iPass SmartConnect Getting Started Guide for Android

Version 1.5.4.90 MARCH 2017

Introduction

This document covers API level details of the iPass SmartConnect SDK for Android and provides details how to use SDK in android studio. Prerequisites and required settings need to be set by the client.

The iPass SmartConnect SDK provides an end-to-end connectivity framework for WiFi networks. The SDK exposes APIs to provision the client with iPass networks.

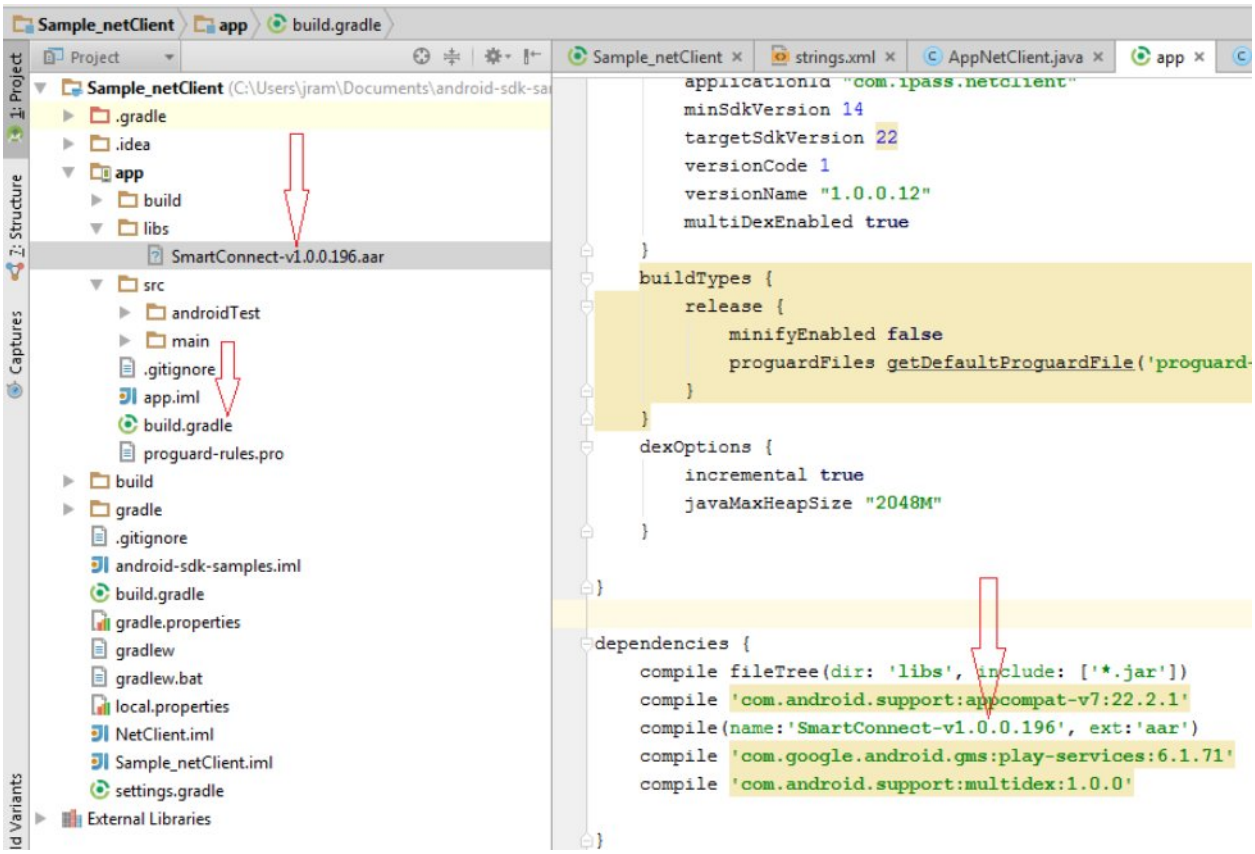
Activation can be done via token or credentials. Once provisioned, the client application will be enabled to make seamless connection on iPass global hotspot spots.

