   ```

   Example 2: If you want your app to use its custom BackupAgent and opt out of full, Intune policy compliant, automatic backups, you must set the attribute and metadata tag to **false**:

   ```
   android:fullBackupContent="false"
   ...
   <meta-data android:name="com.microsoft.intune.mam.FullBackupContent"
   android:value="false" />
   ```

   Example 3: If you want your app to have full backups according to your custom rules defined in an XML file, please set the attribute and metadata tag to the same XML resource:

   ```
   android:fullBackupContent="@xml/my_scheme"
   ...
   <meta-data android:name="com.microsoft.intune.mam.FullBackupContent"
   android:resource="@xml/my_scheme" />
   ```

Key/Value Backup

The Key/Value Backup option is available to all APIs 8+ and uploads app data to the Android Backup Service. The amount of data per user of your app is limited to 5MB. If you use Key/Value Backup, you must use a **BackupAgentHelper** or a **BackupAgent**.

BackupAgentHelper

BackupAgentHelper is easier to implement than BackupAgent both in terms of native Android functionality and Intune MAM integration. BackupAgentHelper allows the developer to register entire

files and shared preferences to a **FileBackupHelper** and **SharedPreferencesBackupHelper** (respectively) which are then added to the BackupAgentHelper upon creation. Follow the steps below to use a BackupAgentHelper with Intune MAM:

1. To utilize multi-identity backup with a BackupAgentHelper, follow the Android guide to Extending BackupAgentHelper.

2. Have your class extend the MAM equivalent of BackupAgentHelper, FileBackupHelper, and SharedPreferencesBackupHelper.

Android class	MAM equivalent
BackupAgentHelper	MAMBackupAgentHelper
FileBackupHelper	MAMFileBackupHelper
SharedPreferencesBackupHelper	MAMSharedPreferencesBackupHelper

Following these guidelines will lead to a successful multi-identity backup and restore.

BackupAgent

A BackupAgent allows you to be much more explicit about what data is backed up. Because the developer is fairly responsible for the implementation, there are more steps required to ensure appropriate data protection from Intune MAM. Since most of the work is pushed onto you, the developer, Intune MAM integration is slightly more involved.

Integrate MAM:

1. Please carefully read the Android guide for Key/Value Backup and specifically Extending BackupAgent to ensure your BackupAgent implementation follows Android guidelines.

2. Have your class extend **MAMBackupAgent**.

Multi-identity Backup:

1. Before beginning your backup, check that the files or data buffers you plan to back up are indeed **permitted by the IT administrator to be backed up** in multi-identity scenarios. We provide you with the `isBackupAllowed` function in **MAMFileProtectionManager** and **MAMDataProtectionManager** to determine this. If the file or data buffer is not allowed to be backed up, then you should not continue including it in your backup.

2. At some point during your backup, if you want to back up the identities for the files you checked in step 1, you must call `backupMAMFileIdentity(BackupDataOutput data, File ... files)` with the files from which you plan to extract data. This will automatically create new backup entities and write them to the **BackupDataOutput** for you. These entities will be automatically consumed upon restore.

Multi-identity Restore:

The Data Backup guide specifies a general algorithm for restoring your application's data and provides a code sample in the Extending BackupAgent section. In order to have a successful multi-identity restore, you must follow the general structure provided in this code sample with special attention to the following:

1. You must utilize a `while(data.readNextHeader())` * loop to go through the backup entities.
2. You must call `data.skipEntityData()` * if `data.getKey()` * does not match the key you wrote in `onBackup`. Without performing this step, your restores may not succeed.
3. Avoid returning while consuming backup entities in the `while(data.readNextHeader())` * construct, as the entities we automatically write will be lost.

*Please assume that `data` is the local variable name for the **BackupDataInput** that is passed to your app upon restore.

Application Configuration Channel

As of the MAM Service 4.0 API, Application Configuration data can be provided via MAM Service check-in responses to MAM-WE enabled apps. Certain partners have requested the ability to integrate their own console's custom settings into normal MAM-WE operation and, so we provide an SDK class to access the data retrieved from these consoles.

Check the Javadoc for **MAMAppConfigManager** and **MAMAppConfig** for more information.

Sample code:

```
MAMAppConfigManager configManager =
MAMComponents.get(MAMAppConfigManager.class);
String identity = “user@contoso.com”

// get the managed user if one exists
MAMUserInfo userInfo = MAMComponents.get(MAMUserInfo.class);
if (userInfo != null) {
   identity = userInfo.getPrimaryUser();
}

MAMAppConfig appConfig = configManager.getAppConfig(identity);
LOGGER.info("App Config Data = " + (appConfig == null ? "null" :
appConfig.getFullData()));
```

Style Customization (optional)

Views generated by the MAM SDK can be visually customized to more closely match the app in which it is integrated. You can customize primary, secondary, and background colors, as well as the size of the app logo. This style customization is optional and defaults will be used if no custom style is configured.

How to customize

In order to have style changes apply to the Intune MAM views, you must first create a style override XML file. This file should be placed in the "/res/xml" directory of your app and you may name it whatever you like. Below is an example of the format this file needs to follow.

