**Zeamster Payment SDK ( Native Android) User Guide for Developers**

# **Table of Content**

# Introduction ……………………………………………………………………………. 3

# Android Payment SDK Uses ………………….…..……………………………… 3

# Introduction

This document is to give a detailed overview on the how to use the Zeamster Android Payment SDK for Native Android Mobile Application.

# Android Payment SDK Uses in Android Apps:

Step-1 Download the latest Android payment sdk library (paymentsdk-1.0.0.aar) file.

Step-2: Create the Android app project and import the library as a project module into project.  
  


Step-3: Add the following line into dependencies sections of the app level build.gradle file.

implementation project(':paymentsdk-1.0.0')

Step-4: Run “Sync Project with Gradle files” from Android Studio file menu option.

Step-5: Rebuild the project.

Step-6: Create an object of the RestServiceClient with required input parameters.

RestServiceClient restServiceClient = **new** RestServiceClient(<*protocol*>, <*hostname>*, <*apiEndpoint>*, **this**);

Protocol = http/https – the protocol being used by api server for processing he transactions

Hostname = api.abc.com - hostname of the API server

APIEndpoint = /v2/transactions - full endpoint of an API

Last parameter being passed “**this**” is the instance of the class where the callback will be received. In case the callback is required at somewhere else instead of the class where the RestServiceClient object is being created then pass the instance of that class as fourth parameter.

Step-8:

1. Now implement the ICallback interface for the class where you will be expecting the callback after transaction API response is returned. This class will be the same class of which we are passing the instance in the RestServiceClient object creation as fourth parameter. In this case, you need to implement two override functions:

@Override  
**public void** onSuccess(String response) {  
 //Add code to show success response of transaction   
}  
  
@Override  
**public void** onFailure(String response) {  
 //Add code to show failure response of transaction  
}

1. If the transaction will happen using the card reader device then it is also required to implement the ICallbackEMV interface by the class where status messages and log responses are expected form the connected device. In this case , you need to implement below override functions:

@Override  
 **public void** deviceScanResponse(String deviceId, String deviceName) {  
 //Add code to hold all the scanned devices.

}  
  
 @Override  
 **public void** deviceScanCompleted() {  
 //Add code to hold result once scan complete.

}  
  
 @Override  
 **public void** deviceConnected(String status) {  
 //Add code to show the device is connected.  
 }  
  
 @Override  
 **public void** deviceDisconnected(String status) {  
 //Add code to show the current connect device got disconnected.  
 }  
  
 @Override  
 **public void** deviceMessage(**final** String message) {  
 //Add code to show the device message on app screen.

}  
  
 @Override  
 **public void** outputLogs(**final** String log) {  
 //Add code to show transaction output logs

}

Note: It is not required to use the same class for implementing the interfaces ICallback and ICallbackEMV. These two interfaces are independent from their use and their implementation is purely based on need. See sample app in case of doubt.

Step-9: Create a map to with developer credentials provided by Zeamster to access the transaction apis and call setRequestHeader with the map created with credentials. Also, if any other header property is required to be set in the request that can also be added to this map (like adding content type to the header request).

HashMap<String, String> requestHeader = **new** HashMap<>();  
requestHeader.put(**"developer-id"**, **"value1"**);  
requestHeader.put(**"user-id"**, **"value2"**);  
requestHeader.put(**"user-api-key"**, **"value3"**);  
restServiceClient.setRequestHeader(requestHeader);

Note: If it is required to add query parameters to the request then create another map and set it on RestServiceClient instance as follows.

HashMap<String, String> queryParamaters = **new** HashMap<>();  
queryParamaters.put(**"param1"**, **"value1"**);  
queryParamaters.put(**"param2"**, **"value2"**);  
queryParamaters.put(**"param3"**, **"value3"**);  
restServiceClient.setQueryParams(queryParamaters);

Step-9: Create another map with all fields required to be sent to transaction API for processing.

HashMap<String, String> payload = **new** HashMap<>();  
payload.put(**"param1"**, **"value1"**);  
payload.put(**"param2"**, **"value2"**);  
payload.put(**"param3"**, **"value3"**);

Step-10: There are two types of transactions that can be performed:

1. **Non EMV transactions** – For non EMV transactions, following steps are required to be performed.
2. Create an instance of TransactionService class with passing restServiceClient object created above as an argument to its constructor. Now, call the processTransaction method and pass the expected arguments as explained below.

