BE PAYMENT READY

Moneris Gateway API - Integration Guide – Java

Version: 1.2.10

Security and Compliance

Your solution may be required to demonstrate compliance with the card associations' PCI/CISP/PABP requirements. For more information on how to make your application PCI-DSS compliant, contact the Moneris Sales Center and visit https://developer.moneris.com to download the PCI_DSS Implementation Guide.

All Merchants and Service Providers that store, process, or transmit cardholder data must comply with PCI DSS and the Card Association Compliance Programs. However, certification requirements vary by business and are contingent upon your "Merchant Level" or "Service Provider Level".

The card association has some data security standards that define specific requirements for all organizations that store, process, or transmit cardholder data. As a Moneris client or partner using this method of integration, your solution must demonstrate compliance to the Payment Card Industry Data Security Standard (PCI DSS) and/or the Payment Application Data Security Standard (PA DSS). These standards are designed to help the cardholders and merchants in such ways as they ensure credit card numbers are encrypted when transmitted/stored in a database and that merchants have strong access control measures.

Non-compliant solutions may prevent merchant boarding with Moneris. A non-compliant merchant can also be subject to fines, fees, assessments or termination of processing services.

For further information on PCI DSS & PA DSS requirements, visit http://www.pcisecuritystandards.org.

Confidentiality

You have a responsibility to protect cardholder and merchant related confidential account information. Under no circumstances should ANY confidential information be sent via email while attempting to diagnose integration or production issues. When sending sample files or code for analysis by Moneris staff, all references to valid card numbers, merchant accounts and transaction tokens should be removed and or obscured. Under no circumstances should live cardholder accounts be used in the test environment.

Changes in v1.2.10

- Added missing request fields in MCP Purchase with Vault

Previous version changes

Changes in v1.2.9

- Corrected sample code for some financial transactions (to remove old method of MCP processing)

Changes in v1.2.8

- Added section about Multi-Currency Pricing (MCP) transactions
- Added new conceptual topics in Credential on File:
  - Merchant- vs. Cardholder-Initiated COF Transactions
  - COF With Previously Stored Credentials
- Discover test card number has been changed to 6011000992927602
- Amended electronic commerce indicator description in Defintion of Response Fields to remove deprecated allowable values (8 and 9)

Changes in v1.2.7

Missing limits for request variables amount, completion amount and transaction amount were replaced

Changes in v1.2.6

Electronic commerce Indicator (crypt_type) request variable's value of '7' amended to reflect that it also represents an American ExpressSafeKey non-authenticated transaction

Changes in v1.2.5

Purchase transaction amended to include Customer ID variable

Changes in v1.2.4

Changes limits in Amount, Transaction Amount, Completion Amount request variables to reflect 10 decimals.

Changes in v1.2.3

This version adds information about passing Offlinx™ data for the Card Match pixel tag via Unified API transactions.

Table of Contents

1 About This Documentation

1.1 Purpose

This document describes the transaction information for using the Java API for sending credit card transactions. In particular, it describes the format for sending transactions and the corresponding responses you will receive.

This document contains information about the following features:

- Basic transactions
- MPI – Verified by Visa, MasterCard Secure Code and American Express SafeKey
- INTERAC® Online Payment
- Vault
- MSR (Magnetic Swipe Reader) and Encrypted MSR
- Apple Pay and Android Pay In-App
- Transaction Risk Management Tool
- Convenience fee
- Visa Checkout
- MasterCard MasterPass
- Level 2/3 Transactions

Getting Help

Moneris has help for you at every stage of the integration process.

Getting Started	During Development	Production
Contact our Client Integration Specialists: clientintegrations@moneris.com Hours: Monday – Friday, 8:30am to 8 pm ET	If you are already working with an integration specialist and need technical development assistance, contact our eProducts Technical Consultants: 1-866-319-7450 eproducts@moneris.com Hours: 8am to 8pm ET	If your application is already live and you need production support, contact Moneris Customer Service: onlinepayments@moneris.com 1-866-319-7450 Available 24/7

For additional support resources, you can also make use of our community forums at

http://community.moneris.com/product-forums/

1.2 Who Is This Guide For?

The Moneris Gateway API - Integration Guide is intended for developers integrating with the Moneris Gateway.

This guide assumes that the system you are trying to integrate meets the requirements outlined below and that you have some familiarity with the Java programming language.

