DEVICE COLLECTOR SDK ANDROID 2.5 GUIDE

Revision: April 28, 2014

Contents

Introduction

A Software Development Kit, also called a “devkit” or SDK, is a set of development tools that allows software engineers to create customized applications for a particular software package, software framework, hardware platform, or operating system. This allows developers to create applications specific to their business needs that will interact with the product or products developed by the SDK creators.

Device Collector SDK Android

The Device Collector SDK Android provides a java jar file which can be used to perform Device Collection interaction with Kount for native Android applications. The SDK includes a collection of development tools that allow you to build your own customized applications in order to submit information to the Risk Inquiry System (RIS) server.

Additional information regarding SDKs can be found in the RIS Software Development Kit Guide.

Using the Device Collector SDK Android

The following information will assist you in acquiring, adding, and and configuring the SDK in a project.

Adding the SDK in a Project

To use the Device Collector with native Android applications, perform the following steps:

1. Obtain the latest version of the SDK from a Merchant Services representative.
2. Unpack the SDK zip file.
3. Copy the lib/devicecollector-sdk-<version>.jar into the "libs" directory of your Android project located at the same level as the project's "src" directory. Create the directory if it doesn't already exist in your project.

Minimum Requirements

The following minimum requirements are needed to utilize the Android Device Collector:

- Min SDK API Level - 8
- Application with the following permissions in the Android Manifest File:

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

The <uses-permission> elements must be children of the <manifest> element.

NOTE: A minimum requirement is what is required to run the software successfully.

- To collect Geo Location and Device ID information (which is on by default), request the following permissions in the manifest:

```
<manifest ...>
          <!-- This is will enhance the device ID -->
         <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
   <!-- pick one of these two for Geo Location (FINE is preferred) -->
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
</manifest>
```

In order to use this SDK, the newest version of the mobile application must be downloaded. Although it is not required to activate the location service, it is highly recommended. This does not involve the actual usage of the location services. If a user turns their GPS or other Location Services off, the Android SDK will just note that the Location Service was not available.

In the Android SDK, if the user turns off Location Services or the Application does not REQUEST the correct permissions at install time in the manifest, the SDK will not use the Location Service. However, if the merchant wants to forcibly prevent Location Services from attempting to run, they can notify their merchant services representative and request to "SKIP" that collector. This is NOT recommended, especially if the app already requests COARSE or FINE LOCATION permissions. However,these can be turned off programmatically, by adding the Enum CollectorEnum.GEO_LOCATION to the EnumSet and inserting it in the datacollectors.skipCollectors() method. See the Reference Implementation included in the SDK for examples.

Referencing Documentation

The following documents are references in this guide. You should read and understand the content in these documents before proceeding:

- **Technical Specifications Guide: Data Collector and Risk Inquiry System (RIS)** - This guide covers in detail information about the Data Collector and RIS. Even if the merchant is not implementing a checkout page within a web browser, the Merchant URL Image with the 302 redirect to the Kount Server URL is still required, and should be setup prior to implementing the Mobile SDK Library. This document provides the information necessary to setup both the Data Collector Merchant URL and the RIS request used to evaluate the purchase request.
- **The Android API:** (http://developer.android.com/guide/components/index.html)

Implementing the SDK Inside Your Application

Implementing the SDK inside your application consists of two basic steps:

1. Creation and Configuration
2. Calling the collect() method

This starts the collection process.

The **Creation** and **Configuration** process involves creating the com.devicecollector.DeviceCollector object, and setting up the object with the following values:

1. **Merchant ID** - This is provided to you by Kount.

2. **Collector URL** - This is your Data Collector URL the 302 redirects to the Data Collector (something like : https://mysite.com/logo.htm). For more information on the Collector URL, see the Data Collector documentation.

3. **The DeviceCollector.StatusListener implementation** - To listen for status changes by the collector. The SDK collects data in the background after collect() is called. To learn the state of the collector, pass in an implementation of this listener to get updates. A listener is optional, and the collector will operate properly if a listener is not set.

4. **Skip Collectors** - This can be used to explicitly skip collectors that you do not want the Device Collector to use. Currently you can choose to skip any collectors in the following list. This must be done PRIOR to calling collect():

    - Geo Location

When you want the collector to gather information about the device, once the DeviceCollector is created and configured, call the collect() method and pass in the sessionId . The collect() method spawns a background thread to collect device information. The collect() method should not be called until a transaction is considered "imminent." For examples around what "imminent" means, see the section on using the collect() method.