```
<?xml version="1.0" encoding="utf-8"?>
<styleOverrides>
    <item
        name="foreground_color"
        resource="@color/red"/>
    <item
        name="accent_color"
        resource="@color/blue"/>
    <item
        name="background_color"
        resource="@color/green"/>
    <item
        name="logo_image"
        resource="@drawable/app_logo"/>
</styleOverrides>
```

You must reuse resources that already exist within your app. For example, you must define the color green in the colors.xml file and reference it here. You cannot use the Hex color code "#0000ff." The maximum size for the app logo is 110 dip (dp). You may use a smaller logo image, but adhering to the maximum size will yield the best looking results. If you exceed the 110 dip limit, the image will scale down and possibly cause blurring.

Below is the complete list of allowed style attributes, the UI elements they control, their XML attribute item names, and the type of resource expected for each.

Style attribute	UI elements affected	Attribute item name	Expected resource type
Background color	PIN screen background color PIN box fill color	background_color	Color

Style attribute	UI elements affected	Attribute item name	Expected resource type
Foreground color	Foreground text color PIN box border in default state Characters (including obfuscated characters) in PIN box when user enters a PIN	foreground_color	Color
Accent color	PIN box border when highlighted Hyperlinks	accent_color	Color
App logo	Large icon that appears in the Intune app PIN screen	logo_image	Drawable

ProGuard

If ProGuard (or any other shrinking/obfuscation mechanism) is used as a build step, Intune SDK classes must be excluded. For ProGuard, this can be accomplished using the following configuration line:

```
-keep class com.microsoft.intune.mam.** { *; }
```

ProGuard will create warnings for classes referenced by the SDK that are shipped as part of the Company Portal. It is safe to ignore those classes using the following configuration lines:

```
-dontwarn com.microsoft.bond.**
-dontwarn Microsoft.Telemetry.**
-dontwarn com.microsoft.intune.mam.client.telemetry.clientschema.**
```

Azure Active Directory Authentication Libraries (ADAL) may have its own ProGuard restrictions. If your app integrates ADAL, you must follow the ADAL documentation on these restrictions.

Limitations

File Size Limitations

For large code bases that run without ProGuard, the limitations of the Dalvik executable file format become an isssue. Specifically, the following limitations may occur:

- The 65K limit on fields.
- The 65K limit on methods.

When many projects are included, every android:package will get a copy of the `R` package, which will dramatically increase the number of fields as libraries are added. The following recommendations may help mitigate this limitation:

- All library projects should share the same android:package where possible. This will not sporadically fail in the field, since it is purely a build-time problem. In addition, newer versions of the Android SDK will pre-process DEX files to remove some of the redundancy. This lowers the distance from the fields even further.
- Use the newest Android SDK build tools available.
- Remove all unnecessary and unused libraries (e.g. `android.support.v4`)

Policy Enforcement Limitations

- **Screen Capture**: The SDK is unable to enforce a new screen capture setting value in Activities that have already gone through Activity.onCreate. This can result in a period of time where the app has been configured to disable screenshots but screenshots can still be taken.
- **Using Content Resolvers**: The "transfer or receive" Intune policy may block or partially block the use of a content resolver to access the content provider in another app. This will cause ContentResolver methods to return null or throw a failure value (e.g. `openOutputStream` will throw `FileNotFoundException` if blocked). The app can determine whether a failure to write data through a content resolver was caused by policy (or would be caused by policy) by making the call:

```
MAMComponents.get(AppPolicy.class).getIsSaveToLocationAllowed(content
URI);
```

- **Exported Services**: The AndroidManifest.xml file included in the Intune App SDK contains **MAMNotificationReceiverService**, which must be an exported service to allow the Company Portal to send notifications to an enlightened app. The service checks the caller to ensure that only the Company Portal is allowed to send notifications.

Recommended Android Best Practices

The Intune SDK maintains the contract provided by the Android API, though failure conditions may be triggered more frequently as a result of policy enforcement. These Android best practices will reduce the likelihood of failure:

- Android SDK functions that may return null have a higher likelihood of being null now. To minimize issues, ensure that null checks are in the right places.

- Features that can be checked for must be checked for through their MAM replacement APIs.
- Any derived functions must call through to their super class versions.
- Avoid use of any API in an ambiguous way. For example, using `Activity.startActivityForResult` without checking the requestCode will cause strange behavior.