Android SDK Integration Guide - Core

Android SDK Integration Guide - Core

Version 1.3.12

This is Phunware's Android SDK README for the Core module. Visit http://maas.phunware.com/ f or more details and to sign up.

ON THIS PAGE

- Android SDK Integration Guide - Core
- Requirements
- Documentation
- Overview
- Session Setup and Usage
  - Update Android Manifest
  - Optional Setup
- Install Modules
  - Register API Keys
  - Defining Keys in the Manifest (Optional)
    - Application ID
    - Access Key
    - Signature Key
    - Encryption Key
- Activities: Start and Stop Session
  - Start
  - Stop
- Analytical Data
- Verify Manifest
- Compiling with ProGuard
- Integrating with Google Play Services API
- PwLog
- Attribution

Requirements

- Android SDK 2.2+ (API level 8) or above
- Android Target 4.1.1.4
- Android Support v4 18.0.+
- Gson 2.2.4
- OkHttp 1.6.0
- okhttp-urlconnection 1.6.0
- Retrofit 1.6.0

Documentation

MaaS Core documentation is included in the Documents folder in the repository as both HTML and as a .jar. You can also find the latest documentation here: Core API iOS Reference

Overview

MaaS Core is designed to have as little impact on developers as possible. It is also a necessary requirement for all other MaaS SDKs.

MaaS Core helps to gather data for analytical purposes and also maintains a session throughout an app. A session is the duration that a user interacts with an app and it's used to uniquely identify analytical data. There are two steps to set up and maintain sessions in any app: application keys need to be registered, then sessions can be started and stopped.

Session Setup and Usage

Update Android Manifest

The MaaS Core relies on a few settings in order to communicate with the MaaS server. The first is the Internet permission, the second is a service that runs network communications asynchronously and the third helps to uniquely identify the device.

```
<!-- necessary for MaaS Core to communicate
with the MaaS server -->
<uses-permission
android:name="android.permission.INTERNET" />

<application>
 <!-- other definitions -->
 <!-- necessary for MaaS Core to communicate
with the MaaS server -->
 <service
android:name="com.phunware.core.internal.CoreSe
rvice" />
 <!-- necessary to generate a UDID -->
 <service
android:name="com.OpenUDID.OpenUDID_service">
  <intent-filter>
   <action android:name="com.openudid.GETUDID"
/>
  </intent-filter>
 </service>
</application>
```

Optional Setup

There are optional configurations that can be set in the `AndroidManifest.xml`. These may be safely left unconfigured.

The first is a permission `ACCESS_NETWORK_STATE`. This permission allows access to the network state (connected, disconnected, etc.). This SDK uses it to determine if there is a network connection before sending analytic events. If there is none connected, then the events are queued up to be sent when a connection is available.

```
<!-- Optional: If used, this will result in
network calls being made more efficiently. -->
<uses-permission
android:name="android.permission.ACCESS_NETWORK
_STATE" />

<!-- Optional: Set this permission in order to
get more detailed analytics. -->
<uses-permission
android:name="android.permission.ACCESS_WIFI_ST
ATE" />

<!-- Optional: Set these following permissions
to get location data in analytics reports -->
<uses-permission
android:name="android.permission.ACCESS_FINE_LO
CATION" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_
LOCATION" />
```

Additionally, there is a `CoreReceiver` that can be used. Currently, this is only used to receive connectivity change events. This is also used to help send analytic events more efficiently.

```
<!-- used by MaaS Core for efficient analytic
caching and flushing -->
<receiver
android:name="com.phunware.core.internal.CoreRe
ceiver">
    <intent-filter>
        <action
android:name="android.net.conn.CONNECTIVITY_CHA
NGE" />
    </intent-filter>
</receiver>
```

> Although the `CoreReceiver` and `ACCESS_NETWORK_STATE` permission do similar things, they are still unique and it is beneficial to use them simultaneously. The receiver will get updates when connectivity changes. However, every other `BroadcastReceiver` that is defined with that intent filter will as well, so the update may not be instantaneous. Connectivity could drop, so checking if a connection is available may be a faster and more reliable method. The `CoreReceiver` will also send any queued-up analytic events once connection is restored.