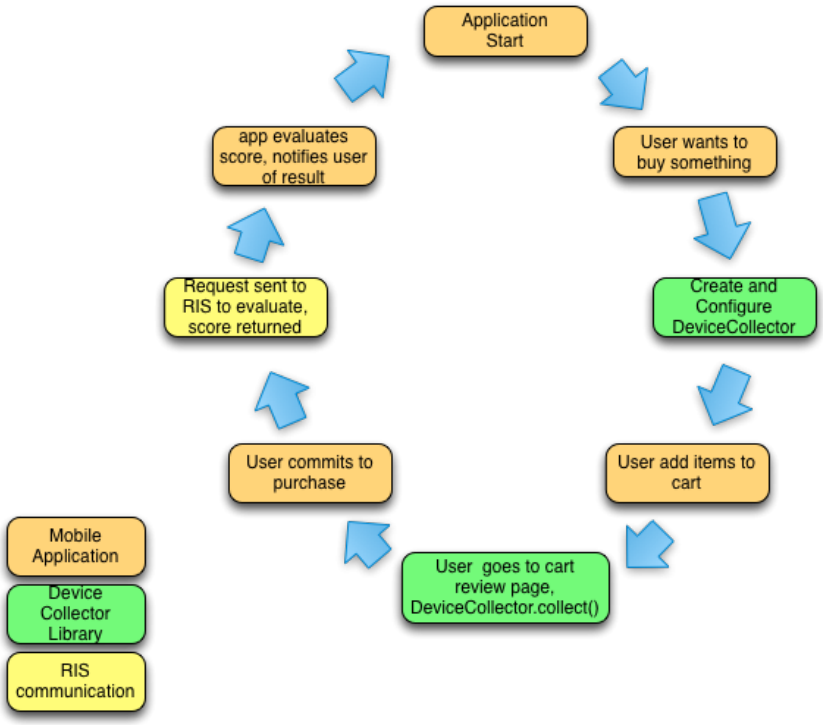
When the DeviceCollector collect process starts, it fires the onCollectorStart() method of the listener if a listener exists.

Information gathered by the DeviceCollector is only germane to the device and its environment and gathers no information about the user.

Once collection completes, the onSuccess() method fires on an implemented listener. If the DeviceCollector failed, the exception onError(DeviceCollector.ErrorCode, Exception) fires. Succeed or fail, the collector process completes and no further action is taken.

A Step-by-step process of adding the SDK to your application may proceed as follows:

- Implement a call to the library at the appropriate point in your application.
- Import com.kount.devicecollector.DeviceCollector class
- Add DeviceCollector.StatusListener interface to your activity class declaration (or other class that spans Activities) and implement interface methods in your listener class
- Instantiate DeviceCollector
- Register your object to receive protocol notifications (set the listener)
- Configure DeviceCollector (set the merchantId, collector URL, and optionally the skipCollector List)
- When close to "checking out", call DeviceCollector.collect(sessionId)



NOTE: The Application does not need to reach the completion of the libraries collect() method. When the task is complete, the application will be notified.

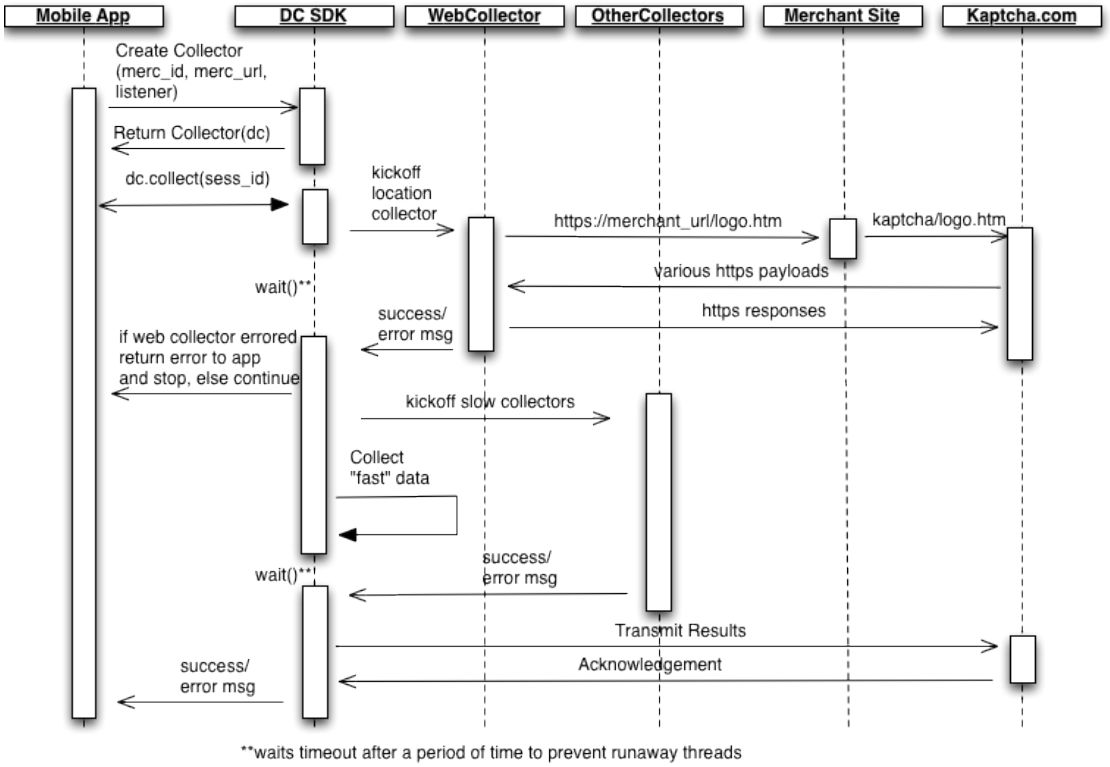
When to Call the SDK Methods

When you invoke the SDK is entirely dependent on how your application works. We can make some suggestions that align with our Web site implementations. Our example implementation is extremely simple and does not cover real world uses. It exists to show what to call and in what order. Actual execution points are application dependent and should not be taken directly from the example.

