Openpay Integration

Version 1.0.0

SDK Integration with Shop System

Table of Contents

1. Overview

This document describes the technical integration of Openpay link cartridge.

Openpay is a next generation payment solution that allows customers to buy now and pay over a flexible timeframe without interest, giving them added cash flow confidence. Creating a seamless omni channel experience (in-store, online and through our app), we service several brands across multiple industries including retail, healthcare, automotive & home improvement.

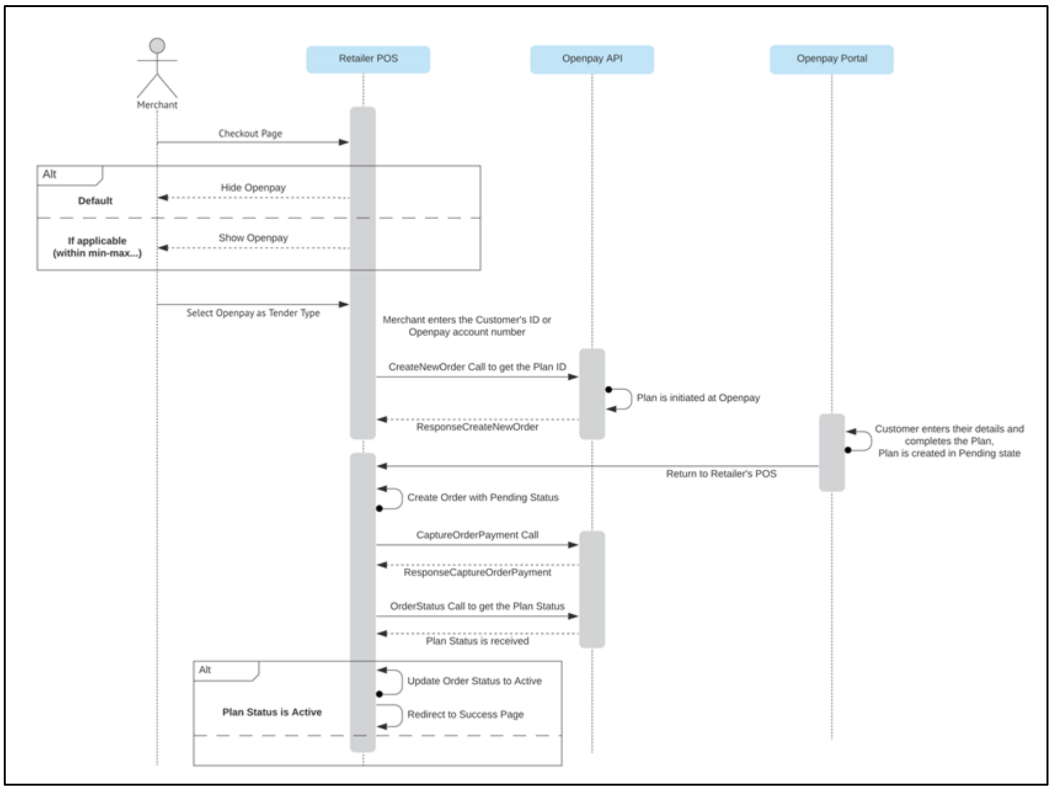
Our customers are empowered to live their best life; with more choice, time and flexibility.

The Openpay SDK is created with multiple layer approach which is further used to integrate with various Java shop systems.

1. Payment SDK includes multiple layers business layer, configuration layer, API layer as described below: -
    - **a.** **Shop System: -** Java based shop system layer interacts with business layer of SDK, by sending request and do further implementations on getting the response back. For more detailed information how to integrate SDK with shop system refer to section 2.2.1
    - **b.** **Business Layer: -** Business layer of SDK includes logic to create request data for each API method and make a call to API, send request and get the response back. This layer is created in generic way to be consumed by any Java based Shop System.
    - **c.** **Configuration Layer: -** This is part of business layer of SDK includes three configuration files named as merchantMapping.ini, mappingShopConfig.ini and mappingApiConfig.ini under resources folder. But as we use SDK in form of jar these files are present at root level of jar.This layer basically handles all configurations like merchantConfig handles authentication related properties. Mapping shop and API config files handle attribute names , method names etc.
    - **d.** **API Layer: -** This layer consists of making call to third party System (Openpay Payments) by sending required headers and body in json format and get the response back. Conversion of request to json, handling of authentication, connection creation all is handled in this layer.