Getting Started

Using Android Studio to Add the SmartConnect Library

1. Open your project and navigate to project view, then copy the SmartSDK-Vx.x.x.xx.aar and hotspot_core-release-vx.x.xx.aar from the module's libs folder.
2. Add dependency in the module's Gradle file:

```
compile(name:'SmartConnect-vx.x.x.xx', ext:'aar')
compile(name:'hotspot_core-release-vx.x.xx', ext:'aar')
```



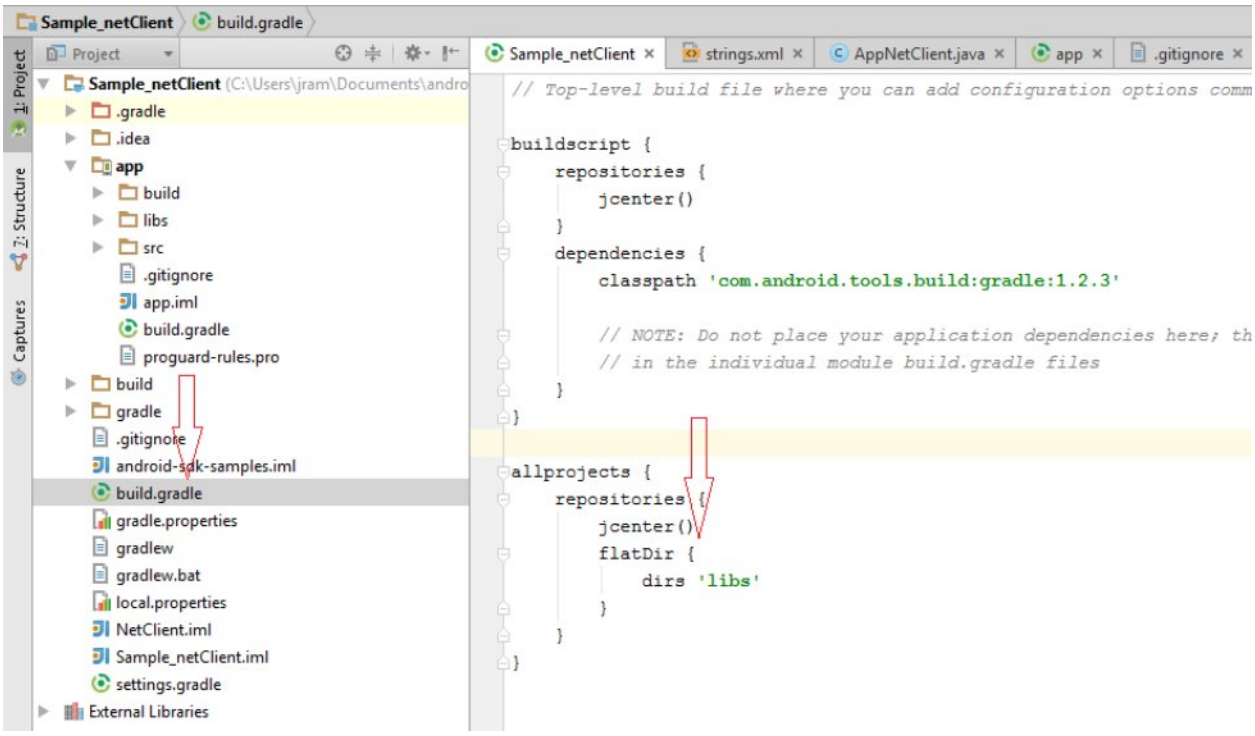
3. Add dependency to Google play service library in module's Gradle file:

```
compile 'com.google.android.gms:play-services-location:8.4.0'
compile 'com.google.android.gms:play-services-gcm:8.4.0'
compile 'com.google.android.gms:play-services-maps:8.4.0'
compile 'com.google.android.gms:play-services-analytics:8.4.0'
```

4. SDK versions 16 and newer are supported. Target version should be 23 or lower in module's Gradle file. SDK supports runtime permissions if Target version is 23.If host application is target to Android SDK version 23, hostApp should register for RuntimePermission callbacks and provide the result of permission back to SDK.

5. Add Dirs in `allprojects` section of application Gradle file:

```
allprojects {
   repositories {
       jcenter()
       flatDir {
           dirs 'libs'
       }
    }
}
```



6. Add the following permissions in the manifest file:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_LOGS" />
```

```
<uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION"/>
```

Starting SmartConnect Service

1. You must initialize SMCCore before making calls to other APIs. Make the call to SMCCore.initialize()from onCreate() function of “Application” class.