Install Modules

Each module requires the Core SDK to run. In order to use any extra modules, they must first be installed into the Core SDK. This is done in code with one line and should be done in the application's onCreate method:

```
@Override
public void onCreate() {
    super.onCreate();
    /* other code */

PwCoreSession.getInstance().installModules(PwAl
ertsModule.getInstance(), ...);
    /* other code */
}
```

Register API Keys

Create a class that extends `Application` and register the `Application` class in the `AndroidManifest.xml` file. This should be called *after* a call to install additional modules. Register the access, signature and encryption key in the `Application's onCreate` method:

```
@Override
public void onCreate() {
    super.onCreate();
    /* other code */
    /* install additional modules */

PwCoreSession.getInstance().registerKeys(this,
      "<my_appid>",
                "<my_accesskey>",
                "<my_signaturekey>",
                "<my_encryptionkey>");

    /* other code */
}
```

Defining Keys in the Manifest (Optional)

```
@Override
public void onCreate() {
    super.onCreate();
    /*
     * Alternatively, register keys when the
keys are defined
     * in the manifest under metadata tags.
     */

PwCoreSession.getInstance().registerKeys(this);
}
```

The `meta-data` tags must be defined *inside* of the `application` tag.

```
<meta-data
android:name="META_DATA_APPLICATION_ID"
android:value="@string/app_id" />
<meta-data android:name="META_DATA_ACCESS_KEY"
android:value="@string/access_key" />
<meta-data
android:name="META_DATA_SIGNATURE_KEY"
android:value="@string/signature_key" />
<meta-data
android:name="META_DATA_ENCRYPTION_KEY"
android:value="@string/encrypt_key" />
```

Application ID

You can find your application key in the MaaS portal.

Access Key

The access key is a unique key that identifies the client making the request. You can find your access key in the MaaS portal.

Signature Key

The signature key is a unique key that is used to sign requests. The signature is used to check both request authorization and data integrity. You can find your signature key in the MaaS portal.

Encryption Key

The encryption key is used to encrypt and decrypt data that is exchanged between the client and the server. You can find your encryption key in the MaaS portal.

Activities: Start and Stop Session

A session is active once it is started and inactive when it has been stopped. A session will expire after two minutes (i.e. "expiration timeout") unless it is restarted prior.

Start

To start the session in an `Activity`, get the `PwCoreSession` instance and call `activityStartSession(activity)`. The passed-in `Activity` should be the current activity. **This should be called in the activities `onStart` method**. This will ensure the session is properly created before fragments can be attached to the activity.

```
@Override
public void onStart() {
    super.onStart();
    /* other code */

PwCoreSession.getInstance().activityStartSessio
n(this);
    /* other code */
}
```

Stop

To stop the session in an `Activity`, get the `PwCoreSession` instance and call `activityStopSession(activity)`. The passed-in `Activity` should be the current activity. **This should be called in the activities `onStop` method**.

```
@Override
public void onStop() {
    super.onStop();
    /* other code */

PwCoreSession.getInstance().activityStopSession
(this);
    /* other code */
}
```

Calling `activityStopSession(context)` will stop the session. However, if `activityStartSession(context)` is called before the expiration timeout is reached, then the session will be resumed. (This is how a session persists between activity transitions.)

Analytical Data

Various types of analytical data are collected and sent to the MaaS server for usage. Most are available without any extra permissions:

1. App Session ID
2. App Version Name
3. App Version Code
4. App Access Key
5. Android Build Version
6. Android OS
7. Screen Size
8. Screen Density
9. Screen DPI
10. Carrier
11. Device OpenGL Version
12. Device Sensors (Accelerometer, Proximity Monitor, etc.)
13. Device Language
14. *Device MAC Address
15. *Device Wi-Fi Info
16. *Device SSID
17. *Device IP
18. **Device User Agent
19. Device Make
20. Device Model

21. Device ID
22. Timestamp
23. Timezone

* Requires `ACCESS_WIFI_STATE` Permission

In order to get MAC Address, Wi-Fi Info, SSID and IP data, the permission for `ACCESS_WIFI_STATE` will need to be included in the manifest. If it is not included, the data will not be collected.

** Requires `ACCESS_NETWORK_STATE` Permission
In order to get the device User Agent, the permission for `ACCESS_NETWORK_STATE` will need to be included in the manifest.
If it is not included, the data will not be collected.

Verify Manifest

`PwCoreModule` has a convenience method to check if the manifest for the Core SDK is set up properly. This should only be used for development and testing, not in production.

Call the method with the line `PwCoreModule.validateManifestCoreSetup(context)`. The passed-in context should be the application context. If there is an error, then an `IllegalStateException` will be thrown with an error message regarding what couldn't be found.

Compiling with ProGuard

If you use ProGuard in your app, be sure to include the following lines in your ProGuard configuration:

```
-keep class * implements android.os.Parcelable
{
    public static final
android.os.Parcelable$Creator *;
}
```

Integrating with Google Play Services API

Google Play Services API offers many features that your app can use. Go to the GooglePlayServicesIntegration sample app to see how MaaS SDKs utilize the API.

PwLog

To view logs from the MaaS SDKs, use `PwLog.setShowDebug(true);`. The logs are all turned off by default.

Attribution

MaaS Core uses the following third-party components:

Component	Description	License
Retrofit	Type-safe REST client for Android and Java by Square, Inc.	Apache 2.0

GSON	A Java library to convert JSON to Java objects and vice-versa.	Apache 2.0