2. Openpay Payment SDK is used in form of JAR file in respective shop system. This JAR file is provided to you by merchant.
3. SDK integration is done for SAP Hybris shop system and all related information is discussed in below chapter.

4. The below diagram represents flow between merchant shop and payment API. This is create order flow diagram which states how order data is sent from shop and using Openpay as checkout payment option order is successfully places in Openpay.


Merchant
Retailer POS
Openpay API
Openpay Portal
Checkout Page
Alt
Default
Hide Openpay
If applicable (within min-max...)
Show Openpay
Select Openpay as Tender Type
Merchant enters the Customer's ID or Openpay account number
CreateNewOrder Call to get the Plan ID
Plan is initiated at Openpay
ResponseCreateNewOrder
Customer enters their details and completes the Plan, Plan is created in Pending state
Return to Retailer's POS
Create Order with Pending Status
CaptureOrderPayment Call
ResponseCaptureOrderPayment
OrderStatus Call to get the Plan Status
Plan Status is received
Alt
Plan Status is Active
Update Order Status to Active
Redirect to Success Page

1.1 Compatibility

SDK is compatible with java version java 8 to java 11

1.2 Limitations, Constraints

None

1.3 Privacy, Payment

None

2. Integration of SDK in Shop System

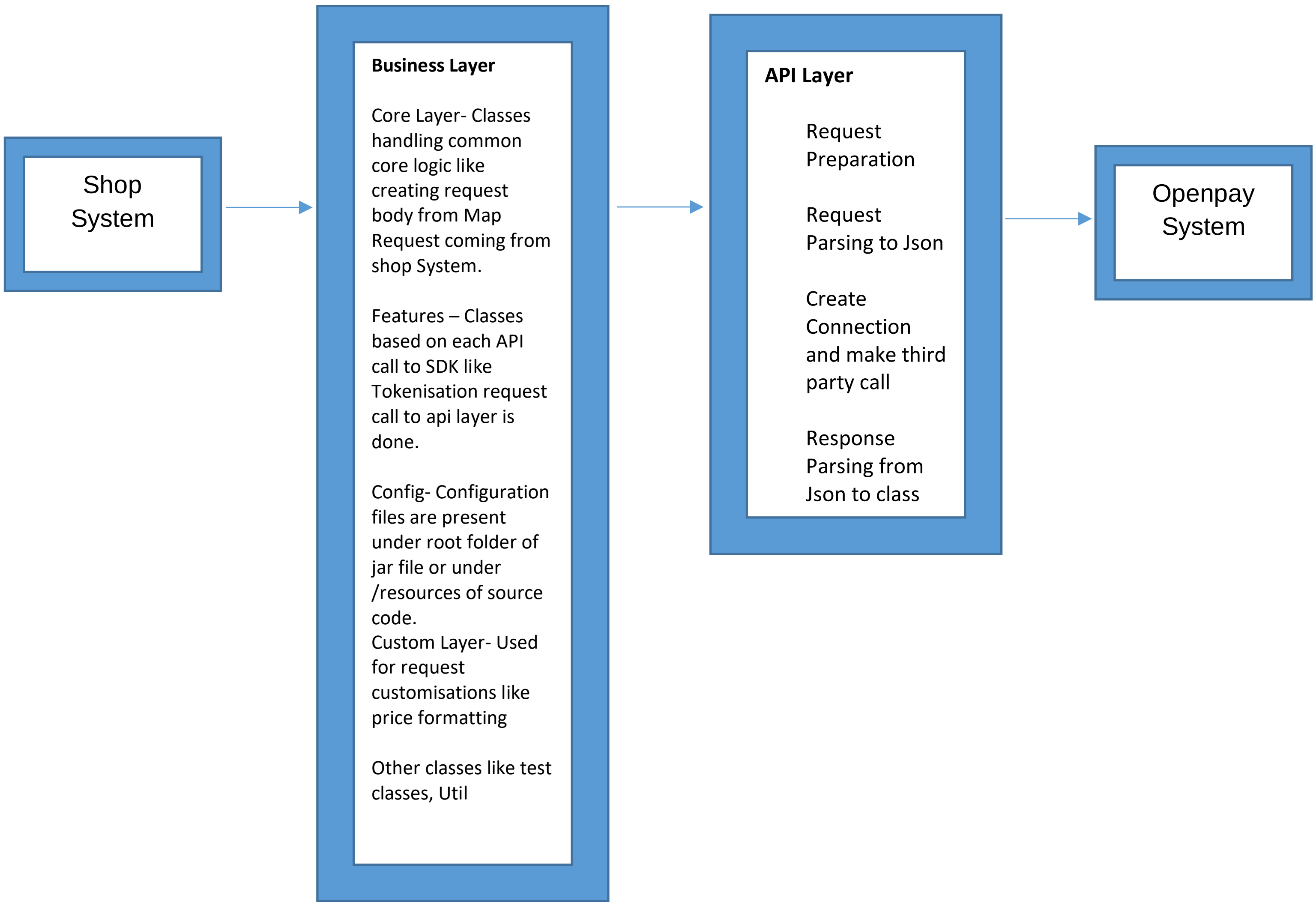
2.1 Installation

SDK is maven-based application created to consume all payment methods of Openpay. SDK is delivered in JAR format. That jar file of SDK will include all external libraries along with the layers we developed to consume the services and configuration files.

Include jar file to specific location of your shop system with other external dependencies to consume the jar methods.

2.2 Introduction to Layer approach

As already provided glimpse of SDK layer approach in overview section, here we will discuss about each layer in detailed way



Shop System
Business Layer
Core Layer- Classes handling common core logic like creating request body from Map Request coming from shop System.
Features – Classes based on each API call to SDK like Tokenisation request call to api layer is done.
Config- Configuration files are present under root folder of jar file or under /resources of source code.
Custom Layer- Used for request customisations like price formatting
Other classes like test classes, Util
API Layer
Request Preparation
Request Parsing to Json
Create Connection and make third party call
