XPRESSPAGO

API Reference Guide

PREFACE

About This Guide

This guide provides brief information about XPRESSPAGO API service and how the JAVA SDK can be used to interact and manipulate XpressPago system.

For more detailed explanation and understanding of terms used in this document, please refer to document “XpressPago User Guide for Administrator”.

Who Should Read This Guide

The primary audience for this guide is technical user(s) responsible for integration with XpressPago. This guide mainly acts as a reference manual.

Contents

Introduction

XpressPago is an automated billing management system that enables you to securely store customer's sensitive payment information and enroll them in automatic debits from a financial instrument. XpressPago also enables payments for the subscribed services via an online application.

The purpose of this document is to describe the XpressPago functionalities exposed by the SDK, and to give a brief overview of its functionalities necessary to start consuming this service.

Introduction to XpressPago SDK

The XpressPago SDK is a convenient and safe way to connect to XpressPago API. By using the SDK, you simply create an object and fill it with the right properties. The SDK will create the right message internally and communicate to the XpressPago API after checking for validations and encrypting the message. This way you do not have to deal with all the details of communication.

This guide will show you some options on how to use the SDK but to view the detailed sample code that shows how to use the SDK you can visit the following URL:

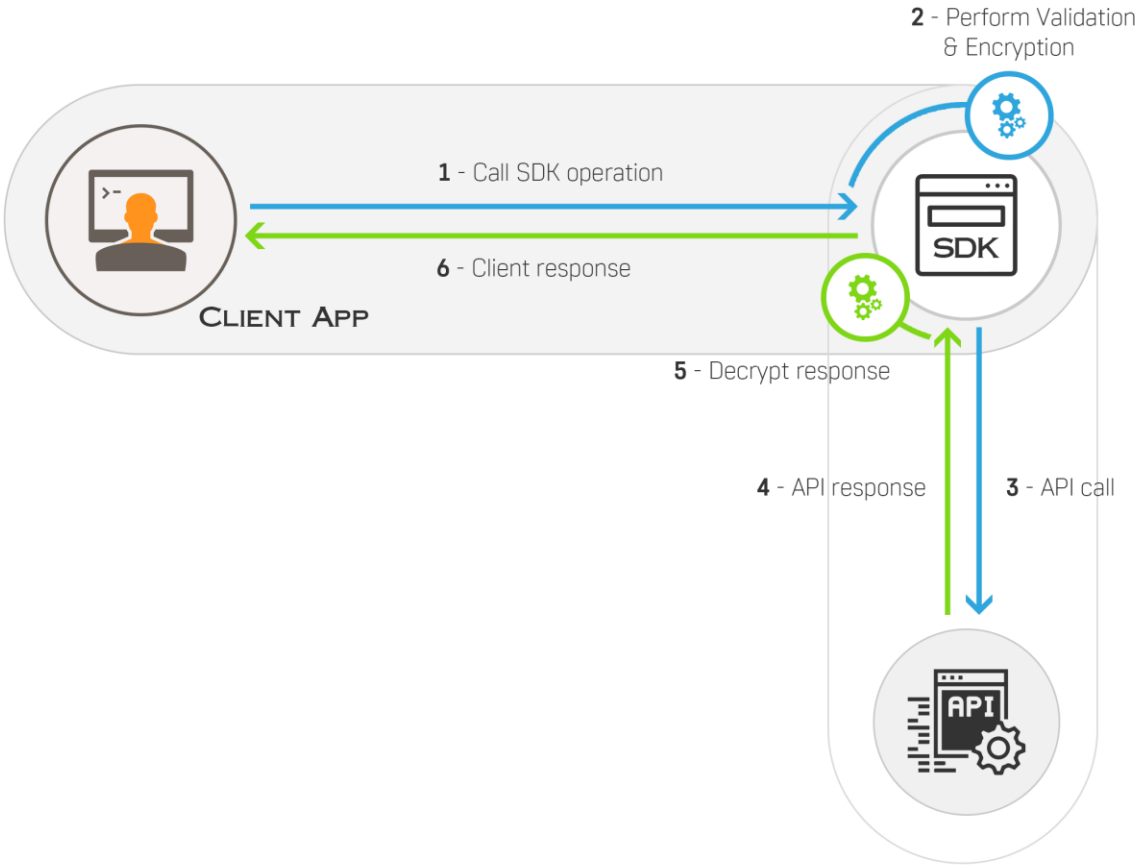
https://repo.croem.net/projects/NEOG/repos/metropago-gateway-java-sdk/browse

