Computop

the payment people

Android SDK

Integration Guide

SDK Version 1.5.207

last update: **2020-05-15**

Document history 2

Introduction 2

1. Overview 2

1.1 List of supported payment methods 3

2. Requirements 3

3. Installation 3

4. How to implement 4

4.1 Configuration 4

4.2 Authentification 4

4.3 Making a payment 4

4.3.1 Available Payment Methods 5

4.3.2 Payment Data 5

4.3.2.1 WeChat Pay 6

4.3.2.2 PayPal 7

4.3.3 Make the checkout 7

4.4 Support screen rotation 8

Document history

Date	Change
2017-10-01	First Version
2017-05-05	Version 1.2.120
2017-06-12	Version 1.3.130
2017-06-21	Fixed spelling errors
2017-11-03	Add payment additional parameters
2018-01-31	Version 1.3.151
2018-03-20	Version 1.4.182 - Add WeChat Pay
2018-07-02	Version 1.5.207 - Update PayPal using Chrome Custom Tabs

Introduction

1. Overview

This is the documentation for the Computop Android SDK, which describes how to integrate payments in your Android app.
The integration of the SDK is achieved by following a list of steps:

A. Configuration of your merchant account in the Computop Paygate system.
B. Making your Android project runnable with gradle and adding the SDK into your project.
C. Configuration of the SDK by inserting appropriate data retrieved from Computop.
D. Configuration of preferable payment methods.
E. Authentication against the merchant backend.
F. Insertion of the payment data.
G. Checkout.
H. Handling of errors.

1.1 List of supported payment methods

The Android SDK currently supports the following payment methods:

- Credit Cards
- SEPA Direct Debit
- PayPal
- WeChat Pay

2. Requirements

Requirements in order to be able to use the SDK:

Preparation

- Existing merchant account at Computop
- Merchant ID which you will receive from from Computop after creation of the merchant account
- Backend and the belonging URL on merchant side to create and deliver auth token (see chapter 4.2)
- Website and the belonging URL's on merchant side to forward and show the status of a payment process in case of a success, failure or a notify event (see chapter 4.3.2).

Development

- Using gradle as dependency management in your Android project
- Android min SDK Version is 15 (4.0.3)

3. Installation

Add jCenter in the main build.gradle:

```
allprojects {
    repositories {
        jcenter()
    }
}
```

Then add the following line to your app build.gradle:

```
implementation 'com.computop.sdk:lib:<current_version>'
```

and sync gradle.

4. How to implement

You could find a demo on https://github.com/ComputopPayment/Computop-Android-Example
Try the demo in order to see how it works or use the following step by step guideline.

4.1 Configuration

Configure the SDK by importing the Computop class and inserting the configuration parameters you receive from Computop, like *merchantID, authURL*.

The provided merchantID in a strings.xml

```
<resources>
    <string name="computop_merchant_id">XXX</string>
    <string name="computop_auth_url">XXX</string>
</resources>
```

4.2 Authentification

One requirement for the Mobile SDK is it to insert the respective Merchant's URL in order to be able to receive the auth token. The SDK is responsible to retrieve then the token under the hood and use it appropriately when executing payment requests. For more information, see the Paygate documentation.

```
<resources>
    <string name="computop_merchant_id">XXX</string>
    <string name="computop_auth_url">XXX</string>
</resources>
```

4.3 Making a payment

Assumed your app user inserted some products into the basket, typed the shipping address and now wants to make the payment - that means you want to provide/show different payment methods to the user, so that he can choose from.

For that you need to ask the SDK for supported payment methods and then you need to configure the payment data.

First initialize an instance of the SDK, activity should be instance of FragmentActivity.

```
Computop computop = Computop.with(getActivity())
```

4.3.1 Available Payment Methods

To get all the available payment methods you need to call:

```

```

```
computop.requestPaymentMethods()
      .subscribeOn(Schedulers.io())
      .observeOn(AndroidSchedulers.mainThread())
      .subscribe((paymentMethods, throwable) -> {
         if (throwable != null) {
            //manage error
             Log.e(TAG, "PaymentMethods error: ", throwable);
             return;
         }

         //manage the paymentMethods list (List<PaymentMethod>)
      });
```

For a payment method you get the icon and the localized description.

4.3.2 Payment Data

The next step is to select a payment method and then fill payment data with the proper key and values based on the documentation for your payment method (more details on `Payment` are in Paygate documentation).

```
Payment getPayment(@NonNull PaymentMethod paymentMethod) {

   Payment payment  = paymentMethod.getPayment();

   payment.setParamWithKey("TransID", "********");
   payment.setParamWithKey("Amount", "100");
   payment.setParamWithKey("Currency", "EUR");
   payment.setParamWithKey("URLSuccess", "http://*****/api/success");
   payment.setParamWithKey("URLNotify", "https://*****/computop/index.php");
   payment.setParamWithKey("URLFailure", "http://*****/api/failure");
   payment.setParamWithKey("RefNr", "******");
   payment.setParamWithKey("OrderDesc", "Tests");
   payment.setParamWithKey("AddrCity", "Berlin");
   payment.setParamWithKey("FirstName", "Lorem");
   payment.setParamWithKey("LastName", "Ipsum");
   payment.setParamWithKey("AddrZip", "10000");
   payment.setParamWithKey("AddrStreet", "Berlin");
   payment.setParamWithKey("AddrState", "AL");
   payment.setParamWithKey("AddrCountryCode", "DE");
   payment.setParamWithKey("Phone", "01777777124");
   payment.setParamWithKey("LandingPage", "Login");
   payment.setParamWithKey("eMail", "******-accounts@****.com");
   payment.setParamWithKey("ShopID", "1");
   payment.setParamWithKey("Subject", "******-accounts@****.com");
   payment.setParamWithKey("Language", "en");
   payment.setParamWithKey("Channel", "APP");
```

```
    payment.setParamWithKey("MerchantID", "*****");

    return payment;
}
```

The method *setParamWithKey()* will return **true** if the value will be set successfully for the specified key.