System Requirements

- Java or above
- Port 443 open for bi-directional communication
- Web server with a SSL certificate

2 Basic Transaction Set

2.1 Purchase

Verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

Purchase transaction object definition

```
Purchase purchase = new Purchase();
```

HttpsPostRequest object for Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(purchase);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Purchase transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric	`purchase.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) +	`purchase.setAmount(amount);`

Variable Name	Type and Limits	Set Method
	decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	
credit card number	*String* 20-character alphanumeric	`purchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric (YYMM format)	`purchase.setExpDate(expiry_ date);`
electronic commerce indic- ator	*String* 1-character alphanumeric	`purchase.setCryptType (crypt);`

Purchase transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 50-character alphanumeric	`purchase.setCustId(cust_id);`
card match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String* 50-character alphanumeric	`purchase.setCmId (transaction_id);`
Customer Information	*Object* N/A	`purchase.setCustInfo (customer);`
AVS Information	*Object* N/A	`purchase.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing cre-	*Object* N/A	`purchase.setCvdInfo (cvdCheck);`

Variable Name	Type and Limits	Set Method
dentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.		
Convenience Fee Information **NOTE:** This variable does not apply to Credential on File transactions.	*Object* N/A	`purchase.setConvenienceFee (convFeeInfo);`
Recurring Billing	*Object* N/A	`purchase.setRecurInfo (recurInfo);`
dynamic descriptor	*Object* 20-character alphanumeric	`purchase .setDynamicDescriptor (dynamic_descriptor);`
wallet indicator **NOTE:** For basic Purchase and Preauthorization, the wallet indicator applies to Visa Checkout and MasterCard MasterPass only. For more, see Appendix A Definitions of Request Fields	*String* 3-character alphanumeric	`purchase.setWalletIndicator (wallet_indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`purchase.setCofInfo(cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Purchase

```
public class TestCanadaPurchase
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String amount = "5.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM format
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
Purchase purchase = new Purchase();
purchase.setOrderId(order_id);
```

Sample Purchase

```
    purchase.setAmount(amount);
    purchase.setPan(pan);
    purchase.setExpdate(expdate);
    purchase.setCryptType(crypt);
    purchase.setDynamicDescriptor("123456");
    //purchase.setWalletIndicator(""); //Refer documentation for possible values
    purchase.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
    alphanumeric characters transaction id generated by merchant
    //optional - Credential on File details
    CofInfo cof = new CofInfo();
    cof.setPaymentIndicator("U");
    cof.setPaymentInformation("2");
    cof.setIssuerId("139X3130ASCXAS9");

    purchase.setCofInfo(cof);

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(purchase);
    mpgReq.setStatusCheck(status_check);

    //Optional - Proxy
    mpgReq.setProxy(false); //true to use proxy
    mpgReq.setProxyHost("proxyURL");
    mpgReq.setProxyPort("proxyPort");
    mpgReq.setProxyUser("proxyUser"); //optional - domainName\User
    mpgReq.setProxyPassword("proxyPassword"); //optional
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    System.out.println("HostId = " + receipt.getHostId());
    System.out.println("MCPAmount = " + receipt.getMCPAmount());
    System.out.println("MCPCurrencyCode = " + receipt.getMCPCurrencyCode());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.2 Pre-Authorization

Verifies and locks funds on the customer’s credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant’s account, a Pre-Authorization Completion transaction must be performed. A Pre-Authorization transaction may only be "completed" once.

Things to Consider:

- If a Pre-Authorization transaction is not followed by a Completion transaction, it must be reversed via a Completion transaction for 0.00. See "Pre-Authorization Completion" on page 23
- A Pre-Authorization transaction may only be "completed" once . If the Completion transaction is for less than the original amount, a Re-Authorization transaction is required to collect the remaining funds by another Completion transaction. See Re-Authorization (page 26).
- For a process flow, see "Process Flow for Basic Pre-Auth, Re-Auth and Completion Transactions" on page 511

Pre-Authorization transaction object definition

```
PreAuth preauth = new PreAuth();
```

HttpsPostRequest object for Pre-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(preauth);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_ token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Pre-Authorization transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric	`preauth.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`preauth.setAmount(amount);`
credit card number	*String* max 20-character alpha-numeric	`preauth.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`preauth.setExpDate(expiry_date);`
electronic commerce indicator	*String* 1-character alphanumeric	`preauth.setCryptType(crypt);`

Pre-Authorization transaction request fields – Optional

Variable Name	Type and Limits	Set Method
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`preauth.setDynamicDescriptor (dynamic_descriptor);`
card match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String* 50-character alphanumeric	`preauth.setCmId(transaction_ id);`
Customer Information	*Object* N/A	`preauth.setCustInfo (customer);`
AVS Information	*Object* N/A	`preauth.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`preauth.setCvdInfo (cvdCheck);`
customer ID	*String* 30-character alphanumeric	`preauth.setCustId(cust_id);`
wallet indicator **NOTE:** For basic Purchase and Preauthorization, the wallet indicator applies to Visa	*String* 3-character alphanumeric	`preauth.setWalletIndicator (wallet_indicator);`

Variable Name	Type and Limits	Set Method
Checkout and MasterCard MasterPass only. For more, see Appendix A Definitions of Request Fields		
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`cof.setCofInfo(cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");`

Variable Name	Type and Limits	Set Method
		NOTE: For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Pre-Authorization

```
package Canada;
import JavaAPI.*;
public class TestCanadaPreauth
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "5.00";
String pan = "4242424242424242";
String expdate = "1902";
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
PreAuth preauth = new PreAuth();
preauth.setOrderId(order_id);
preauth.setAmount(amount);
preauth.setPan(pan);
preauth.setExpdate(expdate);
preauth.setCryptType(crypt);
//preauth.setWalletIndicator(""); //Refer documentation for possible values
//preauth.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
alphanumeric characters transaction id generated by merchant
//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

preauth.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(preauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
```

Sample Pre-Authorization

```
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    //System.out.println("StatusCode = " + receipt.getStatusCode());
    //System.out.println("StatusMessage = " + receipt.getStatusMessage());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.3 Pre-Authorization Completion

Retrieves funds that have been locked (by a Pre-Authorization or Re-Authorization transaction), and prepares them for settlement into the merchant's account.

