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

# **Table of Content**

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

# IOS Payment SDK Uses ………………….…..………………………………..…… 3

# Introduction

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

# IOS Payment SDK Uses in IOS Apps:

Step-1 Download the latest IOS payment sdk framework file.

Step-2: Follow below steps to import the framework into project.

* Select project and Go to **Build Phases>Embed Frameworks.**
* Add PaymentSDK.frameworks from downloaded location as below  
    
  
* Go to **Build Phases>Copy Bundle Resources** and add the IDTECH.bundle(available inside PaymentSDK.frameworks)file as below  
    
  
* Go to **Info section** and set below details:
  + Privacy – Bluetooth Peripheral Usage Description
  + Privacy – Bluetooth Always Usage Description  
    

Step-3: Build the project.

Step-4: Create an object of the RestServiceClient with required input parameters.

let restServiceClient = RestServiceClient(apiProtocol:apiProtocol, apiHostName:apiHostName, apiEndpointPath:apiEndpointPath)

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

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

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

Step-5: 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).

var transactionHeaderDict : [String : String]!  = [:]

transactionHeaderDict["param1"]="Value1";

transactionHeaderDict["param2"]="Value2";

transactionHeaderDict["param3"]="Value3";

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

var transactionQueryParamDict : [String : String]!  = [:]

transactionQueryParamDict["param1"]="Value1";

transactionQueryParamDict["param2"]="Value2";

transactionQueryParamDict["param3"]="Value3";

Step-6: Create another map with all fields required to be sent to transaction API for processing.

var transactionParamDict : [String : String]!  = [:]

transactionParamDict["param1"]="Value1";

transactionParamDict["param2"]="Value2";

transactionParamDict["param3"]="Value3";

Step-7: 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. Set the http request header details for restServiceClient object. Once it is set successfully then only proceed:

 restServiceClient.setHTTPRequestHeaders(headerParams: headerDictionary! as NSDictionary)

1. 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.

let transactionService = TransactionService(restClient: restServiceClient)

transactionService.processTransaction(transactionAction: transactionActionType, paramMap: parameterDictionary,

                transactionId: transactionId)

                { (responseJSON) in

                    DispatchQueue.global().async(execute: {

                        DispatchQueue.main.sync {

                            self.showTransactionResult(responseContent: responseJSON)

                        }

                    })

                }

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

paramMap = 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.

Here is the sample code:



1. Once the transaction complete with success or failure, response will be there in processTransaction() function completion block, now user can add required to code to show the output as we have done in above sample code.
2. **EMV transactions** – For EMV transactions, following steps are required to be performed.
3. Create the RestClientService object with appropriate parameter and set the requested header details, once its done successfully then proceed further.
4. Create an instance of EMVTransaction class with passing restServiceClient object created above as an argument to its constructor as mentioned in below code snippet.

let restServiceClient= RestServiceClient(apiProtocol:apiProtocol as! String, apiHostName:apiHostName as! String, apiEndpointPath:  apiEndpointPath as! String)

        if(restServiceClient.setHTTPRequestHeaders(headerParams: transactionHeaderDict as NSDictionary)) {

            emvTransaction = EMVTransaction(restClient: restServiceClient)

            emvTransaction.delegate = self

            emvTransaction.setDeviceType(type: DeviceType.IDTECH\_VP3300BT)

        }

After creating the emvtransaction object , set it delegate as **self** and then set the device type.

1. To perform EMV transaction, we need to implement the EMVUIDelegate where we are calling the EMV transaction. In this case we need to implement below function for EMVUIDelegate:

func deviceMessage(message: String!) {

        DispatchQueue.main.async {

//Add code to show device LCD message

        }

    }

    func deviceScanResponse(deviceID: String!, deviceName: String!) {

//Add code to hold the device scan response and connect applicable device as below

        emvTransaction.connectDeviceByName(name: deviceName)

    }

    func deviceConnected() {

        DispatchQueue.main.async {

            //Add code to show the device is connected.

        }

    }

    func deviceDisconnected() {

        DispatchQueue.main.async {

            //Add code to show device is disconnected.

        }

    }

    func outputLogs(logs: String!) {

        DispatchQueue.main.async {

// Add code to show transaction output logs

        }

    }

    func transactionResponse(data: String!) {

        DispatchQueue.main.async {

            //Add code to show the transaction response.

        }

    }

1. Now call scanForDevices() method.

emvTransaction.scanForDevices()

1. Device scan responses are returned to deviceScanResponse() method of the EMVUIDelegate delegate implementing class. Use this method to do something as per need with each result returned by the BLE Scanner. This method receives device name.
2. Once expected card reader is found, use connectDeviceByName() method of the emvTransaction instance to connect the card reader device.
3. Once device is connected successfully a response is received by the deviceConnected() method of the ICallbackEMV interface implementing class.
4. If device gets disconnected deviceDisconnected() method of the EMVUIDelegate delegate implementing class receive a response for the same. It can be used to take action as per need.
5. 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(jsonPayload: transactionParamDict)

* **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 which should have the **action** field which need to be performed like ‘sale’ or ‘refund’.

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 EMVUIDelegate delegate implementing class.
2. All the processing logs of the transaction will be received by outputLogs () method of the EMVUIDelegate delegate 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.initialDeviceSetup()

Note: While doing initial device set up call End point path should be:  
apiEndpointPath:"/devices/idtechinitialize"