Glossary

SDK	Software Development Kit
API	Application Program Interface
Customer	Customer is an individual that use services provided by XpressPago processing. Customer is the core concept in all XpressPago processing.
Card	Card is one of the main entities in the XpressPago platform. Different financial companies including banks issue cards to the customers. Customer can add these cards in XpressPago and use them to perform payments against their added accounts.
Account	Account is an identity maintained in XpressPago, which is used to receive bill payments. Different type of accounts can exist depending upon the Service Type they are associated with.

Payment Instruction	Payment Instruction is an association between a card and an account. You can setup an instruction to automatically bill a card for a particular account.

How it Works



SDK Object

Once you have added the reference of the SDK files to your project and required third party libraries (mentioned in *Required Third Party Libraries* section) you can initialize the SDK by using the following statement:

Example:

```
MetropagoGateway sdk = new MetropagoGateway("SANDBOX", " 100177","AERT99HY");
```

If you need to perform transaction operations (Sale/Refund etc.), you should initialize the SDK by using the following statement (with Terminal Id):

```
MetropagoGateway sdk = new MetropagoGateway("SANDBOX", "100177" ,"AERT99HY" ,"100177001");
```

Initialize SDK

Initialize SDK (MetropagoGateway Object Reference) This object is used to setup the communication channel with the API			
Text field	**Description**	**Format**	**Required**
Environment	Identifies if you are connecting to the test or production API	Options • SANDBOX • PRODUCTION	✓
Merchant Id	A unique number provided by Sigma that identifies you as a merchant.	Numeric (6)	✓
Secret key	Secret key provided by Sigma authenticating the merchant. This should be shared with limited and only relevant individuals.	Alphanumeric (200)	✓
Terminal Id	A unique number provided by Sigma that identifies Terminal.	Alphanumeric (12)	✓

Customer Services

Customer is an individual that use services provided by XpressPago processing. Customer is the core concept in all XpressPago processing. Before you could store a card, or create a scheduled payment, you need to register a customer on the XpressPago system.

Customer Manager

Customer manager defines SDK methods that allow registering customer and managing customer accounts, cards and payments instructions.

Create Customer

Functionality: Registers customer in XpressPago platform. Customers are uniquely identified based on their unique identification that you must send. Once the customer is successfully saved in XpressPago, a customer token is sent for future references.

Example:

```
CustomerManager customerManager = new CustomerManager(sdk);
Customer customer = new Customer();
customer.setUniqueIdentifier("12347852");
customer. setFirstName ("John");
Customer customerSavedResult = customerManager.SaveCustomer(customer);
customerSavedResult.getResponseDetails().getIsSuccess(); // shows if the operation was successful
```

Search Customer

Functionality: Searches for a customer based on the provided filters. As described in “Create Customer” section, you can search for a customer using the Unique Identifier that you sent, or the Customer token that we generated on the response for “Create Customer”. Use customer search options to include required customer entities and information.

Example:

```
CustomerSearch customerfilter = new CustomerSearch();
customerfilter.setCustomerId("1");
customerfilter.setUniqueIdentifier("12347852");

CustomerSearchOption customerOptions = new CustomerSearchOption();

customerOptions.setIncludeShippingAddress(true);

customerOptions.setIncludeCardInstruments(true);

customerfilter.setSearchOption(customerOptions);
List<Customer> srcustomer = customerManager.SearchCustomer(customerfilter);
```

Update Customer

Functionality: Updates the customers profile information. Everything that is left NULL or empty, would be treated as unchanged and will not be updated.