TransactionService transactionService = **new** TransactionService(restServiceClient);

transactionService.processTransaction(transactionAction, payLoad, *transactionId*);

transactionAction = one of the actions of TransactionAction Enum class as per requirement.

payload = input parameters required by Transaction API for a type of action to be performed.

transactionId = required, when the action is to be performed on an existing transaction.

1. Now receive the response, which will be returned to onSuccess() or onFailure() functions implemented for the interface ICallback.
2. **EMV transactions** – For EMV transactions, following steps are required to be performed.
3. Create an instance of EMVTransaction class with passing restServiceClient object created above as an argument to its constructor along with other required arguments as mentioned in below code snippet.

EMVTransaction **emvTransaction** = **new** EMVTransaction(restServiceClient, **this**, getActivity(), ReaderInfo.DEVICE\_TYPE.***DEVICE\_VP3300\_BT***);

“**this**” will be the instance of the class where the callback response from device is expected.

Third argument will be the current activity.

Fourth argument is the Device Type as per need for the target device to be used as medium for transaction. Many other device types can be found in the ReaderInfo.DEVICE\_TYPE enum class. Use any of those as and when required.

1. Implement the ICallbackEMV interface for the class of whose instance we are passing as the second argument while creating the instance of EMVTransaction class. Refer above code snippet where we are creating the EMVTransaction class object.
2. Now call any scanForDevices() method out of 3 of its types available, as per need.

**emvTransaction**.scanForDevices(); (default search time of 10 seconds)

**or**

**emvTransaction**.scanForDevices(timeout\_in\_miliseconds);

or

**emvTransaction**.scanForDevices(device\_name\_or\_matching\_substring, timeout\_in\_miliseconds);

1. Device scan responses are returned to deviceScanResponse() method of the ICallbackEMV interface implementing class. Use this method to do something as per need with each result returned by the BLE Scanner. This method receives two results, device name and device address.

**Note**: in any case, if user want to stop the scanning of device here is the method which can be used to stop the scanning:

**emvTransaction**.stopScanForDevices();

1. Once expected card reader is found, use connectDeviceByName() or connectDeviceByAddress() method of the emvTransaction instance to connect the card reader device.

**emvTransaction**.connectDeviceByName(deviceName);

or

**emvTransaction**.connectDeviceByAddress(deviceAddress);

Note- Android Payment SDK library also has an additional response listener method deviceScanCompleted() which can also be used if required.

1. Once device is connected successfully a response is received by the deviceConnected() method of the ICallbackEMV interface implementing class.
2. If device gets disconnected deviceDisconnected() method of the ICallbackEMV interface implementing class receives a response for the same. It can be used to take action as per need.
3. Now once everything is setup, use the credit/debit card and card reader device to gather card track data and call any of the following methods. Any of these can be used to call on emvTransaction with the required parameters action and payload as inputs.

* **performEMVSale()  
    
  emvTransaction**.performEMVSale(*transactionAction*, *payload*);
* **cancelEMVSale()**

**emvTransaction**.cancelEMVSale();

**Note**: Current SDK support two EMV transaction type **: SALE and REFUND**;   
In order to perform both transaction, we just need to call the same performEMVSale() function with payload and applicable action type.

Action and Payload inputs will be similar of non-EMV transactions. For the details on what all parameters should be sent can be found on Zeamster Transaction API documents. Here is one of the documentation link:  
<https://docs.zeamster.com/developers/api/endpoints/transactions>

1. All the current status of transaction will be received by deviceMessage() method of the ICallbackEMV interface implementing class.
2. All the processing logs of the transaction will be received by outputLogs () method of the ICallbackEMV interface implementing class.
3. If user wants to disconnect the current connected IDTech device, then just use the below sample code:

**emvTransaction**.disconnectDevice();

Step-13: Once the response is received take the appropriate action after parsing the json string.

Initial Device Setup:

All the step will be same as EMV transaction just instead of calling the **performEMVSale()** , we need to call below function as:

**emvTransaction**.runInitialDeviceSetup();