Response Parsing from Json to class
Openpay System

2.2.1 Shop System

This layer refers to the specific shop system which we have to integrate with Openpay payment system. SDK will be provided in form of JAR.
Import JAR file in shop system where other external dependencies are located. All external libraries used in SDK is also included in JAR.

<u>Integration of SDK with shop system</u>

1. To consume all API's from SDK (refer Openpay documentation) to understand flow of various api's. Will be explaining how integration to be done with sdk using **Refund and Create New Order** API example.
2. To make refund from shop system will be sending refund related data to SDK system. As we can see in above flow diagram shop system will call business layer of SDK which is entry point for doing integration.
3. PlugintSDK.java is entry class for business layer and desired function call is made from shop system with required attributes to be sent.
4. For refund request below is function defination from PlugintSDK.java inside jar file

```
/**
 * Refund method creates request from @param refundData @param orderId and make
 * a call to api method in SDK layer. As this is a Generic method created to be
 * used on multiple platforms, Api method ([Method] section) and
 * class([ApiClass] section are loaded from mappingApiconfig.ini using
 * reflections
 *
 * Refund method basically used to do refunds
 *
 * @param refundData    It is a Map object coming from calling function which
 *                      contains relevant data required to make refund call like
 *                      refundData with key value pair map where keys can be
 *                      relevant to the attributes.Please refer API
 *                      documentation for fields related information {key,
 *                      value} : {String, Object}
 *
 * @param orderId       Object contains orderId/id's to get refund for
 *                      particular order. Refer API document to check data types
 *
 * @param attemptNumber Integer to be sent from calling function (of shop
 *                      system) with constant value as 1.This attribute is used
 *                      as counter to check number of attempts made for
 *                      retrying.
 *
```

Call the above function with required parameters, please refer to comments and Java Docs for the detailed explanation of function definition. Use the response for further implementations in shop system.