Things to Consider:

- A Pre-Authorization or Re-Authorization transaction can only be completed once. Refer to the Re-Authorization transaction for more information on how to perform multiple Completion transactions.
- To reverse the full amount of a Pre-Authorization transaction, use the Completion transaction with the amount set to `0.00`.
- To process this transaction, you need the order ID and transaction number from the original Pre-Authorization transaction.
- For a process flow, see Appendix D Process Flow for Basic Pre-Auth, Re-Auth and Completion Transactions

Pre-Authorization Completion transaction object

```
Completion completion = new Completion();
```

HttpsPostRequest object for Pre-Authorization Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(completion);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Pre-Authorization Completion transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`completion.setOrderId(order_id);`
completion amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`completion.setCompAmount(comp_amount);`
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores	`completion.setTxnNumber(txn_number);`

Variable Name	Type and Limits	Set Method
	variable length	
electronic commerce indicator	*String* 1-character alphanumeric	`completion.setCryptType (crypt);`

Pre-Authorization Completion transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`completion.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`completion .setDynamicDescriptor (dynamic_descriptor);`
shipping indicator	*String* 1-character alphanumeric	`completion.setShipIndicator (ship_indicator);`

Sample Pre-Authorization Completion

```
package Canada;
import JavaAPI.*;
public class TestCanadaCompletion
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test1436981327037";
String amount = "1.00";
String txn_number = "152900-0_10";
String crypt = "7";
String cust_id = "my customer id";
String dynamic_descriptor = "my descriptor";
String processing_country_code = "CA";
boolean status_check = false;
Completion completion = new Completion();
completion.setOrderId(order_id);
completion.setCompAmount(amount);
completion.setTxnNumber(txn_number);
completion.setCryptType(crypt);
```

Sample Pre-Authorization Completion

```
    completion.setCustId(cust_id);
    completion.setDynamicDescriptor(dynamic_descriptor);
    completion.setShipIndicator("P");

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(completion);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.4 Re-Authorization

If a Pre-Authorization transaction has already taken place, and not all the locked funds were released by a Completion transaction, a Re-Authorization allows you to lock the remaining funds so that they can be released by another Completion transaction in the future.

Re-Authorization is necessary because funds that have been locked by a Pre-Authorization transaction can only be released by a Completion transaction one time. If the Completion amount is less than the Pre-Authorization amount, the remaining money cannot be "completed".

For a process flow, Appendix D Process Flow for Basic Pre-Auth, Re-Auth and Completion Transactions.

Re-Authorization transaction object definition

```
ReAuth reauth = new ReAuth();
```

HttpsPostRequest object for Re-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(reauth);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Re-Authorization transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`reauth.setOrderId(order_id);;`
original order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`reauth.setOrigOrderId(orig_order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits	`reauth.setAmount(amount);`

Variable Name	Type and Limits	Set Method
	(cents) after the decimal point **EXAMPLE:** 1234567.89	
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores variable length	`reauth.setTxnNumber(txn_number);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`reauth.setCryptType(crypt);`

Re-Authorization transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`reauth.setCustId(cust_id);`
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`reauth.setDynamicDescriptor(dynamic_descriptor);`
Customer Information	*Object* N/A	`reauth.setCustInfo(customer);`
AVS Information	*Object* N/A	`reauth.setAvsInfo(avsCheck);`
CVD Information	*Object*	`reauth.setCvdInfo(cvdCheck);`

Variable Name	Type and Limits	Set Method
NOTE: When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	N/A	

Sample Re-Authorization

```
package Canada;
import JavaAPI.*;
public class TestCanadaReauth
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String order_id = "mvt271355ss7ssss839ssdfsdfsdf";
String orig_order_id = "mvt3213820409";
String amount = "4.00";
String txn_number = "200069-0_10";
String crypt = "8";
String dynamic_descriptor = "123456";
String cust_id = "my customer id";
String processing_country_code = "CA";
boolean status_check = false;
ReAuth reauth = new ReAuth();
reauth.setOrderId(order_id);
reauth.setCustId(cust_id);
reauth.setOrigOrderId(orig_order_id);
reauth.setTxnNumber(txn_number);
reauth.setAmount(amount);
reauth.setCryptType(crypt);
reauth.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(reauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
```

Sample Re-Authorization

```
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.5 Force Post

Retrieves the locked funds and prepares them for settlement into the merchant's account.

Used when a merchant obtains the authorization number directly from the issuer by a third-party authorization method (such as by phone).