Example:

```
Customer upcustomer = new Customer();
upcustomer.setUniqueIdentifier("12347852");
upcustomer.setCustomerId("1");
upcustomer.setStatus("Inactive");
Customer customerUpdatedResult  = customerManager.UpdateCustomer(upcustomer);
```

Customer Object Definition

Customer Object Reference
This object is sent to the API to create, update a customer. This is the same object type that is returned when customer is searched

Text field	Description	Format	Required
Customer Id	An Id generated by the Xpresspago system. Can be used in Update, and needs to be empty for Update customer	Alphanumeric (100)	
Unique Identification	Unique identification value of customer. Customer will be identified in the system based on this value.	Alphanumeric (50)	✓
Username	User may log in using this name	Alphanumeric (30)	
First Name	Customers First name	Alphanumeric (250)	
Last Name	Customers Last name	Alphanumeric (250)	
Company	Customers company affiliation	Alphanumeric (50)	
Website	Website address for the customer	Alphanumeric (65)	
Email	Primary Email Address of customer	Alphanumeric (300)	
Phone	Contact number	Numeric (10)	
Shipping Address	Address and contact information of the customer	Address Object (definition in miscellaneous section)	

Customer Search Option Object Definition

Customer Search Option Object Reference This object is sent to the API to include different customer entities and information when customer is searched			
Text field	**Description**	**Format**	**Required**
Include All	Customer Object include all information and entities in response	Boolean	
Include Card Instruments	Customer Object include card entities in response	Boolean	
Include Custom Fields	Customer Object include custom fields in response	Boolean	
Include Associated Entities	Customer Object include account entities in response	Boolean	
Include Billing Address	Customer Object include billing information in response	Boolean	
Include Payment Instructions	Customer Object include Payment Instructions information in response	Boolean	
Include Shipping Address	Customer Object include shipping information in response	Boolean	

Customer Additional Fields

Functionality: Additional fields that you may need to see later in XpressPago. In the additional fields, you can send any piece of information that you would need to see in the future. For example, you may add a value “2” against a field “NumberOfCars” against a customer or any other field information that you feel useful.

Example:

```
Customer customer = new Customer();
HashMap CustomFields = new HashMap();
CustomFields.put("PhoneOffice ", "95401011100");
CustomFields.put("NumberOfCars ", "2");
```

```
customer.setCustomFields(CustomFields);
customerManager.SaveCustomer(customer); // The same can be used for Updating a customer
```

Customer Object Additional fields

The additional fields that can be used to send information related to a customer's profile against Xpresspago

Text field	Description	Format	Required
Customer Type	Categorize customer as "Corporate" or "Consumer"	Options • Corporate • Consumer	
Identification Type	Identification type	Options • RUC • CEDULA • PASSPORT • OTHER	
Sex	"Male" or "Female"	Options • MALE • FEMALE	
Date of Birth	Customer date of birth	Date (mm/dd/yyyy)	
Phone Home	Home contact number	Numeric (10)	
Phone Mobile	Cell contact number	Numeric (10)	
Phone Office	Office contact number	Numeric (10)	
Preferred Communication	Mode of communication used to contact by customer	Options • SMS • EMAIL • NONE • BOTH	
Email2	Secondary email address	Alphanumeric (50)	

Customer Response Object Definition

Customer Create and Update Response	
Label	**Notes**
Customer Token	A unique token generated by the Xpresspago API that can be used to identify the customer later
Unique Identification	The Identification that was sent in the request
Response Code	Alpha numeric code with maximum length of three characters
Response Summary	Corresponding description of response code
Is Success	Showing if the operation was successful
Validation Errors	Collection of type Validation Errors describing the reasons why the operation failed.

