Retention Science Android SDK

Updated: Jan 7, 2015
Current Version: 1.0
Minimum API level: 5

Latest Revision Notes:

Updating document to reflect the minimum API level for Android app developers. The current minimum required API level for the Retention Science SDK is 5 (Android 2.0, Eclair). This should be adequate for most app developers.

If it is required to support a lower API level, modifications to the SDK can be made.

For more information on Android OS distribution see the following:

https://developer.android.com/about/dashboards/index.html

Document Index:

Part 1 - ::::: Implementing the Retention Science JAR file :::::

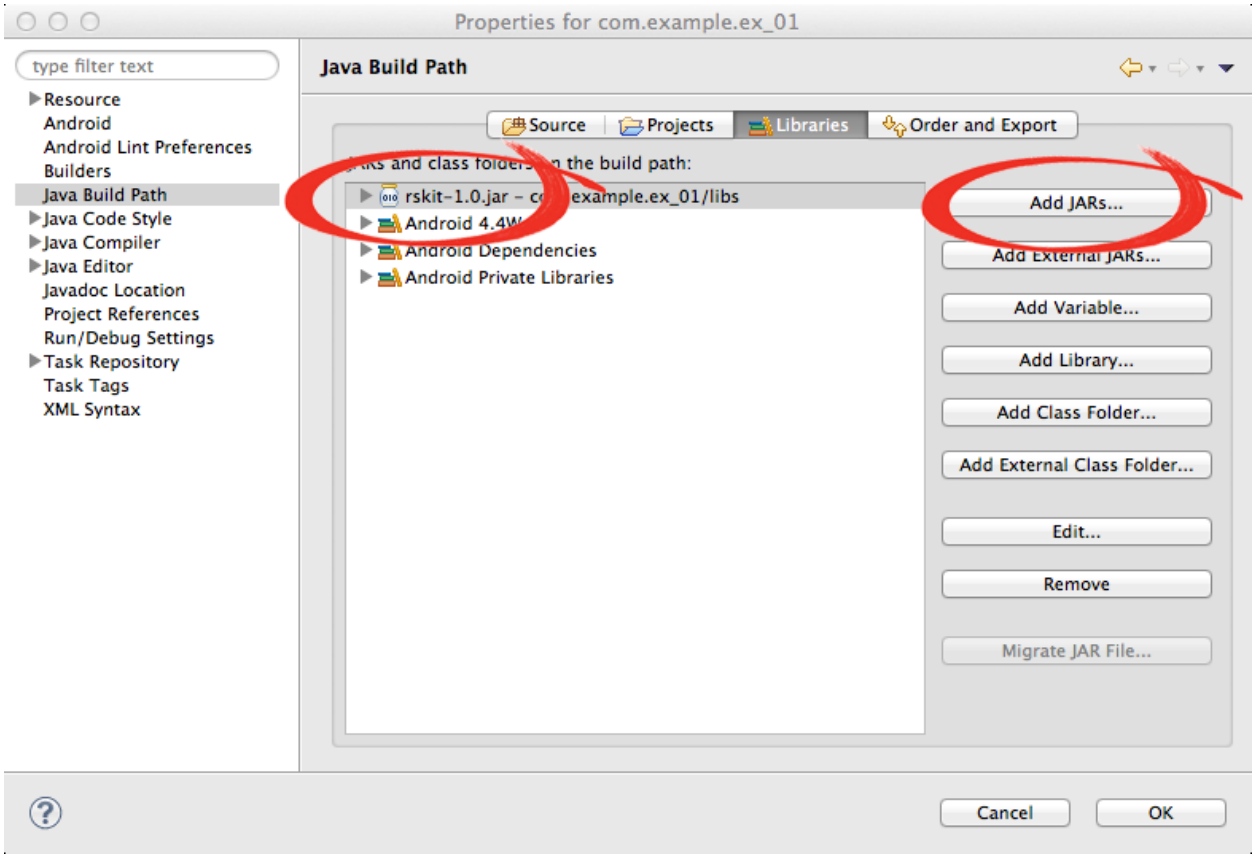
1.) **locate the rskit-1.x.jar file provided as part of your Retention Science SDK**

    If you are provided the Java files directly, then import them as you would any other collection of classes, and skip to #5.