Things to Consider:

- This transaction is an independent completion where the original Pre-Authorization transaction was not processed via the same Moneris Gateway merchant account.
- It is not required for the transaction that you are submitting to have been processed via the Moneris Gateway. However, a credit card number, expiry date and original authorization number are required.
- Force Post transactions are not supported for UnionPay

Force Post transaction object definition

```
ForcePost forcepost = new ForcePost();
```

HttpsPostRequest object for Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(forcepost);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String*	`mpgReq.setStoreId(store_id);`

Variable Name	Type and Limits	Set Method
	N/A	
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Force Post transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`forcepost.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`forcepost.setAmount(amount);`
credit card number	*String* max 20-character alpha-numeric	`forcepost.setPan(pan);`
expiry date	*String* 4-character alphanumeric	`forcepost.setExpDate(expiry_date);`

Variable Name	Type and Limits	Set Method
	YYMM	
authorization code	*String* 8-character alphanumeric	`forcepost.setAuthCode(auth_code);`
electronic commerce indicator	*String* 1-character alphanumeric	`forcepost.setCryptType(crypt);`

Force Post transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`forcepost.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`forcepost.setDynamicDescriptor(dynamic_descriptor);`

Sample Force Post

```
package Canada;
import JavaAPI.*;
public class TestCanadaForcePost
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "my customer id";
String store_id = "moneris";
String api_token = "hurgle";
String amount = "1.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM format
String auth_code = "88864";
String crypt = "7";
String dynamic_descriptor = "my descriptor";
String processing_country_code = "CA";
boolean status_check = false;
ForcePost forcepost = new ForcePost();
forcepost.setOrderId(order_id);
```

Sample Force Post

```
    forcepost.setCustId(cust_id);
    forcepost.setAmount(amount);
    forcepost.setPan(pan);
    forcepost.setExpdate(expdate);
    forcepost.setAuthCode(auth_code);
    forcepost.setCryptType(crypt);
    forcepost.setDynamicDescriptor(dynamic_descriptor);

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(forcepost);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CorporateCard = " + receipt.getCorporateCard());
    //System.out.println("MessageId = " + receipt.getMessageId());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.6 Purchase Correction

Restores the full amount of a previous Purchase, Pre-Authorization Completion or Force Post transaction to the cardholder's card, and removes any record of it from the cardholder's statement.

This transaction can be used against a Purchase or Pre-Authorization Completion transaction that occurred same day provided that the batch containing the original transaction remains open. When using the automated closing feature, Batch Close occurs daily between 10 and 11 pm Eastern Time.

Things to Consider:

- To process this transaction, you need the order ID and the transaction number from the original Completion, Purchase or Force Post transaction.

Purchase Correction transaction object definition

```
PurchaseCorrection purchasecorrection = new PurchaseCorrection();
```

HttpsPostRequest object for Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(purchasecorrection);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_ token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_ check);`

Purchase Correction transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`purchasecorrection .setOrderId(order_id);`
transaction number	*String*	`purchasecorrection`

Variable Name	Type and Limits	Set Method
	255-character, alpha-numeric, hyphens or under-scores variable length	`.setTxnNumber(txn_number);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`purchasecorrection .setCryptType(crypt);`

Purchase Correction transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`purchasecorrection.setCustId (cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`purchasecorrection .setDynamicDescriptor (dynamic_descriptor);`

Sample Purchase Correction

```
package Canada;
import JavaAPI.*;
public class TestCanadaPurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test1432065003686";
String txn_number = "42014-0_10";
String crypt = "7";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
boolean status_check = false;
PurchaseCorrection purchasecorrection = new PurchaseCorrection();
purchasecorrection.setOrderId(order_id);
purchasecorrection.setTxnNumber(txn_number);
purchasecorrection.setCryptType(crypt);
purchasecorrection.setDynamicDescriptor(dynamic_descriptor);
purchasecorrection.setCustId("my customer id");
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

Sample Purchase Correction

```
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(purchasecorrection);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.7 Refund

Restores all or part of the funds from a Purchase, Pre-Authorization Completion or Force Post transaction to the cardholder's card.

Unlike a Purchase Correction, there is a record of both the initial charge and the refund on the cardholder's statement.

For processing refunds on a different card than the one used in the original transaction, the Independent Refund transaction should be used instead.

To process this transaction, you need the order ID and transaction number from the original Completion, Purchase or Force Post transaction.

Refund transaction object definition

```
Refund refund = new Refund();
```

HttpsPostRequest object for Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(refund);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Refund transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`refund.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`refund.setAmount(amount);`
transaction number	*String* 255-character, alpha-	`refund.setTxnNumber(txn_number);`

Variable Name	Type and Limits	Set Method
	numeric, hyphens or under-scores variable length	
electronic commerce indicator	*String* 1-character alphanumeric	`refund.setCryptType(crypt);`

Sample Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaRefund
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String amount = "1.00";
String crypt = "7";
String dynamic_descriptor = "123456";
String custid = "mycust9";
String order_id = "mvt2713618548";
String txn_number = "911464-0_10";
String processing_country_code = "CA";
boolean status_check = false;
Refund refund = new Refund();
refund.setTxnNumber(txn_number);
refund.setOrderId(order_id);
refund.setAmount(amount);
refund.setCryptType(crypt);
refund.setCustId(custid);
refund.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(refund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
```