Card Services

Adding and managing cards is only possible against a customer. That is why they are all part of customer's object:

Add Card

Functionality: Registers a card against customer in XpressPago platform. If a card is successfully saved, you receive a token that you can use later to refer to it.

Example:

```
// upCardCustomer is a customer object. It could be a new customer or an old customer fetched using the Search customer functionality
List<CreditCard> ccCreditCard = new ArrayList< CreditCard >();
CreditCard card = new CreditCard();
card.setCardholderName("John Snow");
card.setExpirationDate("0116");

// you may also use:
//card.setExpirationMonth("01");
//card.setExpirationYear("2016");

card.setNumber("4111111111111111");
ccCreditCard.add(card);
upCardCustomer.setCreditCards(ccCreditCard);

customerManager.UpdateCustomer(upCardCustomer);
```

Update Card without loading customer

Functionality: Updates a card based on customer and card token. In this case the user does not need to search for the customer first.

Example:

```
Customer customer = new Customer("ERA6546SD-ASDA4654SD");
CreditCard card = new CreditCard("AHUSHuhuIHu39824yu983HIUAhDEIUH93284");

card.setExpirationDate("0116");

// you may also use:
card.setExpirationMonth("01");
card.setExpirationYear("2016");
List<CreditCard> cardList = new ArrayList< CreditCard >();
cardList.add(card);
```

```
customer.setCreditCards(cardList);
customerManager.UpdateCustomer(customer);
```

Update Card using a loaded customer

Functionality: Updates a card against a customer. In this case the user needs to search for the customer before finding the right card, and updating it.

Example:

```
// upCardCustomer_1 is a customer object. It could be a new customer or an old customer fetched using the
Search customer functionality
List<CreditCard> ccCreditCard = upCardCustomer_1.getCreditCards();
for(int i = 0; i< ccCreditCard.size(); i++){
   CreditCard card = ccCreditCard.get(i);
   if("AHUSHuhuIHu39824yu983HIUAhDEIUH93284".equals(card.getToken())){
      card.setExpirationMonth("01");
      card.setExpirationYear("2016");
   }
}

upCardCustomer_1.setCreditCards(ccCreditCard); // the list contains the updated card
Customer customerUpdatedWithCardResult =  customerManager.UpdateCustomer(upCardCustomer_1);
```

Card Object Definition

Card Object Reference			
Text field	**Description**	**Format**	**Required**
Cardholder Name	Owner of the Instrument	Alphanumeric (50)	
Number	Card or Bank Account Number	Numeric (20)	✓
Expiration Year	Required in YYYY format	Numeric (4) (YYYY)	✓
Expiration Month	Required in MM format	Numeric (2) (MM)	✓
CVV	Card verification code	Numeric (4)	
Status	“Active” or “Inactive”. By default, is assumed active	Options • ACTIVE • INACTIVE	

Customer Id	A read-only numeric Id that shows what Id the card is connected to	Numeric (20)	
Customer Identifier	A read-only customer identifier that shows what unique Identifier the customer is connected to. This identification was provided by you when the customer was being created	Alphanumeric (50)	
Billing Information	Address and contact information of the customer	Contact Information Object (definition in miscellaneous section)	

Card Additional Fields

Functionality: Additional fields that you may need to see later in Xpresspago. In the additional fields, you can send any piece of information that you would need to see in the future. For example, you may add a value “2” against a field “TrustLevel” against a card or any other field as you may need.

Example:

```
HashMap CustomFields = new HashMap();
CustomFields.put("A field ", "Test");
CustomFields.put("TrustLevel ", "2");
card.setCustomFields(CustomFields);
```

Card Response Object Definition

Card Create and Update Response	
Label	**Notes**
Card Token	A unique token generated by the Xpresspago API that can be used to identify the card later
Response Code	Three-digit Alpha numeric code
Response Summary	Corresponding description of response code

Is Success	Showing if the operation was successful
Validation Errors	Collection of type Validation Errors describing the reasons why the operation failed.