The values of `Amount` and `Currency` will be validated during the checkout process. So you have to ensure that you are using valid data.
The currency is the currency you want to use you for the payment. You have to use three characters for currency coed (DIN/ISO 4217), e.g. “EUR”.
The amount is the lowest unit of the currency you are using. That means if you are using EUR as currency the amount needs to be in cents, e.g. an amount of 100 is 1 EUR.

`URLSuccess` and `URLFailure` are the URL’s to which the SDK redirect the status of a payment process. These URL’s normally point to a HTML site on your merchant backend to show the status to the user of your app.
For the rest of the parameters you should take a look into the Paygate documentation. Some of the parameters need to be defined by you and are mandatory.

Now you know the available payment methods and you have configured them. It’s time to show the payment methods to the user.

4.3.2.1 WeChat Pay

When using WeChat Pay as a payment method you also have to add the class `WXPayEntryActivity` which extends from `WeChatPayEntryActivity`.

```
public class WXPayEntryActivity extends WeChatPayEntryActivity {
}
```

This class needs to be placed inside a package called `com.mypackage.wxapi`, where `com.mypackage` is your `applicationId` defined in your app build.gradle.

For example, if your `applicationId` is `com.computop.android.sdk.example`, the class `WXPayEntryActivity` should be placed inside the package `com.computop.android.sdk.example.wxapi`

Besides that, you have to define the following Strings in your string.xml.

```
<string name="computop_wechat_app_id" translatable="false">your wechat app id</string>
```

```
<string name="computop_wechat_mch_id" translatable="false">your wechat merchant
id</string>
<string name="computop_wechat_key" translatable="false">your wechat key</string>
```

4.3.2.2 PayPal

For this payment method, the Chrome Custom Tab is used. To get back from the Chrome Custom Tab to your app after payment is done (or canceled), you have to redirect to a custom uri from your merchant backend.

The flow after payment is done/canceled:

- Computop will call your merchant backend using URLSuccess/URLFailure you specified.
- On your merchant backend you have to redirect from your URLSuccess/URLFailure to the custom uri to get back to our app, e.g.:
    - URLSuccess → merchant://com.shop.demo.PayPalReturn://return
    - URLFailure → merchant://com.shop.demo.PayPalReturn://cancel

It is important to use the values **return** and **cancel** as path segments for success and failure.

In your app's string.xml you have to define the Strings `computop_paypal_intent_uri_host` and `computop_paypal_intent_uri_scheme`. Following the example above, you must define the two Strings as follows:

```
<string name="computop_paypal_intent_uri_host"
translatable="false">com.shop.demo.PayPalReturn</string>
<string name="computop_paypal_intent_uri_scheme" translatable="false">merchant</string>
```

Optional: Besides that you can define the color of the Chrome Custom Tab toolbar in your colors.xml as follows:

```
<color name="colorCctToolbar">#3F51B5</color>
```

4.3.3 Make the checkout

Now that you have configured the SDK, selected the payment method and filled the values for your payment data, you are ready to make a payment.

For the Checkout we are using Retrofit underneath.
Start the checkout by instantiating subscription object (Observable). The `Computop` class is a top-level class that facilitates the payment procedure. It is responsible for validating payment data and instantiating a `WebView` when a new payment is triggered by passing the respective `payment` and `paymentMethod`.

```
Disposable disposable = computop
    .withPaymentMethod(method)
```

```
    .setWebViewListener(() -> {
        Log.i(TAG, "[WebViewClient] onPageFinishedLoading");
    })
    .checkout()
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(o ->
    {
        Log.v(TAG, "Payment received");
    }, throwable -> {

        if (throwable instanceof ComputopError) {
            Log.e(TAG, "Computop Error " + (throwable));
            showErrorAlert((ComputopError) throwable);
        } else if (throwable instanceof InternalError) {
            Log.e(TAG, "Internal Error boolean isPaymentCanceled(): " + ((InternalError)
throwable).isPaymentCanceled());
        } else {
            Log.e(TAG, "Payment error: ", throwable);
        }
    });
```

`Observable<T> subscribeOn(Schedulers.io())` will execute the operation on the background thread. When you subscribe to this observable you can show a loading indicator to the user. To get notified when the Webview is loaded you can set the listener `OnWebViewLoadedListener` and hide the loading indicator.

In case of a successful transaction you will get notified in onNext Consumer.

In case of an error you will receive the response in onError Consumer. You have to handle all types of errors including network errors during the checkout process and give feedback to the user.

4.4 Support screen rotation

If the activity where you are doing the payment using Computop SDK (only for payment methods that use WebView) needs to support screen rotation than you have to handle the configuration change.

To do this you have to add this flag in your AndroidManifest.xml file:

```
<activity android:name=".MyActivity"
          android:configChanges="orientation|screenSize"
          android:label="@string/app_name">
```

In this case you will have to handle the configuration changes yourself for your activity. For more info please check:

- https://developer.android.com/guide/topics/resources/runtime-changes.html#HandlingTheChange
- https://developer.android.com/reference/android/app/Activity.html#ConfigurationChanges