```
String SDKKey = "VALID SDK KEY";
try {
   SMCCore.initialize( SDKKey , getApplicationContext());
} catch (SMCException e) {
   e.printStackTrace();
}
```

   Note: If the host application has a service running in a process other than main, then the host app should initialize the call only when main process is started. Host app can use the following snippet of code to check the same:

```
            private boolean isMainProcess() {
                    Context context = getApplicationContext();
                         ActivityManager
    activityManager=(ActivityManager)context.getSystemService
    (Context.ACTIVITY_SERVICE);
         List<ActivityManager.RunningAppProcessInfo> appList
         = activityManager.getRunningAppProcesses();
                    for (ActivityManager.RunningAppProcessInfo info :
appList) {
                              //provide the extn of another process
                            if ( !info.processName.contains(<e.g
     ':remote'>)  && info.pid == android.os.Process.myPid() ) {
                                      return true;
                             }
                    }
                    return false;
            }
```

2. Call SMCActivation.activete() API to activate, activation API takes token or credentials.
3. Once activated, your device will be able to connect iPass footprints.

Using ProGuard

If you use ProGuard on the host application code, the following rules for ProGuard should be added to the ProGuard configuration:

```
-keepclassmembernames class * {
    public protected <methods>;
}

-keepclasseswithmembers class * {
   native <methods>;
}

-keepclassmembers class * {
   @android.webkit.JavascriptInterface <methods>;
}

-keep public interface com.openmobile.proguard.NonObfuscateable
-keep public class * implements com.openmobile.proguard.NonObfuscateable

-keepclassmembers class * implements com.openmobile.proguard.NonObfuscateable{
  public *;
  <methods>;
  native <methods>;
}

-keep public class com.ipass.smartconnect.activation.ActivationManager$*{*;}

-keep class * extends java.util.ListResourceBundle {
   protected Object[][] getContents();
}