5. **As SDK is based on generic approach i.e. compatible with any Java based shop system, business layer plays around the logics to create request for each API call with data being sent from shop system.**

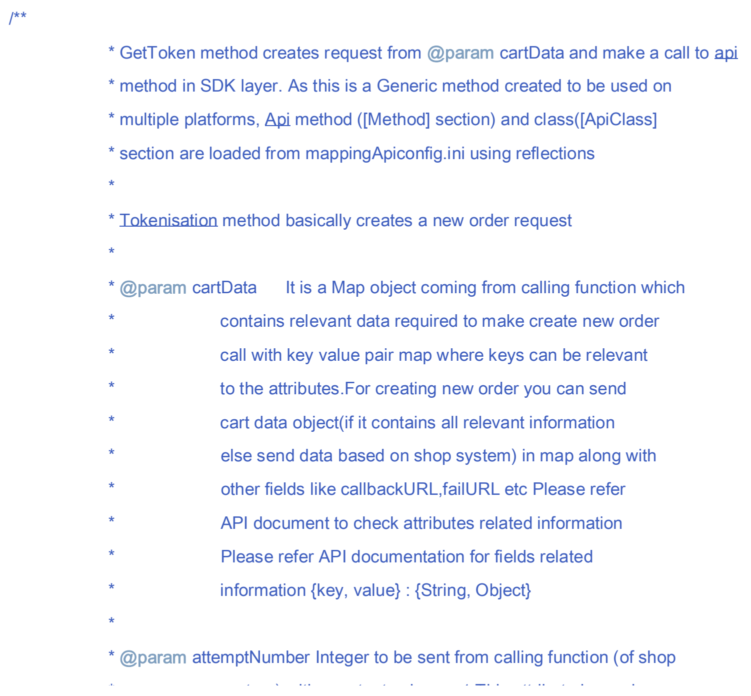
6. **Keys being used in parameter Map refundData has to be set as values in mappingShopConfig.ini located on root of Jar file under [Refund] section.**

For example:
refundData Map created is as follows
refundData = ["newPurchasePrice": $200.10
"reducePriceBy": 0
]

These keys set in refundData has to be set in Refund section of mappingShopConfig.ini as follows:

```
[Refund]
newPurchasePrice = newPurchasePrice
reducePriceBy = reducePriceBy
```

7. For create new order request below is function definition from PlugintSDK.java inside jar file



```
/**
     * GetToken method creates request from @param cartData and make a call to api
     * method in SDK layer. As this is a Generic method created to be used on
     * multiple platforms, Api method ([Method] section) and class([ApiClass]
     * section are loaded from mappingApiconfig.ini using reflections
     *
     * Tokenisation method basically creates a new order request
     *
     * @param cartData      It is a Map object coming from calling function which
     *                  contains relevant data required to make create new order
     *                  call with key value pair map where keys can be relevant
     *                  to the attributes.For creating new order you can send
     *                  cart data object(if it contains all relevant information
     *                  else send data based on shop system) in map along with
     *                  other fields like callbackURL,failURL etc Please refer
     *                  API document to check attributes related information
     *                  Please refer API documentation for fields related
     *                  information {key, value} : {String, Object}
     *
     * @param attemptNumber Integer to be sent from calling function (of shop
```

Call the above function with required parameters, please refer to comments and Java Docs for the detailed explanation of function definition. Send all fields required to make this API call in cartData Map from shop system. Use the response for further implementations in shop system example fetch transaction token from response map and redirect to Openpay using redirectURL which can be stored in property file of shop system.