Account Services

Account is an identity maintained in XpressPago, which is used to receive bill payments. Different type of accounts can exist depending upon the Service Type they are associated with.

You can add and manage accounts (services) against a customer:

Add Account

Functionality: Registers an account against customer in XpressPago platform. If the account is successfully saved, you receive a token that you can use later to refer to it.

Example:

```
// upCustomer_1 is a customer object. It could be a new customer or an old customer fetched using the Search customer functionality

List<CustomerEntity> customerAccounts = new ArrayList<CustomerEntity>();
CustomerEntity account = new CustomerEntity();
account.setAccountNumber("222222");
account.setCustomerId(upEntityCustomer.getCustomerId());
account.setFriendlyName("Primary Acc");
account.setServiceTypeName("Servicio Prueba A");
account.setStatus("ACTIVE");
customerAccounts.add(account);
upEntityCustomer. setCustomerEntities (customerAccounts);

Customer customerSavedWithAccountResult = customerManager.UpdateCustomer(upEntityCustomer);
```

Update Account without loading a customer

Functionality: Updates an account based on customer and account token. In this case the user does not need to search for the customer first.

Example:

```
Customer customer = new Customer("ERA6546SD-ASDA4654SD");
CustomerEntity account = new CustomerEntity("AHUSHuhuIHu39824yu983HIUAhDEIUH93284");
account.setStatus("INACTIVE");
List<CustomerEntity> accountList = new ArrayList< CustomerEntity>();
```

```
accountList.add(account);
customer. setCustomerEntities (accountList);
customerManager.UpdateCustomer(customer);
```

Update Account using a loaded customer

Functionality: Updates an account against a customer. In this case the user needs to search for the customer before finding the right card, and updating it

Example:

```
// upCardCustomer_1 is a customer object. It could be a new customer or an old customer fetched using the
List<CustomerEntity> customerAccounts = upCustomer_1. getCustomerEntities();
for(int i = 0; i< customerAccounts.size(); i++){
   CustomerEntity account = customerAccounts.get(i);
   if("1".equals(account.getId())){
      account.setStatus("INACTIVE");
   }
}
upCustomer_1. setCustomerEntities(customerAccounts);
Customer customerUpdatedWithAccountResult =  customerManager.UpdateCustomer(upCustomer_1);
```

Account Object Definition

Account Object Reference			
Text field	**Description**	**Format**	**Required**
Account Number	Product number issued to the customer	Alphanumeric (50)	✓
Service Type Name	Product opted by the customer. The service should be registered against your profile by Sigma	Alphanumeric (50)	✓
Friendly Name	A friendly name for the account		
Status	“Active”,“Inactive” or “Black Listed”. By default, is assumed active	Options: • ACTIVE • INACTIVE • BLACKLISTED	

Account Additional Fields

Functionality: Additional fields that you may need to see later in XpressPago. In the additional fields, you can send any piece of information that you would need to see in the future. For example, you may add a value “2” against a field “TrustLevel” against an account or any other field as you may need.

Example:

```
HashMap CustomFields = new HashMap();
CustomFields.put("A field ", "Test");
CustomFields.put("TrustLevel ", "2");
account.setCustomFields(CustomFields);
```

Account Response Object Definition

Account Create and Update Response	
Label	**Notes**
Account Token	A unique token generated by the Xpresspago API that can be used to identify the account later
Response Code	Three-digit Alpha numeric code
Response Summary	Corresponding description of response code
Is Success	Showing if the operation was successful
Validation Errors	Collection of type Validation Errors describing the reasons why the operation failed.

Payment Instruction Services

Payment Instruction is an association between a card and an account. You can setup an instruction to automatically bill a card for a particular account.