Sample Refund

```
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

2.8 Independent Refund

Credits a specified amount to the cardholder's credit card. The credit card number and expiry date are mandatory.

It is not necessary for the transaction that you are refunding to have been processed via the Moneris Gateway.

Things to Consider:

- Because of the potential for fraud, permission for this transaction is not granted to all accounts by default. If it is required for your business, it must be requested via your account manager.

Independent Refund transaction object definition

```
IndependentRefund indrefund = new IndependentRefund();
```

HttpsPostRequest object for Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(indrefund);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Independent Refund transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`indrefund.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`indrefund.setAmount(amount);`
credit card number	*String* max 20-character alpha-numeric	`indrefund.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`indrefund.setExpDate(expiry_date);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`indrefund.setCryptType(crypt);`

Independent Refund transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`indrefund.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`indrefund .setDynamicDescriptor (dynamic_descriptor);`

Sample Independent Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaIndependentRefund
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String cust_id = "my customer id";
String amount = "20.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
IndependentRefund indrefund = new IndependentRefund();
indrefund.setOrderId(order_id);
indrefund.setCustId(cust_id);
indrefund.setAmount(amount);
indrefund.setPan(pan);
indrefund.setExpdate(expdate);
indrefund.setCryptType(crypt);
indrefund.setDynamicDescriptor("123456");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(indrefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
```

Sample Independent Refund

```
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

2.9 Card Verification with AVS and CVD

Verifies the validity of the credit card, expiry date and any additional details (such as the Card Verification Digits or Address Verification details). It does not verify the available amount or lock any funds on the credit card.

Things to Consider:

- The Card Verification transaction is only supported by Visa, Mastercard, Discover and American Express
- For some Credential on File transactions, Card Verification with AVS and CVD is used as a prior step to get the Issuer ID used in the subsequent transaction
- For Card Verification, CVD is supported by Visa, MasterCard, Discover and American Express
- For Card Verification, AVS is supported by Visa, MasterCard and Discover.
- When testing Card Verification, please use the Visa and MasterCard test card numbers provided in the MasterCard Card Verification and Visa Card Verification tables available in CVD & AVS (E-Fraud) Simulator.
- For a full list of possible AVS & CVD result codes refer to the CVD and AVS Result Code tables.

Card Verification transaction object definition

```
CardVerification cardVerification = new CardVerification();
```

HttpsPostRequest object for Card Verification transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(cardVerification );
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Card Verification transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`cardVerification.setOrderId(order_id);`
credit card number	*String* max 20-character alpha-numeric	`cardVerification.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`cardVerification.setExpDate(expiry_date);`
electronic commerce indic-	*String*	`cardVerification`

Variable Name	Type and Limits	Set Method
ator	1-character alphanumeric	`.setCryptType(crypt);`
AVS Information	*Object* N/A	`cardVerification.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`cardVerification.setCvdInfo (cvdCheck);`

Card Verification transaction request fields – Optional

Variable Name	Type and Limits	Set Method
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`cardVerification.setCofInfo (cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Cre-

Variable Name	Type and Limits	Set Method
the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.		dential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Card Verification

```
package Canada;
import JavaAPI.*;
public class TestCanadaCardVerification
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String pan = "4242424242424242";
String expdate = "1901"; //YYMM format
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.setAvsStreetNumber("212");
avsCheck.setAvsStreetName("Payton Street");
avsCheck.setAvsZipCode("M1M1M1");
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.setCvdIndicator("1");
cvdCheck.setCvdValue("099");
CardVerification cardVerification = new CardVerification();
cardVerification.setOrderId(order_id);
cardVerification.setPan(pan);
cardVerification.setExpdate(expdate);
```

Sample Card Verification

```
cardVerification.setCryptType(crypt);
cardVerification.setAvsInfo(avsCheck);
cardVerification.setCvdInfo(cvdCheck);

//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cardVerification.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cardVerification);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
System.out.println("IssuerId = " + receipt.getIssuerId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

2.10 Batch Close

Takes the funds from all Purchase, Completion, Refund and Force Post transactions so that they will be deposited or debited the following business day.

For funds to be deposited the following business day, the batch must close before 11 pm Eastern Time.