8. Keys being used in parameter Map cartData has to be set as values in mappingShopConfig.ini located on root of Jar file under [createOrder] section.

For example cartData Map sent from shop system created is as follows

```
{totalItems=2, redirectUrl=https://uat-mrt.magentaretail.com.au/, code=00015001, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,522.87, value=500.98}, origin=ONLINE, failUrl=https://uat-mrt.magentaretail.com.au/, subTotal={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,513.88, value=500.88}, planCreationType=Pending, customerQuality=-1, emailAddress=ecommerce@xx.yy, entries=[{updateable=true, product={purchasable=true, code=1432722, name=Gigashot K80H, description=null, url=/Open-Catalogue/Cameras/Hand-held-Camcorders/Gigashot-K80H/p/1432722}, statusSummaryMap={}, quantity=1, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$396.88, value=396.88}, deliveryPointOfService=null, cancellableQty=0, cancelledItemsPrice=null, entries=null, deliveryMode=null, entryNumber=0, configurationInfos=[], basePrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$396.88, value=396.88}}, {updateable=true, product={purchasable=true, code=898503, name=1V, description=null, url=/Open-Catalogue/Cameras/Film-cameras/1V/p/898503}, statusSummaryMap={}, quantity=1, totalPrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,117.00, value=2117.11}, deliveryPointOfService=null, cancellableQty=0, cancelledItemsPrice=null, entries=null, deliveryMode=null, entryNumber=1, configurationInfos=[], basePrice={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,117.00, value=2117.11}}], deliveryMode={code=standard-gross, name=Standard Delivery, description=3-5 business days, deliveryCost={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$8.99, value=8.99}}, deliveryAddress={lastName=Gupta, country={isocode=BY, name=Belarus}, town=Bangalore, companyName=null, postalCode=560048, title=Dr., titlefirstNameCode=dr, firstName=Nupur, formattedAddress=14171 prestige shantiniketan, Whitefield, Bangalore, 560048, phone=09740755388, visibleInAddressBook=true, shippingAddress=true, id=8796650110999, billingAddress=true, region=null, line2=Whitefield, line1=14171 prestige shantiniketan, email=ecommerce@xx.yy, defaultAddress=false}, chargeBackCount=-1, totalPriceWithTax={currencyIso=USD, minQuantity=null, maxQuantity=null, priceType=BUY, formattedValue=$2,522.87, value=500.98}, net=false}
```

CartData map sent from shop system, above example is based on Hybris shop system where as it is cartData object is set to cartData map along with other mandatory fields like origin,redirectUrl etc. Optional fields can be left as blank as you can see below like dob if not available. Also, above request map is nested map, below config values are set in (.) pattern below like for resLine1 will set value as deliveryAddress.phone. If we have array in request map then for those entries use 'array' (prefix before field name) as text after main key, ex. for product data we have entries array so, values below will be set as itemName = entries.array.product.name

Also in mappingShopConfig.ini in section [PriceFormattingFields] and key priceFormattingFields add all attribute names which handle price value like purchasePrice in shop is coming as pu**rchasePrice = totalPrice.value.** So set totalPrice.value in the priceFormattingFields key. Other values are also added separated by (,)

```
[createOrder]
origin = origin
redirectUrl = redirectUrl
cancelUrl = failUrl
failUrl = failUrl
planCreationType = planCreationType
chargeBackCount = chargeBackCount
customerQuality = customerQuality
firstName = deliveryAddress.firstName
middleName =
surName = deliveryAddress.lastName
email = emailAddress
dob =
gender =
phoneNumber = deliveryAddress.phone
resline1 = deliveryAddress.line1
resline2 = deliveryAddress.line2
rescity = deliveryAddress.town
resstate = deliveryAddress.town
respincode = deliveryAddress.postalCode
delline1 = deliveryAddress.line1
delline2 = deliveryAddress.line2
delcity = deliveryAddress.town
delstate = deliveryAddress.town
delpincode = deliveryAddress.postalCode
deliveryDate =
description = deliveryMode.description
purchasePrice = totalPrice.value
retailerOrderNo = code
itemName = entries.array.product.name
itemGroup = entries.array.product.name
itemCode = entries.array.product.code
itemGroupCode = entries.array.product.name
itemRetailUnitPrice = entries.array.basePrice.value
itemQty = entries.array.quantity
itemRetailCharge = entries.array.totalPrice.value
```

