Visual Developer Network, Inc.

Kount Device Collector Xamarin Bindings SDK

Version 2.5

Monday December 14th 2015

Introduction

This Software Development Kit is intended to aid in the use of the Kount Collector SDK with the Xamarin Development Environment for both iOS and Android. This allows easy native Xamarin use of the Kount Device Collection API's. A Xamarin Workspace is included which contains two solutions, one for Android and one for iOS.

Both solutions include a bindings source code for the relevant platform and a fully functional demo application, which reports collection information through the Kount API on the relevant platform.

The bindings for both platforms have been designed to be as close to identical as possible to allow for a greater amount of shared code, Areas of differences will be point out in the Platform specific section of this document.

As the Xamarin Bindings are extremely close to the original API's much of the original Kount API documentation is still relevant, please use that for reference as to how the collection process works and for FAQ's.

This document alone and the sample workspace provided in the SDK should be enough alone to have devices reporting using iOS and Android through an application developed with Xamarin.

Using the SDK Device Collector for iOS/Android in Xamarin

Differences between iOS and Android bindings

- The Bindings for Android are located in the **DeviceCollectorBindingsAndroid** namespace.
- The Bindings for IOS are located in the **DeviceCollectorBindingsIOS** namespace.

- The Device Collector object is named **DeviceCollector** on Android and named **DeviceCollectorSDK** on IOS (for the remainder of this document, **DeviceCollector** will be used, it refers to the **DeviceCollectorSDK** in the case of IOS).

- When creating the **DeviceCollector** object on the android you must pass the current android activity. The debugging level is not changeable for Android.

- When creating the **DeviceCollectorSDK** object on iOS, you pass true or false for extended debugging to the console.

- The Android bindings **OnCollectorError** method passes back any java exception if one occurs as well as a **DeviceCollector.ErrorCode** object containing the error information.

- The IOS bindings **OnCollectorError** method passes back an NSError if one was created as well as an integer error code which matches the const values provided iOS SDK documentation.

- Finally the I**DeviceCollectorSDKDelegate** interface is located inside the **DeviceCollector** object and must be prefixed with the object name when specifying the interface.

Setting up the SDK for use

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

- Creation and Configuration
- Calling the **collect**() method

The Creation and Configuration process involves creating the **DeviceCollector** object for Android or the **DeviceCollectorSDK** object for IOS, and setting up the object with the following values:

Merchant ID

This is provided to you by Kount (or CMT).

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.

The IDeviceCollectorSDKDelegate 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 delegate to get updates. A delegate is optional and the collector operates properly if a delegate is not set.

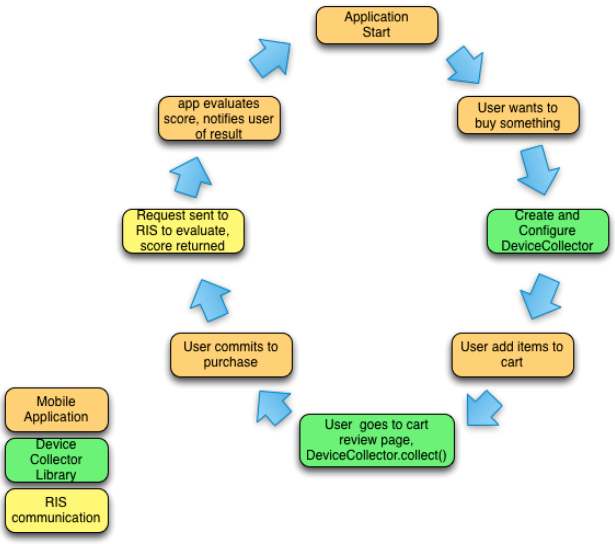
When you want the collector to gather information about the device, once the **DeviceCollector** is created and configured, call the **collect**() method and pass in the sessionId . The **collect**() method spawns a background thread to collect device information. The **collect**() method should not 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 delegate if a delegate 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 **OnCollectorSuccess**() method fires on an implemented delegate. If the **DeviceCollector** failed, the **OnCollectorError(…)** method fires. Succeed or fail, the collector process completes and no further action is taken.

The step-by-step process of adding the SDK to your application may appear as follows:

- Implement a call to the library at the appropriate point in your application.
- Import **DeviceCollector** header
- Add **IDeviceCollectorSDKDelegate** protocol to class declaration
- Implement **IDeviceCollectorSDKDelegate** protocol methods in your class
- Instantiate collector
- Register your object to receive protocol notifications
- Configure collector (merchantId, collectorUrl)
- When close to "checking out" call [collector collect:sessionId]



NOTE: The session id used in the **DeviceCollector** must be the same that is passed to the RIS call when the purchase is being evaluated.

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.

Example IOS code

```
using DeviceCollectorBindingsIOS;

public partial class ViewController : UIViewController, IDeviceCollectorSDKDelegate
{
	const string deviceCollectorURL = "https://tst.kaptcha.com/logo.htm";
	const string merchantId = "160700";
	DeviceCollectorSDK deviceCollector;

	partial void SendDeviceInformation_Click (UIButton sender)
	{
		deviceCollector = new DeviceCollectorSDK(true); // Send true for debug to DeviceCollectorSDK
		deviceCollector.SetDelegate(this);
		deviceCollector.SetCollectorUrl (deviceCollectorURL);
		deviceCollector.SetMerchantId (merchantId);
		deviceCollector.Collect(Guid.NewGuid().ToString("N");
	}

	[Foundation.Export ("onCollectorError:withError:")]
	public void OnCollectorError (int errorCode, Foundation.NSError error)
	{
	}

	[Foundation.Export ("onCollectorStart")]
	public void OnCollectorStart ()
	{
	}

	[Foundation.Export ("onCollectorSuccess")]
	public void OnCollectorSuccess ()
	{
	}

  }
```

Example Android Code

```
using DeviceCollectorBindingsAndroid;

namespace DeviceCollectorSample
{
   [Activity (Label = "DeviceCollectorSample", MainLauncher = true, Icon = "@mipmap/icon")]
   public class MainActivity : Activity, DeviceCollector.IDeviceCollectorSDKDelegate
   {
      const string deviceCollectorURL = "https://tst.kaptcha.com/logo.htm";
      const string merchantId = "160700";
      DeviceCollector deviceCollector;

      void SendDeviceInformation_Click (object sender, EventArgs e)
      {
         deviceCollector = new DeviceCollector(this);
         deviceCollector.SetStatusListener(this);
         deviceCollector.SetCollectorUrl (deviceCollectorURL);
         deviceCollector.SetMerchantId (merchantId);
         deviceCollector.Collect(Guid.NewGuid ().ToString ("N"));
      }

      public void OnCollectorError (DeviceCollector.ErrorCode errorCode, Java.Lang.Exception javaException)
      {
      }

      public void OnCollectorStart ()
      {
      }

      public void OnCollectorSuccess ()
      {
      }

   }
}
```

Reference Implementation Applications and Original SDK documentation

A full reference implementation application for IOS and Android using the Xamarin bindings are accompanying this document.

The full iOS and Android SDK documentation and sample applications are provided along with this document for reference please see the Q&A sections in each document for common Q&A's. The Xamarin bindings implementation mirrors the corresponding iOS and Android implementations almost identically.