Batch Close transaction object definition

```
BatchClose batchclose = new BatchClose();
```

HttpsPostRequest object for Batch Close transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(batchclose);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Optional connection object field

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Batch Close transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
ECR (electronic cash register) number	*String* No limit (value provided by Moneris)	`batchclose.setEcrno(ecr_no);`

Sample Batch Close

```
package Canada;
import JavaAPI.*;
public class TestCanadaBatchClose
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String ecr_no = "66013455"; //ecr within store
String processing_country_code = "CA";
boolean status_check = false;
```

Sample Batch Close

```
BatchClose batchclose = new BatchClose();
batchclose.setEcrno(ecr_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(batchclose);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
if ((receipt.getReceiptId()).equals("Global Error Receipt") ||
receipt.getReceiptId().equals("null") ||
receipt.getReceiptId().equals(""))
{
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = null");
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
}
else
{
for (String ecr : receipt.getTerminalIDs())
{
System.out.println("ECR: " + ecr);
for(String cardType : receipt.getCreditCards(ecr))
{
System.out.println("\tCard Type: " + cardType);
System.out.println("\t\tPurchase: Count = "
+ receipt.getPurchaseCount(ecr, cardType)
+ " Amount = "
+ receipt.getPurchaseAmount(ecr,
cardType));
System.out.println("\t\tRefund: Count = "
+ receipt.getRefundCount(ecr, cardType)
+ " Amount = "
+ receipt.getRefundAmount(ecr, cardType));
System.out.println("\t\tCorrection: Count = "
+ receipt.getCorrectionCount(ecr, cardType)
+ " Amount = "
+ receipt.getCorrectionAmount(ecr,
cardType));
}
}
}
```

Sample Batch Close
```
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

2.11 Open Totals

Returns the details about the currently open batch.

Similar to the Batch Close; the difference is that it does not close the batch for settlement.

Open Totals transaction object definition

```
OpenTotals opentotals = new OpenTotals();
```

HttpsPostRequest object for Open Totals transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(opentotals);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

Open Totals transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
ECR (electronic cash register) number	No limit (value provided by Moneris)	`opentotals.setEcrno(ecr_no);`

Sample Open Totals

```
package Canada;
import JavaAPI.*;
public class TestCanadaOpenTotals
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String ecr_no = "66013455";
//String ecr_no = "66011091";
String processing_country_code = "CA";
OpenTotals opentotals = new OpenTotals();
opentotals.setEcrno(ecr_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(opentotals);
mpgReq.send();

try
{
Receipt receipt = mpgReq.getReceipt();
if ((receipt.getReceiptId()).equals("Global Error Receipt") ||
receipt.getReceiptId().equals("null") ||
receipt.getReceiptId().equals(""))
{
System.out.println("CardType = null");
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = null");
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
}
else
{
for (String ecr : receipt.getTerminalIDs())
{
System.out.println("ECR: " + ecr);

for (String cardType : receipt.getCreditCards(ecr))
{
System.out.println("\tCard Type: " + cardType);
System.out.println("\t\tPurchase: Count = "
+ receipt.getPurchaseCount(ecr, cardType)
+ " Amount = "
+ receipt.getPurchaseAmount(ecr,
cardType));
```

Sample Open Totals

```
    System.out.println("\t\tRefund: Count = "
    + receipt.getRefundCount(ecr, cardType)
    + " Amount = "
    + receipt.getRefundAmount(ecr, cardType));
    System.out.println("\t\tCorrection: Count = "
    + receipt.getCorrectionCount(ecr, cardType)
    + " Amount = "
    + receipt.getCorrectionAmount(ecr,
    cardType));
    }
    }
    }
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

3 Credential on File

3.1 About Credential on File

When storing customers' credit card credentials for use in future authorizations, or when using these credentials in subsequent transactions, card brands now require merchants to indicate this in the transaction request.

In the Moneris API, this is handled by the Moneris Gateway via the inclusion of the Credential on File info object and its variables in the transaction request.

While the requirements for handling Credential on File transactions relate to Visa, Mastercard and Discover only, in order to avoid confusion and prevent error, please implement these changes for all card types and the Moneris system will then correctly flow the relevant card data values as appropriate.

While in the testing phase, we recommend that you test with Visa cards because implementation for the other card brands is still in process.

NOTE: If either the first transaction or a Card Verification authorization is declined when attempting to store cardholder credentials, those credentials cannot be stored —therefore the merchant must not use the credential for any subsequent transactions.

3.2 Credential on File Info Object and Variables

The Credential on File Info object is nested within the request for the applicable transaction types.

Credential on File Info Object:

cof

Variables in the cof object:

Payment Indicator
Payment Information
Issuer ID

For more information, see Definition of Request Fields – Credential on File.

3.3 Credential on File Transaction Types

The Credential on File Info object applies to the following transaction types:

- Purchase
- Pre-Authorization
- Purchase with 3-D Secure – cavvPurchase
- Purchase with 3-D Secure and Recurring Billing
- Pre-Authorization with 3-D Secure – cavvPreauth
- Purchase with Vault – ResPurchaseCC
- Pre-Authorization with Vault – ResPreauthCC
- Card Verification with AVS and CVD
- Card Verification with Vault – ResCardVerificationCC
- Vault Add Credit Card – ResAddCC
- Vault Update Credit Card – ResUpdateCC
- Vault Add Token – ResAddToken
- Vault Tokenize Credit Card – ResTokenizeCC
- Recurring Billing
- MCP Purchase
- MCP Pre-Authorization
- MCP Pre-Authorization Completion
- MCP Purchase with Vault
- MCP Pre-Authorization with Vault

3.4 Merchant- vs. Cardholder-Initiated COF Transactions

Transactions defined as Credential on File (COF) can be initiated one of two ways: by a merchant or by a cardholder. The initiator of a transaction is important because it determines which Credential on File indicator fields need to be sent in the transaction request.