NOTE : Don't change keys or section names of mappingShopConfig.ini while doing changes as these keys are mapped to keys of mappingApiConfig.ini

9. **The above steps are main steps used to integrate SDK JAR with shop system.**

10. **Add properties like Openpay redirect URL (provided by Openpay) , callbackURL, failURL (i.e. data needs to be handled from shop system) in shop system property file.**

11. **Price data for various API has to be sent with some denomination value (description provided in API doc), this price formatting is already handled by SDK for request related data.**

12. **For response related data please handle price formatting in shop system i.e. for API call { {/orders/limits}**

13. **For various API calls, which functions to call from PlugintSDK.java in businessLayer of JAR is as follows in below table. Please follow java docs for reference to function definition.**

API calls from Openpay	**Function Definition to be referred from SDK**
GET{/orders/{orderId}	**public Map<String, Object> updateShopOrder(final Object orderId, int attemptNumber) throws Exception();**
POST {/orders}	**public Map<String, Object> getToken(final Map<String, Object> cartData, int attemptNumber) throws Exception();**
POST {/orders/{orderId}/capture}	**public Map<String, Object> capturePayment(final Object orderId, int attemptNumber) throws Exception();**
POST {/orders/{orderId}/refund}	**public Map<String, Object> refund (final Map<String, Object> refundData, final Object orderId, int attemptNumber) throws Exception ();**
GET {/orders/limits}	**public Map<String, Object> getPSPConfig(int attemptNumber) throws Exception();**

14. **Initiating Token call from shop and redirecting to Openpay. Below example is just algorithm, please use logic based on your shop system like getting cart data.**

```
public void sendReuestToCreateNewOrder() {

//get cartData from shop

final CartData cartData = getCheckoutFacade().getCheckoutCart();

Map cartDataMap = omapper.convertValue(cartData, Map.class);

//put the other required attributes in property file and read the values from property file

cartDataMap.put("origin", getConfigurationService().getConfiguration().getProperty("origin"));

cartDataMap.put("redirectUrl", getConfigurationService().getConfiguration().getProperty("redirectUrl"));

cartDataMap.put("failUrl", getConfigurationService().getConfiguration().getProperty("failUrl"));

cartDataMap.put("planCreationType", getConfigurationService().getConfiguration().getProperty("planCreationType"));

cartDataMap.put("chargeBackCount",
Integer.parseInt(getConfigurationService().getConfiguration().getProperty(OpenpaypaymentaddonConstants.

CHARGE_BACK_COUNT).toString()));

cartDataMap.put("customerQuality", Integer.parseInt(getConfigurationService().getConfiguration()
                .getProperty(OpenpaypaymentaddonConstants.CUSTOMER_QUALITY).toString()));

final Map<String, Object> createTokenResponse = getPlugintSdk().getToken(cartDataMap,1);

//getting transaction token

//create a function to get transaction token from response by iterating the response

final String transactionToken = getTransactionToken(createTokenResponse,
getConfigurationService().getConfiguration().getProperty(OpenpaypaymentaddonConstants.REDIRECT_TOKEN_FIELD_NAME).toString())
;

//fetch redirect url provided by Openpay from property file

final String redirectUrlToOpenPay = getConfigurationService (). getConfiguration(). getProperty("OpenPayRedirectUrl").toString();

//redirect URL to browser using ajax call

return redirectUrlToOpenPay + transactionToken;

                }
```

2.2.2 Business Layer

This section will explain what all is availble in jar for this section. Except config files no file can be changed under JAR. As explained in diagram under section 2.2 business layer consist of multiple packages like

