**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:

1. Download the latest Android payment sdk library (paymentsdk-1.0.0.aar) file.
2. Create an Android project and import the paymentsdk-1.0.0.aar file as a module into the project. Go To File 🡪 Project Structure. Click on **Dependencies** and click the + to add a new one. Select **JAR/AAR Dependency**. Enter the full path to the .aar file. Select **Implementation** as the configuration for the dependency.

Alternatively, you can drag the .aar file to your project’s libs folder, and include it using the dependency method above.

1. Run “Sync Project with Gradle files” from Android Studio file menu option.
2. Rebuild the project.
3. Create an object of type 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 the transactions

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

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

The last parameter passed, “**this**”, is the instance of the class where the callback will be received. If the callback is required somewhere else instead of the class where the RestServiceClient object is being created then pass the instance of that class as fourth parameter.

1. Add the callback methods:
2. 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 add “implements ICallback” to the class definition and 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 you are using a card reader, implement the ICallbackEMV interface on the class where status messages and log responses are expected by implementing the following callback methods. Ensure that you include the following import statement as well:

import com.zeamster.paymentsdk.transaction.ICallbackEMV;

@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: You do not have to use the same class for implementing the ICallback and ICallbackEMV interfaces.

1. Create a map to the developer credentials provided by Zeamster, and call setRequestHeader with that map. Any additional headers necessary can be included in this map as well (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 you need to provide additional query string parameters, create a separate HashMap and include those as follows:

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

1. Create another map with all fields required to be sent to the transaction API for processing.

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

1. There are two types of transactions that can be performed:
2. **Non EMV transactions** –
3. Create an instance of the TransactionService class, passing the restServiceClient object to its constructor. Next, call the processTransaction method:

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

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

transactionAction = one of the actions of TransactionAction Enum class

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. Next, process the response, which will be returned to the onSuccess() or onFailure() functions you implemented for interface ICallback.
2. **EMV transactions** –
3. Create an instance of the EMVTransaction class passing the restServiceClient object as an argument to its constructor along with the other required arguments. Note that currently the SDK only support DEVICE\_VP3300\_BT devices.

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

“**this**” is the instance of the class where the callback response from the reader is expected.

The third argument should be the current activity.

The fourth argument is the Device Type of the card reader. The defined device types can be found in the ReaderInfo.DEVICE\_TYPE enum class.

1. Implement the ICallbackEMV interface for the class we passed as the second argument while creating the instance of the EMVTransaction class.
2. Call the scanForDevices() method to discover the reader. It has 3 overloads: choose the one that fits your situation the most:

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

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

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

1. The response is returned to the deviceScanResponse() method that you implemented earlier in this process. 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 you have identified your reader from the results, call either connectDeviceByName() or connectDeviceByAddress() to connect the card reader.

**emvTransaction**.connectDeviceByName(deviceName);

**emvTransaction**.connectDeviceByAddress(deviceAddress);

Note: the Android Payment SDK library also has an additional response listener method named deviceScanCompleted(), which can additionally be used to handle the scan results.

1. Once a successful connection has been made to your reader, the deviceConnected() callback method will be notified.
2. If your reader gets disconnected, then the deviceDisconnected() callback method will be notified.
3. After making a successful connection, you can use the following two methods on the emvTransaction object to start an EMV transaction or cancel it.

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

**emvTransaction**.cancelEMVSale();

**Note**: The current SDK supports only two EMV transaction types**: SALE and REFUND**.

Action and Payload inputs for each transaction type will be similar to the ones for non-EMV transactions. For the details on which parameters to send, check the Zeamster Transaction API documentation at:   
<https://docs.zeamster.com/developers/api/endpoints/transactions>

1. You will be notified of the current status of the transaction in the deviceMessage() callback method.
2. All processing logs from the transaction are passed to the outputLogs() callback method.
3. If at any time you would like to disconnect from the EMV reader, call the following method:

**emvTransaction**.disconnectDevice();

1. Once the response is received take the appropriate action after parsing the json string.

Initial Device Setup:

Perform all the steps from the EMV Transaction section above, but instead of calling the **performEMVSale() method**, call the following one:

**emvTransaction**.runInitialDeviceSetup();