Merchant-initiated Credential on File transactions: transactions in which the merchant intends to store cardholder credentials or use credentials that have already been stored. This includes sending the Credential on File Info object in the transaction request, and including all three of its fields: **issuer ID**, **payment indicator** and **payment information**.

Cardholder-initiated Credential on File transactions: transactions which are triggered by some action by the cardholder. For cardholder-initiated transactions, only the **payment indicator** and **payment information** fields are required.

For simplicity in developing your integration, the Moneris Gateway also allows cardholder-initiated transactions to be processed according to the same Credential on File rules as apply to merchant-initiated transactions. Technically, the **issuer ID** indicator is not required for cardholder-initiated transactions, but for convenience, if it is included in the transaction request, the Moneris Gateway will just ignore it when forwarding the request to the host.

3.5 Initial Transactions in Credential on File

When sending an *initial* transaction with the Credential on File Info object, i.e., a transaction request where the cardholder's credentials are being stored for the *first* time, it is important to understand the following:

- You must send the cardholder's Card Verification Digits (CVD)
- **Issuer ID** will be sent without a value on the initial transaction, because it is received in the response to that initial transaction; for all *subsequent* merchant-intiated transactions and all administrative transactions you send this **Issuer ID**
- The **payment information** field should always be set to a value of 0 on the first transaction
- The **payment indicator** field should be set to the value that is appropriate for the transaction

3.6 COF With Previously Stored Credentials

When processing a transaction with cardholder information that was already stored **prior to** the implementation of Credential on File requirements, you must:

- Include the Credential on File Info object in the transaction request, and
- Send the **payment information** value as 2, and
- Store the **issuer ID** returned in the transaction and associate it with the cardholder credentials for future use

Once the **issuer ID** has been stored and associated with the cardholder credentials, send it in all subsequent transactions going forward. **Issuer ID** is only required when sending merchant-initiated transactions.

3.7 Vault Tokenize Credit Card and Credential on File

When you want to store cardholder credentials from previous transactions into the Vault, you use the Vault Tokenize Credit Card transaction request. Credential on File rules require that only previous transactions with the Credential on File Info object can be tokenized to the Vault.

For more information about this transaction, see 4.3.10 Vault Tokenize Credit Card – ResTokenizeCC.

3.8 Credential on File and Converting Temporary Tokens

In the event you decide to convert a temporary token representing cardholder credentials into a permanent token, these credentials become stored credentials, and therefore it is necessary to send Credential on File information.

For Vault Temporary Token Add transactions where you subsequently decide to convert the temporary token into a permanent token (stored credentials):

1. Send a transaction request that includes the Credential on File Info object to get the Issuer ID; this can be a Card Verification, Purchase or Pre-Authorization request

2. After completing the transaction, send the Vault Add Token request with the Credential on File object (Issuer ID only) in order to convert the temporary token to a permanent one.

3.9 Card Verification and Credential on File Transactions

In the absence of a Purchase or Pre-Authorization, a Card Verification transaction is used to get the unique issuer ID value (**issuerId**) that is used in subsequent Credential on File transactions. Issuer ID is a variable included in the nested Credential on File Info object.

For all first-time transactions, including Card Verification transactions, you must also request the cardholder's Card Verification Details (CVD). For more on CVD, see 10.2 Card Validation Digits (CVD).

For a complete list of these variables, see each transaction type or Definition of Request Fields – Credential on File

The Card Verification request, including the Credential on File Info object, must be sent immediately prior to storing cardholder credentials.

For information about Card Verification, see 2.9 Card Verification with AVS and CVD.

3.9.1 When to Use Card Verification With COF

If you are not sending a Purchase or Pre-Authorization transaction (i.e., you are not charging the customer immediately), you must use Card Verification (or in the case of Vault Add Token, Card Verification with Vault) first before running the transaction in order to get the Issuer ID.

Transactions this applies to:

Vault Add Credit Card – ResAddCC
Vault Update Credit Card – ResUpdateCC
Vault Add Token – ResAddToken
Recurring Billing transactions, if:

- the first transaction is set to start on a future date

3.9.2 Credential on File and Vault Add Token

For Vault Add Token transactions:

1. Send Card Verification with Vault transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Add Token request including the Credential on File object (with Issuer ID only; other fields are not applicable)

For more on this transaction type, see 4.3.9 Vault Add Token – ResAddToken.

3.9.3 Credential on File and Vault Update Credit Card

For Vault Update Credit Card transactions where you are updating the credit card number:

1. Send Card Verification transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Update Credit Card request including the Credential on File Info object (Issuer ID only).

For more on this transaction type, see 4.3.3 Vault Update Credit Card – ResUpdateCC.