# Keep SafeParcelable value, needed for reflection. This is required to support backwards
# compatibility of some classes.
-keep public class com.google.android.gms.common.internal.safeparcel.SafeParcelable {
   public static final *** NULL;
}
```

 # Needed by google-api-client to keep generic types and @Key annotations accessed via reflection

-keepclassmembers class * {
  @com.google.api.client.util.Key <fields>;
}

-keepattributes Signature,RuntimeVisibleAnnotations,AnnotationDefault

 # Keep the names of classes and members we need for client functionality.
-keepnames @com.google.android.gms.common.annotation.KeepName class *
-keepclassmembernames class * {
    @com.google.android.gms.common.annotation.KeepName *;
}

 # Needed for Parcelable/SafeParcelable Creators to not get stripped
-keepnames class * implements android.os.Parcelable {
    public static final ** CREATOR;
}

-keep class org.apache. {*;}**
-keep class org.slf4j. { *;}**
-keep class org.xbill. { *;}**
-keep class org.spongycastle. { *;}**
-keep class org.jsoup. { *;}**
-keep class org.json. { *;}**
-keep class ch.qos. { *;}**
-keep class com.google. { *;}**
-keep class com.devicescape. {*;}**
-keep class com.dd. {*;}**
-keep class com.google.gson. {*;}**
-keep class com.private_wifi. {*;}**

-dontwarn sun.net.**
-dontwarn com.accurisnetworks.**
-dontwarn org.apache.**

-dontwarn javax.annotation.**
-dontwarn org.slf4j.**
-dontwarn sun.net.**
-dontwarn org.xbill.**
-dontwarn de.blinkt.**
-dontwarn com.google.**
-dontwarn org.spongycastle.**
-dontwarn org.jsoup.**
-dontwarn org.json.**
-dontwarn ch.qos.**
-dontwarn com.devicescape.**
-dontwarn com.dd.**
-dontwarn com.google.gson.**
-dontwarn com.private_wifi.**

UI Recommendations

The host app should have a UI screen which displays the current connection status and supported network with annotation. The SDK will also provide a network list that is sorted based on supported networks and signal strength. Network confidence level can be used to apply annotations.

On Typical Networks

1. AUTO-CONNECT is enabled, SDK will connect seamlessly to iPass network.
2. AUTO-CONNECT is disabled, Host app should provide option to user to select network from network list.

Login Procedure for CAPTCHA-based Networks

Some networks require a CAPTCHA to be entered by the user so that the SDK can sign in to the network. This is relayed to the app with a connection status code of CAPTCHA_PENDING. When this occurs, the client will have to open a Webview with the CAPTCHA url provided with the network object. You will then have to enter the CAPTCHA shown in the Webview. Once you enter the CAPTCHA, the SDK will poll the url in the background and provide an appropriate status code as to the connection state.