1. com.plugint.businessLayer.constant
2. com.plugint.businessLayer.core
3. com.plugint.businessLayer.features
4. com.plugint.businessLayer.util
5. com.plugint.customLayer
6. Config files which are present on /root of JAR and /resources of source code

For detailed understanding of all classes under above packages please refer to Java docs. Config file section will be explained in below section. PlugintSDK.java under com.plugint.businessLayer.core is the entry point for integration already explained in section 2.2.1

2.2.2.1 Configuration Section

This section is part of business layer and main logic resides here. There are 3 configuration files named as

1. **merchantConfig.ini** – This file is basically for merchant to add configuration properties like authentication details. Any changes related to authentication are done under [Authentication] section.
2. **mappingApiConfig.ini** – This file consist of all API related properties like host name. This file should not be changed until there is some changes in API document. As business layer uses Java Reflections method names, api class names , api request model names are present under this file.
3. **mappingShopConfig.ini** – This file consist of all shop related properties as discussed in point 6 of section 2.2.1. This file will be changed for every new Java based shop system. But only values need to be changed based on request sent from shop system, already explained above in point 6 of section 2.2.1.

If any new API method gets added then new section is introduced in both mappingShopConfig.ini and mappingApiConfig.ini with keys as common for both files.

2.2.2.2 Custom Layer

This layer is part of business layer created to customise attributes for request. Taking an example, need to send prices to Openpay in certain format,i.e. multiply each price value by 100 and send it. These kind of customisations are handled in this layer.

The changes need to be made is in mappingShopConfig.ini in section [PriceFormattingFields] and key priceFormattingFields. For this particular field add all attribute names which handle price value like purchasePrice in shop is coming as pu**rchasePrice = totalPrice.value.** So set totalPrice.value in the priceFormattingFields key. Other values are also added separated by (,).

2.2.3 Api Layer

This section is interacted by Business Layer. It basically contains all the logic related to client side interatcion like parsing json request, handling headers, creating connection and parsing response back to java objects. This layer basically contains multiple packages listed below

1. com.plugint.client
2. com.plugint .client.api
3. com.plugint.client.auth
4. com.plugint.client.model

For detailed understanding of all java classes present under above packages please refer java docs. Java documentation is present before each function inside class and separate folder is created under source code /target/apidocs/com/plugint.

OrderApi.java is main API class being called from business layer. As business layer uses Java Reflections method names, api class names everything is present under mappingApiConfig.xml

Below table represent the method call done to API layer (OrdersApi.java) from business layer(PlugintSDK.java).

Function Definition to be referred from SDK	**Function of API Layer**
public Map<String, Object> updateShopOrder(final Object orderId, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersOrderIdCancelPost(String orderId) throws ApiException();**
public Map<String, Object> getToken(final Map<String, Object> cartData, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersPost(IntegrationApiModelsCommandsCreateOrder body) throws ApiException();**
public Map<String, Object> capturePayment(final Object orderId, int attemptNumber) throws Exception();	**public IntegrationApiModelsOrder ordersOrderIdCapturePost(String orderId) throws ApiException();**
public Map<String, Object> refund (final Map<String, Object> refundData, final Object orderId, int attemptNumber) throws Exception ();	**public IntegrationApiModelsOrder ordersOrderIdRefundPost(String orderId, IntegrationApiModelsCommandsRefund body) throws ApiException();**
public Map<String, Object> getPSPConfig(int attemptNumber) throws Exception();	**public IntegrationApiModelsLimits ordersLimitsGet() throws ApiException ();**

Above API functions are handled in business layer using Java Reflections. If there are any changes to API documentation like change in method name or change in API models, please refer to mappingAPIconfig.xml. This file contains separate section for each type like for API models → [ApiModels] section has to be changed

3. Known Issues

None

4. Versions

Version	Date	Changes
1.0.0	15-May-2020	• Initial release