Add Payment Instruction using Account Token

Functionality: Registers a payment instruction against customer in XpressPago platform. If the instruction is successfully saved, you receive a token that you can use later to refer to it.

Example:

```
// upEntityCustomer is a customer object. It could be a new customer or an old customer fetched using the Search customer functionality

List<Instruction> lstInstruction = new ArrayList<Instruction>();
Instruction instruction = new Instruction();
instruction.setInstrumentToken("96932473-1E0B-485F-921D-02E7E15B031E");
instruction. setAccountToken ("96932473-1E0B-485F-921D-02E7E15B031E");
instruction. setScheduleDay ("25");
Calendar expiryDate = new GregorianCalendar(2027, 7, 26);

instruction.setExpirationDate(new SimpleDateFormat("MM/dd/yyyy KK:mm:ss a").format(expiryDate.getTime()));
instruction.setCustomerId(upEntityCustomer.getCustomerId());
instruction.setStatus("ACTIVE");
lstInstruction.add(instruction);
upEntityCustomer. setPaymentInstructions (lstInstruction);

Customer customerSavedWithInstructionResult = customerManager.UpdateCustomer(upEntityCustomer);
```

Add Payment Instruction using Account Number

Functionality: Registers a payment instruction against customer in XpressPago platform. If the instruction is successfully saved, you receive a token that you can use later to refer to it.

Example:

```
// upEntityCustomer is a customer object. It could be a new customer or an old customer fetched using the Search customer functionality

List<Instruction> lstInstruction = new ArrayList<Instruction>();
Instruction instruction = new Instruction();
instruction.setInstrumentToken("96932473-1E0B-485F-921D-02E7E15B031E");
instruction. setAccountNumber ("12345678");
instruction. setScheduleDay ("25");
```

```
Calendar expiryDate = new GregorianCalendar(2027, 7, 26);
instruction.setExpirationDate(new SimpleDateFormat("MM/dd/yyyy KK:mm:ss a").format(expiryDate.getTime()));
instruction.setCustomerId(upEntityCustomer.getCustomerId());
instruction.setStatus("ACTIVE");
lstInstruction.add(instruction);
upEntityCustomer. setPaymentInstructions (lstInstruction);

Customer customerSavedWithInstructionResult = customerManager.UpdateCustomer(upEntityCustomer);
```

Update Payment Instruction without loading a customer

Functionality: Updates a payment instruction based on customer and payment instruction token. In this case the user does not need to search for the customer first.

Example:

```
Customer customer = new Customer("ERA6546SD-ASDA4654SD");
Instruction paymentInstruction = new Instruction ("AHUSHuhuIHu39824yu983HIUAhDEIUH93284");
paymentInstruction.setStatus("INACTIVE");
List< Instruction > instructionList = new ArrayList< Instruction >();
instructionList.add(paymentInstruction);
customer.setPaymentInstructions(instructionList);
customerManager.UpdateCustomer(customer);
```

Update Payment Instruction using a loaded customer

Functionality: Updates a payment instruction against a customer. In this case the user needs to search for the customer before finding the right instruction, and updating it.

Example:

```
// upCustomer_5 is a customer object. It could be a new customer or an old customer fetched using the
 for(int i = 0; i< lstInstruction.size(); i++){
        Instruction entity = lstInstruction.get(i);
        if("1".equals(entity.getId())){
             entity.setStatus("INACTIVE");
        }
    }
upCustomer_5.setPaymentInstruction(lstInstruction);
Customer customerUpdatedWithInstructionResult =  customerManager.UpdateCustomer(upCustomer_5);
```

Payment Instruction Object Definition

Payment Instruction Object Reference			
Text field	**Description**	**Format**	**Required**
Instrument Token	Unique identifier of the card generated by Xpresspago API	Alphanumeric (50)	✓
Account Token	Unique identifier of the account generated by Xpresspago API	Alphanumeric (50)	✓
Scheduled Day	A friendly name for the account		✓
Instruction Expiration Date	The date after which instruction would not apply	Date (MM-DD-YYYY) format	✓
Status	“Active” or “Inactive”	Options • ACTIVE • INACTIVE	✓

Payment Instruction Additional Fields

Functionality: Additional fields that you may need to see later in XpressPago. In the additional fields, you can send any piece of information that you would need to see in the future. For example, you may add a value “2” against a field “TrustLevel” against an account or any other field as you may need.