CAPTCHA_TIMEOUT - CAPTCHA page has timed out. (The default timeout is 5 mins).

CAPTCHA_SUCCESS - The CAPTCHA entered by the user is correct.

CAPTCHA_CLOSE_LOGIN_FAILED - The CAPTCHA entered was correct, however the user's credentials were incorrect and so the login has failed.

CAPTCHA_GATEWAY_REJECTED - The max number of CAPTCHA attempts has been reached. The webpage will not accept any more CAPTCHA attempts from the same connection (The default limit is 15).

CAPTCHA_FAILED - The CAPTCHA entered is wrong. Error code is thrown as an intermediary every time a wrong CAPTCHA is entered. The connection is still active and the user can still attempt to enter the CAPTCHA again until the retry limit is reached.

If you close the Webview, the SDK should be notified to disconnect the active connection or the connection will stay active.

Speed Test

The SDK provides the APIs for performing Speed test functionality against the best reachable server configured in the profile. It provides a result in the form of latency, download, and upload data rate. The host application can start the speed test by calling the *SMCSpeedTest.startTest()* API when the device connects to a network. The SDK will provide the stats for latency, download, and upload data rate through callbacks. Reference code is available in *SpeedTestResultActivity.java* file of the SDK "Sample" application.

Accurate Usage Tracking

For Android Nougat and above, SmartConnect SDK uses NetworkStatsManager to obtain accurate usage information required for Demeter. The SDK requires the following two permissions:

- PACKAGE_USAGE_STATS : This is a protected permission and will require explicit consent from the user.
- READ_PHONE_STATE : This is a dangerous permission and can be obtained through the Runtime permission request.

To check if accurate usage tracking is enabled, use

SMCCore.getUsageTracker().getUsageTrackingState(context).

This returns an **EnumUsageTrackingState** that indicates what permission is required.

- If the return type is **USAGE_PERMISSION_REQUIRED** it means that **PACKAGE_USAGE_STATS** permission is not granted to the application. To obtain this permission, the client will have to redirect the user to the native usage page and the user will have to grant the permission. This can be done through launching an intent with Settings.ACTION_USAGE_ACCESS_SETTINGS action.
- If the return type is **PHONE_STATE_PERMISSION_REQUIRED** it means **READ_PHONE_STATE** permission is not granted to the application. This can be requested during runtime.

If the return type is ENABLED it means accurate tracking is enabled. **NOTE:** For devices running Android Nougat and later, this will be returned by default as no special permissions are required to track usage.