2.) **Place the file in your library folder**

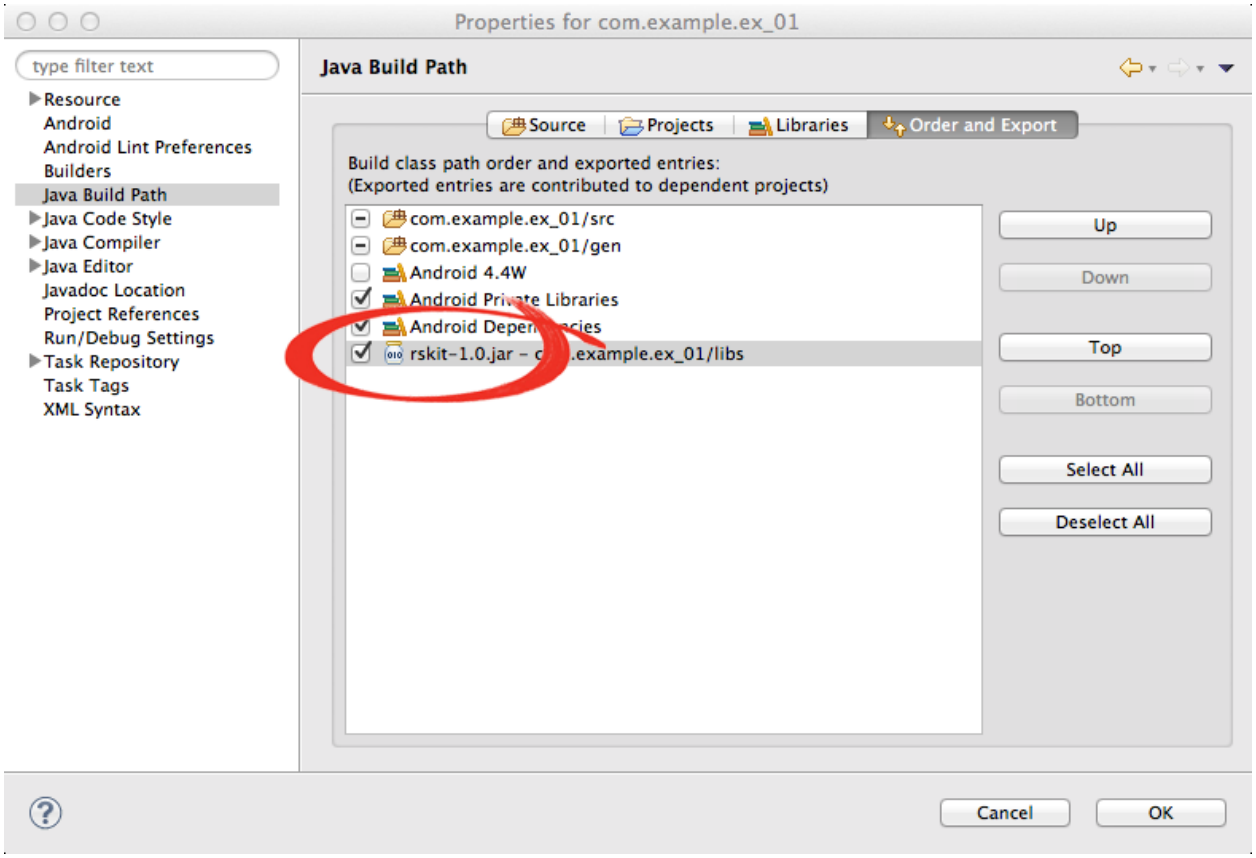
    Commonly there is a folder called “libs” in your project, but depending on your configurations it may be in another location, or you may want to place it somewhere else.

3.) **Add the JAR file to the Java build path “libraries” list**



You can access the above menu by selecting your project, and right clicking / control - clicking. Towards the bottom of the menu will be the properties option. Selecting it will open up the above image.

4.) **Add the JAR file to the list of exported files.**



5.) Add the required permissions to your AndroidManifest.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ex_01"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="19" />
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.READ_PHONE_STATE" />
        <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application
```

The Following permissions need to be added to the file, as the above screenshot.

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET"></uses-permission>

Now, the RSClient class should be accessible within your project. To completely implement the RSClient, we need to make sure the init methods are called prior to any event calls. You may have other preferred implementations for library initializers being called prior to complete app initialization, but the following is the suggested implementation.

6.) Implement an extended Application class with the needed RSClient init

MainApplication | com.example.ex_ | com.example.ex_ | com.example.ex_ | com.example.ex_ | cor

```
package com.example.ex_01;

import com.retentionscience.client.RSClient;

public class MainApplication extends Application{

    public void onCreate(){
        RSClient.initializeWithSiteId("152", this);
        RSClient.DEBUG_enableDeveloperMode();
        RSClient.enableDebugLogs();
        RSClient.enableVerboseLogs();
        RSClient.setUserId("newuser1235");
        RSClient.setDeviceIdSource("067e6162-3b6f-4ae2-a171-2470b63dff00");
        super.onCreate();
    }
}
```

The core initialization only requires the following:

RSClient.initializeWithSiteId(“XXXX”, this);

The other calls in the above images are not required, and depend on your implementation. (also note that there are debug and developer functions called, also not needed).
“XXXX” is the Id provided to you by Retention Science.

7.) Point the AndroidManifest to the Application class you created

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ex_01"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="19" />
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.READ_PHONE_STATE" />
        <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application
        android:name="com.example.ex_01.MainApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
```

Now your application will always call the RSClient initialize function when the app is Initialized.