Example:

```
HashMap CustomFields = new HashMap();
CustomFields.put("A field ", "Test");
CustomFields.put("TrustLevel ", "2");
instruction.setCustomFields(CustomFields);
```

Payment Instruction Create and Update Response	
Label	**Notes**
Payment Instruction Token	A unique token generated by the XpressPago API that can be used to identify the payment instruction later
Response Code	Three-digit Alpha numeric code
Response Summary	Corresponding description of response code
Is Success	Showing if the operation was successful
Validation Errors	Collection of type Validation Errors describing the reasons why the operation failed.

Transaction Manager Services

Transaction Manager defines SDK methods that allow you to perform transaction operations.

Update Transaction Status

Functionality: Updates a transactions status.

Example:

```
TransactionManager transactionManager = new TransactionManager(sdk);
Transaction updateTransaction = transactionManager.UpdateTransaction(new Transaction("Account Payment # 12345687", "POSTED", “1005158784”));
```

Perform Sale / Preauthorization

Functionality: Perform Sale / Preauthorization Transaction using Customer’s information**.**

Example:

```
Transaction transRequest = new Transaction();
transRequest.setAmount(1);
Customer cust = new Customer();
CreditCard card = new CreditCard();
cust.setCustomerId("12968");
card.setToken("A2478342-617A-4EE5-BC1B-ECD3D132D9CF");
List<CreditCard> cardList = new ArrayList<>();
cardList.add(card);
cust.setCreditCards(cardList);
transRequest.setCustomerData(cust);
TransactionOptions transactionOptions = new TransactionOptions();
```

For Sale

```
transactionOptions.setOperation("Sale");
transRequest.setTransactOptions(transactionOptions);
Transaction sale_response = transactionManager.Sale(transRequest);
```

For Preauthorization

```
transactionOptions.setOperation("PreAuthorization");
transRequest.setTransactOptions(transactionOptions);
Transaction preauth_response = transactionManager.PreAuthorization(transRequest);
```

Perform Refund / Adjustment

Functionality: Refund Transaction using Reference Transaction Id.

Example:

```
Transaction transRequest = new Transaction();
transRequest.setAmount(1);
transRequest.setTransactionId(“100478996”);
```

For Refund

```
TransactionOptions  transactionOptions = new TransactionOptions();
transactionOptions.setOperation("Refund");
transRequest.setTransactOptions(transactionOptions);
Transaction refund_response = transactionManager.Refund(transRequest);
```

For Adjustment

```
TransactionOptions  transactionOptions = new TransactionOptions();
transactionOptions.setOperation("Adjustment");
transRequest.setTransactOptions(transactionOptions);
Transaction adjustment_response = transactionManager.Adjustment(transRequest);
```

Search Transaction

Functionality: Search transactions using different filters (Credit Card Number, Cardholder, Date, Reference Transaction Id etc.)