The DeviceCollector does most of it's work in the background. When it completes, has an error, or times out, it will notify the class passed in that implements the DeviceCollector.StatusListener. Protections have been put in place to make the library monitors itself and will quietly finish if it takes too long. In addition the usage of the DeviceCollector is not mandatory prior to the RIS call, so it can be turned off in the event that System API changes cause the DeviceCollector not to function properly. However, disabling the DeviceCollector will effectively turn off device fingerprinting for the payment transaction that is being assessed by RIS.

Here are some key points to consider when implementation of the library and how we would see it being used in most cases.

The process flows between libraries and servers is as follows:



NOTE: Code examples are not provided in this document. The **Reference Implementation** example project includes all necessary code examples. The following sections will refer you to the **Reference Implementation** as required.

Creating the Collector

The Device Collector can be created at any time before the checkout/transaction request call is made. It's recommended that the collector be created in a way that you can track its state when your application changes. The developer should keep a "reference" to the Collector once the process as started until it is completed (in either Success or Error). The application does not need to wait on the process to continue checkout.

You can create the DeviceCollector at any time, but you can wait to create it until it is needed. However, the DeviceCollector object should be created at a level in the application that does not destroy the object before completion (i.e. a member variable in the Activity or Application level, and not a local veriable or in a subclass not kept at the member level).

Prior to collect() being called, the DeviceCollector can safely be destroyed and recreated without concern. Once collect() has been called, you should keep track of the DeviceCollector instance until the process completes in Success or Error, or your application finishes. You do not need to wait on this process to continue the user's experience, but you do not want multiple instances of this library running for the same transaction.

See the **Reference Implementation** for code examples on both **Fragment** and **non-Fragment** style implementations.

When the collector is actively running it runs in the background as an android.os.AsyncTask

(http://developer.android.com/reference/android/os/AsyncTask.html).

If your application handles destruction and recreation states while active (which most do) you will need to keep track of the Device Collector throughout those states. (i.e. storing the handles to the DeviceCollector and your class that implements the DeviceCollector.StatusListener)

For example, if your application supports configuration changes such as changing orientations (Landscape to portrait) the Android OS destroys and recreates the app unless you specifically override that behavior. If it is not overridden then the OS uses the State Bundles and onRetainNonConfigurationInstance() methods to handle recreation of the application. If your collector is running you should keep a reference to it during this process. For more information on the Android lifecycle, especially configuration changes, see their documentation here:

http://developer.android.com/reference/android/app/Activity.html#ConfigurationChanges

Calling collect()

It is important to collect the most up-to-date information possible just before a payment method is used. In the realm of shopping carts on a web site, this is normally at the checkout step. On the checkout page, a merchant displays order information the user would like to submit and also displays and gathers last minute information like method of payment or billing/shipping information. Work flow models differ, but

generally there is a page that allows the user to confirm their purchase prior to the transaction submission to the processing server. Our suggestion is to execute the collect() method in this review step ("prior to" or "as" the review step is being displayed).

The collection process starts when you call the collect(sessionId) method of the DeviceCollector. This process runs asynchronously in the background and should not stop or interrupt your application.

The collect() call should only happen once per transaction.

If your application resets (i.e. due to an orientation change) you should keep track of whether or not you have already called collect(). The collector will notify your listener, if implemented, of the changes in status as things move along, which you can react to or ignore.

The calling application does not need to wait for the completion of the DeviceCollector. If the DeviceCollector does not complete prior to a call to the Risk Inquiry System (RIS) for the same transaction (session ID), then the DeviceCollector information will not be used. This prevents any "gating" effects to the application from the collector.

The collect() method should only be called once per session ID (transaction). **NOTE:** The session id used in the DeviceCollector must be the same that is passed to RIS call when the purchase is being evaluated.

See the **Reference Implementation** for code examples on both **Fragment**and **non-Fragment** style implementations.

The Source for the Example

See the **Reference Implementation** for code examples on both **Fragment** and **non-Fragment** style implementations.

Error Codes

- **NO_NETWORK** - Network access not available.
- **INVALID_URL** - Invalid collector URL provided.
- **INVALID_MERCHANT** - Invalid Merchant Id provided.
- **INVALID_SESSION** - Invalid Session Id provided.
- **MERCHANT_CANCELLED** - Merchant Cancelled by calling finishNow().
- **PERMISSION_DENIED** - Merchant did not grant a needed permission for a collector (i.e. INTERNET).
- **RUNTIME_FAILURE** - Generic runtime failure.


Kount®