Part 2 - :::::How the Retention Science SDK works:::::

The RS SDK has a main queue that events are added too (via the "*track*" methods). A single low priority & spaced out thread will pull items off the queue as a small threadpool has available threads for processing and the HTTP POST request. Some events are also cached to disk (based off of HTTP failure & an extended queue length) and are re-sent on the next app restart.

Some of the following methods allow custom configurations of the SDK, but generally speaking the only methods used will be "*initializeWithSiteId", "track", "setUserId"* and *"setDeviceIdSource".*

Part 3 - :::::Using the Retention Science SDK:::::

High Level:

The core or the RSClient is made up of two methods, "*initializeWithSiteId*" & "*track*". "*initializeWithSiteId*" is called once and must be called before any track calls are made. Once its called, the app can start making "*track*" calls with various parameters. If that is all you need as an App developer there are no other methods required.

Method Breakdowns:

RSClient.initializeWithSiteId(siteId, application)

The siteID is provided by Retention Science, and the Application instance ideally is passed from the custom Application class (as mentioned previously in the implementation manual.

RSClient.track(action) - (and other various parameters)

The track method has 3 parts (2 optional), the event action string is always required and is the first parameter. The other optional parameters are a combination of a metadata packet (either in JSONObject or Bundle form) & a Location object. The calls are not instantaneous, and the details of the logic implementation will be covered after the method details.

RSClient.setUserId(userId)

Used to define a userId depending on the app implementation. This userId is defined by the app developer and is provided as an optional parameter to add to all events.

RSClient.setDeviceIdSource(source)

This is used primarily to track marketing. This will mark all events to include a source parameter for App developer tracking.

RSClient.useGeneratedUUIDs(useGenerated)

Getting UUIDs on Android can be a bit tricky. By default the SDK generates UUIDs from the android DeviceID, but this may not be available or desired in many cases. Choosing generated UUIDs will create a UUID for the device and save it to disk. This will be entirely random, but also will not be the same if a user uninstalls / re-installs the app.

RSClient.getUniqueDeviceId()

Provides access to the UUID used by the Retention Science SDK

RSClient.enableDuplicateEventChecking()

This will check (based on a hash on the event data) if the event being added to the SDK event queue is already in the queue, and prevent additions if it is already there. Please note this will have a small performance impact.

RSClient.setMaxQueueSize(maxSize)

This limits the amount of events in the event queue. By default it is unlimited. If you are firing off a very large amount of events, this may be worth considering, but the queue should have little impact on performance even if it is large.

RSClient.setMaxCachedEvents(maxEvents)

This is used if you wish to adjust the amount of events cached on disk, which will be restored and resent in the event of extended internet disconnection. If your app is heavily web dependent, touching this won't do much, more applicable for "light" web dependent apps.

RSClient.setCachedInQueueTreshold(queueTreshold)

If a collection of events are filling the queue (haven't been sent to the Retention Science servers yet) it is possible that those events will be lost of the user looses internet connections, exits the app, and it gets evicted. This adjusts how many events can be in the queue before new events are also cached to disk.

RSClient.setSessionDuration(maxDelay)

This adjust the event delay (time between events) for those events to be considered part of the same session. If the events are spaced out beyond this, they will generate new session keys on firing.

RSClient.setLoopIntervalSpacing(ms)

This simply reduces the thread time spent logging events. This should only be adjusted if you are sending a large volume of events in occasional intervals & these intervals demand very high CPU. It is highly unlikely this will need to be adjusted, but it it is provided for convenience.

RSClient.setSocketTimeout(socketTimeout)

Used to adjust how many milliseconds an HTTP event will stay open before it is considered a failure and returns an error. Any calls affected by this will be added back into the queue and retried.

RSClient.retryEventsCachedOnDisk()

This is automatically called as part of the "*initializeWithSiteId"* call, and it is advised not to trigger it manually unless you have specific needs related to calling cached events.

RSClient.isInitialized()

A simple boolean to check if the Retention Science SDK has been initialized. All "track" calls do this check internally, so there is no need to call this externally.

RSClient.enableDebugLogs()

By default no info is passed to LogCat. Enable this for initial setup / development.

RSClient.enableVerboseLogs()

Verbose logs will write the entire JSON string to logcat when a call comes back successfully. Note that debug logs must also be enabled.