Example:

```
TransactionSearchRequest transactionSearchRequest = new TransactionSearchRequest();
transactionSearchRequest.setTransactionId("102753132");
   transactionSearchRequest.setCustomerId(customerId);
List<Transaction> transactions = transactionManager.SearchTransaction(transactionSearchRequest);
```

Sale / Preauthorization Transaction Object Definition

Sale / Preauthorization Transaction Object Reference			
Text field	**Description**	**Format**	**Required**
Amount	Amount of Transaction	Decimal	✓
Customer Data	Customer Token, Card Token and Account Token is set in Customer's Model		✓
Order Tracking Number	Client Tracking Number for Transaction Reference	Alphanumeric (50)	

Refund / Adjustment Transaction Object Definition

Refund / Adjustment Transaction Object Reference			
Text field	**Description**	**Format**	**Required**
Amount	Amount of Transaction	Decimal	✓
Transaction Id	Reference Transaction Id	Numeric(50)	✓

Transaction Status Object Definition

Update Transaction Status Object Reference			
Text field	**Description**	**Format**	**Required**
Transaction Id	Id generated by XpressPago that needs to be updated	Numeric (12)	✓
Status	Change the status of the transaction to	Options • POSTED • PENDING • ERROR • OTHER	✓
Change Description	Description/reason of the change	Alphanumeric (500)	✓

Transaction Response Object Definition

Transaction Update Response	
Label	**Notes**
Transaction Id	The same Id that was sent in when update was requested
Response Code	Three-digit Alpha numeric code
Response Summary	Corresponding description of response code
Is Success	Showing if the operation was successful
Validation Errors	Collection of type Validation Errors describing the reasons why the operation failed.

Miscellaneous Objects and Services

Address Definition

Address Object Reference This object used wherever an address information is required			
Text field	**Description**	**Format**	**Required**
AddressLine1	Address information of the customer	Alphanumeric (100)	
AddressLine2	Address Line 2 information of the customer	Alphanumeric (100)	
Country	Address Country	Alphanumeric (50)	
City	Address City	Alphanumeric (30)	

State	Address State	Alphanumeric (50)	
Sub Division	Address Sub Division	Alphanumeric (50)	
Zip Code	Address Zip Code	Alphanumeric (50)	

Validation Error Definition

Validation Object Reference
This object is a read only object and it is sent as a part of response to describe the details of the errors encountered while operating the request

Text field	Description	Format
Count	Number of errors encountered	Numeric (2)
Error Code	A short code representing the error	Alphanumeric (5)
Error Summary	A brief description of the error	Alphanumeric (150)
Error Description	Address Country	Alphanumeric (1000)
Error Details	A collection describing all the failures that lead to this error	A collection of Alphanumeric (1000)

Response Codes

0	Success
1	Updated
2	Deleted
3	Error
4	Exception
5	Saved
6	Invalid Request
11	Duplicate Customer Identification
12	InvalidCustomerInformation
13	DuplicateCreditCard
14	DuplicateBeneficaryAccount
15	AccountNotFound
16	BeneficiaryServiceNotFound
50	TransactionInformationNotFound
51	CustomerNotFound
52	CreditCardNotFound
53	DefaultPaymentMethodNotFound
54	TokenNotFound
55	InactiveCreditCard
56	InvalidTransactionOperation
100	BillingAddressRequired
101	ShippingAddressRequired
102	ZipCodeRequired
103	CreditCardRequired
104	CardHolderRequired
105	ExpirationDateRequired
106	CompanyNameRequired
107	ACHAccountNumberRequired
108	ACHAccountHolderRequired
109	WalletAccountNumberRequired
110	WalletAccountHolderRequired
111	TransactionFailedAtThirdparty
17	InvalidInstructionInformation

Required Third Party Libraries

Gson

You can download jar file from the below link and add reference jar into your project to use SDK.

Download link

http://search.maven.org/#artifactdetails%7Ccom.google.code.gson%7Cgson%7C2.5%7Cjar

SDK Logs

You can enable disabled logs of SDK, which is maintained in an xml file named **MetroPagoAPILogs.xml** located in user home directory.

Example:

```
sdk.setLogsEnabled(true);
```

SDK Culture

You can change SDK culture to English or Spanish. By default, SDK culture is English.

Example:

```
sdk.setCulture("es");
```

Culture	Culture Value
English	en
Spanish	es