Android SDK Guide

This guide provides detailed instructions for integrating the Scanovate Colombia SDK into your Android application, enabling you to collect precise information about events occurring on the device with our COLLECTOR library.

Requirements and Compatibility

Before starting the integration process, ensure your development environment meets the following requirements:

- **Android Studio**: The latest version is recommended for optimal compatibility.
- **Minimum SDK Version**: Android SDK version 21 (Lollipop) or higher.
- **Target SDK Version**: Android SDK version 34 (Android 14) to ensure your app is compatible with the latest Android OS.
- **Compile SDK Version**: Android SDK version 34.

Installation

1. Add the library

Download the "Collector.aar" library and add it to your project's `libs` folder. Ensure you configure your project's `build.gradle` file to include the library as a dependency:

```
dependencies {
// ... your dependencies

// Collector dependencies
implementation 'com.google.code.gson:gson:2.10.1'
implementation 'com.google.android.gms:play-services-location:21.0.1'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4'
implementation 'com.squareup.okhttp3:okhttp:4.11.0'
implementation 'com.squareup.moshi:moshi-kotlin:1.14.0'
implementation 'com.google.android.gms:play-services-safetynet:18.0.1'
implementation 'com.google.guava:guava:31.0.1-jre'

// Optional Huawei location dependency
implementation 'com.huawei.hms:location:6.12.0.300'

// Optional Play Integrity dependency
```

```
implementation 'com.google.android.play:integrity:1.4.0'


// Optional dnsjava dependency in cases we see dns lookup issues
implementation 'dnsjava:dnsjava:3.6.1'


// Collector dependency
implementation files('libs/collector-release.aar')
}
```

This is a small example of how to invoke the method that will launch the library.

""*You need to have location permissions. If you do not have them, these parameters will be omitted and will not be sent to the Collector.*""

This example shows how to start the Collector in the application. The provided URL for sending data to the Collector and the environment or project identifier are defined. Then, the **startCollector()** method is used to initialize the CollectorAgent with the URL and the CID. Finally, the "START" context is sent when starting the session. This is a basic starting point for integrating the Collector into your application, ensuring that you have the necessary location permissions for proper operation. Configuration parameters such as the service **URL** and **CID** can be adjusted according to the specific needs of your project.

```
// Definition of the URL provided for sending data to the Collector
val URL = "URL_PROVIDED_FOR_DATA_SENDING_TO_COLLECTOR" // Definition of the environment or project
identifier
val CID = "ENVIRONMENT_OR_PROJECT_IDENTIFIER"


// Method to initialize the Collector
fun startCollector() {
    // Initialization of the CollectorAgent with the URL and CID
    CollectorAgent.Companion.initialize(application, URL, CID)


    // Sending the "START" context when starting the session
    CollectorAgent.Companion.get().sendContext("START")
}
```

Parameters Explained

- **this.getApplication():** The application context obtained from the activity.
- **URL**: The URL of the collector being initialized.
- **CID**: The environment or project identifier for initializing the collector.

The sendContext method sends the context of the recently performed action or the stage it's currently in. This helps understand what's happening within the session and what action the user took. For example, if the user enters the "Money Transfer" activity, we want you to send us a string "TRANSFER_MONEY" using the sendContext function.

```
collectorAgent.sendContext("TRANSFER_MONEY")
```

To send external input events that are not automatically collected by the Collector library, the `submitExternalInputEvent` method is used. This method is suggested to be used if you have a custom input component, such as selectors or button presses, and you want to ensure that the system verifies that a human user is interacting with the application, not an automated bot.

By sending an external input event with `submitExternalInputEvent`, you can provide details about the action performed by the user, such as the event type (e.g., click or swipe), the ID of the element related to the event (if available), a description of the element, the input type (such as submit button or text input), the source class where the event originated, and any text associated with the event.

```
collectorAgent.submitExternalInputEvent(
            EventType.Click, // Event type, such as Click, Swipe, etc.
            0, // ID of the element related to the event (if available)
            "SetParameters Button", // Name or description of the element
            InputType.Submit, // Input type, such as Submit, Text, etc.
            "SecondActivity", // Name of the source class where the event originated
            "Set Parameters" // Input text associated with the event (if applicable)
        );
```

To configure the CSID and UserID, you use the CollectorSetParameters method. The CSID is new in each session, while the UserID remains constant for the same user.

```
var CSID: String? = null // Customer SessionID
var USERID: String? = null // CustomerID

fun CollectorSetParameters(csid: String, userId: String) {
    collectorAgent.setCSID(csid)
    collectorAgent.setUserID(userId)
    CollectorAgent.Companion.get().sendContext("SETPARAMETERS")
}
```

Logging failed user logins

There is a way to report failed user logins, the backend could match the amount of failed login attempts with the userids that are being used to authenticate to correct the possibility of fraud

happening from the specific device.

```
val collector = CollectorAgent.get()


// failed login using password
collector.sendFailedLogin(uid = userId, method = "password")


// failed login using otp
collector.sendFailedLogin(uid = userId, method = "otp")


// failed login using biometric
collector.sendFailedLogin(uid = userId, method = "biometric")
```

Demo Application

For a comprehensive example, including full source code demonstrating the integration and usage of the Scanovate Colombia SDK, visit our GitHub repository:

Scanovate Colombia SDK Demo App For Android

This demo app provides a hands-on example to help you understand how to integrate and utilize the SDK in your own applications.

Versions

Huawei - Google - a14062024 (Latest)

Huawei - a14062024h (Latest)

Google - a14062024g (Latest)

Revision #9
Created 6 May 2024 15:28:02 by Admin
Updated 25 November 2024 16:00:34 by roger de avila