Logging

The SDK adds logging under the application package, which can then be extracted using the *SMCCore.extractLogs()* API. The API will extract the logs and store them in device memory as Log.zip. Add extract log functionality in the debug menu of the application.

Connection Error Codes

ISMCConnectionListener callback interface should be implemented to be notified when an error occurs.

VALUE	DESCRIPTION
100	The authentication attempt has failed.
101	Internal Gateway Error. There's some problem with the gateway. Please try again later.
102	Authentication request timed out on server.
103	Server has disabled authentication for this account. Please contact your administrator.
104	Unexpected response to the authentication request.
105	Unknown error.
106	Association with the Wi-Fi network failed.
107	Failed to Obtain IP.
108	Network error occurred.
109	Authorization rejected.

APIs

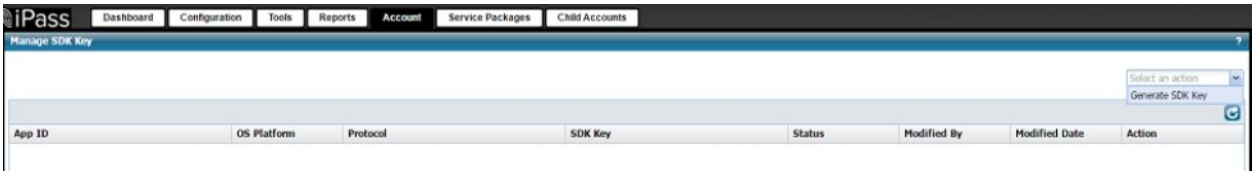
The SDK bundle has a documentation folder which contains more detailed information about the APIs that the iPass SDK provides.

Architecture Supported

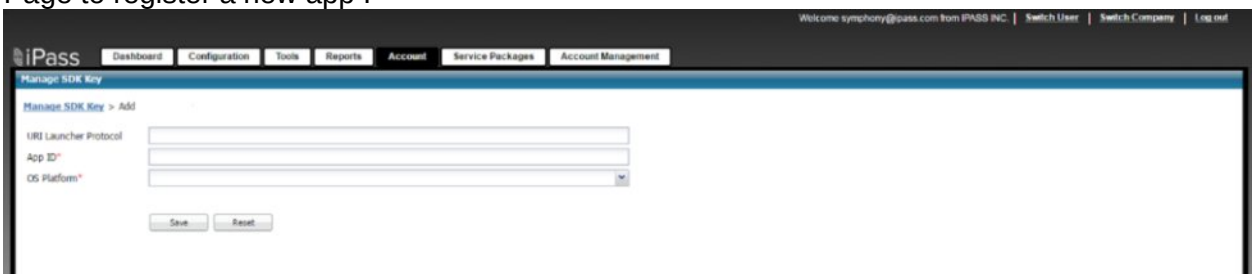
SDK supports armeabi, armeabi-v7a and arm64-v8a architectures. Host application, which is using native code, should add explicit JNI libraries for all the supported architecture.

How to register Your App and Generate an SDK Key

1. Sign in to the iPass Portal.
2. From the **Accounts** tab, select **Manage SDK Key**.
3. Select the **Generate SDK Key** to register your app and create a new SDK Key.



Page to register a new app .

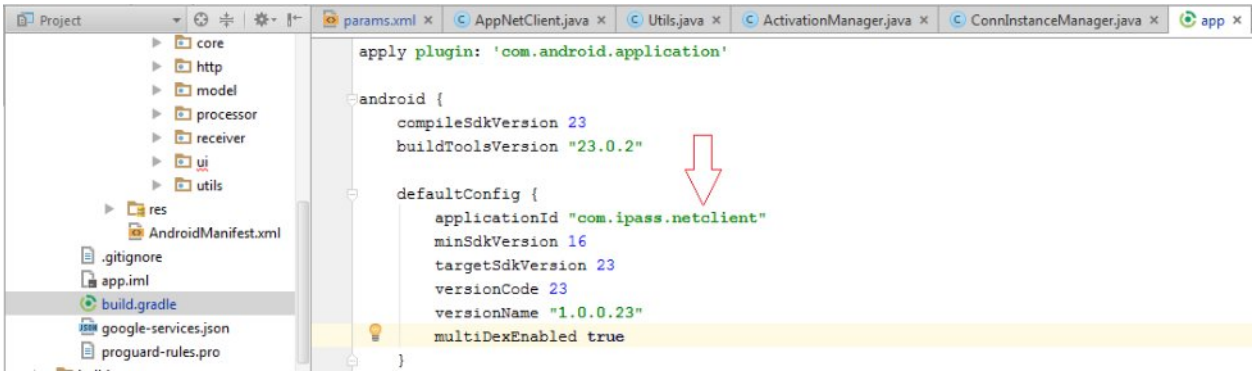


The **Generate SDK Key** page has the following fields:

- URI Launcher Protocol
- App ID
- OS Platform

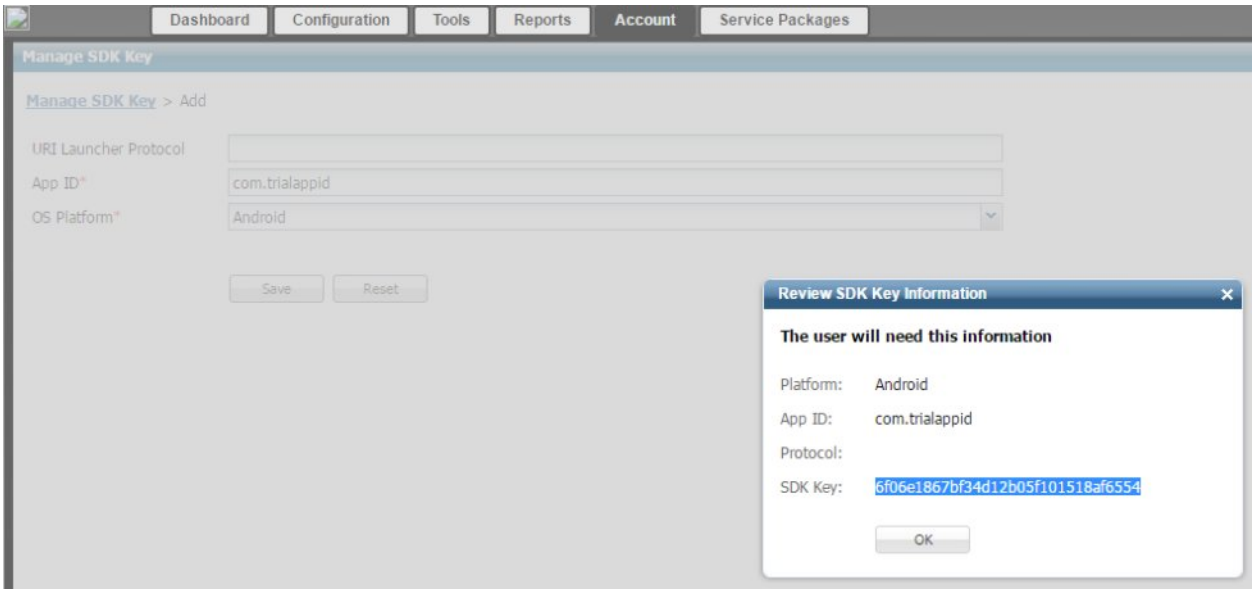
App ID and OS Platform are mandatory fields. The URI Launcher Protocol is not mandatory and can remain blank.

The App ID should be applicationId, which you can find in the app Gradle file. For example, reference app's application id is *com.ipass.netclient*.

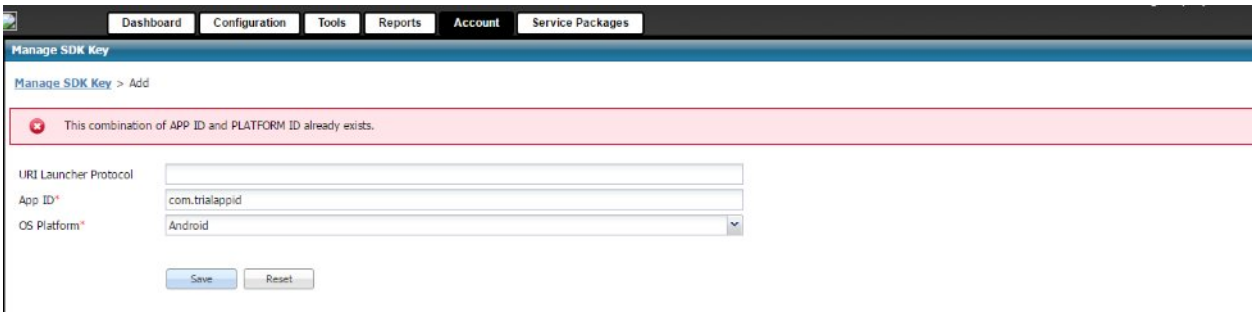


The SDK key is generated based on the combination of "App ID" and "Platform." This combination has to be unique and never used before.

Once the App is registered, **Review SDK Key Information** shows the generated SDK Key, along with app information.



If your “App ID” and “Platform” combination has been used previously, you’ll receive an error message:
“This combination of App ID and Platform ID already exists.”



After the SDK Key is generated, the App ID details can be viewed on the **Manage SDK Key** page. Copy the SDK key to use in the client app.