3.9.4 Credential on File and Vault Add Credit Card

For Vault Add Credit Card transactions:

1. Send Card Verification transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Add Credit Card request including the Credential on File Info object (Issuer ID only)

For more on this transaction type, see 4.3.1 Vault Add Credit Card – ResAddCC.

3.9.5 Credential on File and Recurring Billing

NOTE: The value of the **payment indicator** field must be **R** when sending Recurring Billing transactions.

For Recurring Billing transactions which are set to start **immediately**:

1. Send a Purchase transaction request with both the Recurring Billing and Credential on File info objects (with Recurring Billing object field **start now** = true)

For Recurring Billing transactions which are set to start on a **future** date:

1. Send Card Verification transaction request including the Credential on File info object to get the Issuer ID
2. Send Purchase transaction request with the Recur and Credential on File info objects included

For updating a Recurring Billing series where you are updating the card number (does not apply if you are only modifying the schedule or amount in a recurring series):

1. Send Card Verification request including the Credential on File info object to get the Issuer ID
2. Send a Recurring Billing Update transaction

For more information about the Recurring Billing object, see Definition of Request Fields – Recurring.

4 Vault

4.1 About the Vault Transaction Set

The Vault feature allows merchants to create customer profiles, edit those profiles, and use them to process transactions without having to enter financial information each time. Customer profiles store customer data essential to processing transactions, including credit and signature debit.

The Vault is a complement to the Recurring Billing module. It securely stores customer account information on Moneris secure servers. This allows merchants to bill customers for routine products or services when an invoice is due.

4.2 Vault Transaction Types

The Vault API supports both administrative and financial transactions.

4.2.1 Administrative Vault Transaction types

ResAddCC
Creates a new credit card profile, and generates a unique data key which can be obtained from the Receipt object.

This data key is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

EncResAddCC
Creates a new credit card profile, but requires the card data to be either swiped or manually keyed in via a Moneris-provided encrypted mag swipe reader.

ResTempAdd
Creates a new temporary token credit card profile. This transaction requires a duration to be set to indicate how long the temporary token is to be stored for.

During the lifetime of this temporary token, it may be used for any other vault transaction before it is permanently deleted from the system.

ResUpdateCC
Updates a Vault profile (based on the data key) to contain credit card information.

All information contained within a credit card profile is updated as indicated by the submitted fields.

EncResUpdateCC
Updates a profile (based on the data key) to contain credit card information. The encrypted version of this transaction requires the card data to either be swiped or manually keyed in via a Moneris-provided encrypted mag swipe reader.

ResDelete

Deletes an existing Vault profile of any type using the unique data key that was assigned when the profile was added.

It is important to note that after a profile is deleted, the information which was saved within can no longer be retrieved.

ResLookupFull

Verifies what is currently saved under the Vault profile associated with the given data key. The response to this transaction returns the latest active data for that profile.

Unlike ResLookupMasked (which returns the masked credit card number), this transaction returns both the masked and the unmasked credit card numbers.

ResLookupMasked

Verifies what is currently saved under the Vault profile associated with the given data key. The response to this transaction returns the latest active data for that profile.

Unlike ResLookupFull (which only returns both the masked and the unmasked credit card numbers), this transaction only returns the masked credit card number.

ResGetExpiring

Verifies which profiles have credit cards that are expiring during the current and next calendar month. For example, if you are processing this transaction on September 30, then it will return all cards that expire(d) in September and October of this year.

When generating a list of profiles with expiring credit cards, only the **masked** credit card numbers are returned.

This transaction can be performed no more than 2 times on any given calendar day, and it only applies to credit card profiles.

ResIscorporatecard

Determines whether a profile has a corporate card registered within it.

After sending the transaction, the response field to the Receipt object's getCorporateCard method is either `true` or `false` depending on whether the associated card is a corporate card.

ResAddToken

Converts a Hosted Tokenization temporary token to a permanent Vault token.

A temporary token is valid for 15 minutes after it is created.

ResTokenizeCC

Creates a new credit card profile using the credit card number, expiry date and e-commerce indicator that were submitted in a previous financial transaction. A transaction that was previously done in Moneris Gateway is taken, and the card data from that transaction is stored in the Moneris Vault.

As with ResAddCC, a unique data key is generated and returned to the merchant via the Receipt object. This is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

4.2.2 Financial Vault Transaction types

ResPurchaseCC
Uses the data key to identify a previously registered credit card profile. The details saved within the profile are then submitted to perform a Purchase transaction.

ResPreauthCC
Uses the data key to identify a previously registered credit card profile. The details within the profile are submitted to perform a Pre-Authorization transaction.

ResIndRefundCC
Uses the unique data key to identify a previously registered credit card profile, and credits a specified amount to that credit card.

ResMpiTxn
Uses the data key (as opposed to a credit card number) in a VBV/SecureCode Txn MPI transaction. The merchant uses the data key with ResMpiTxn request, and then reads the response fields to verify whether the card is enrolled in Verified by Visa or MasterCard SecureCode. Retrieves the vault transaction value to pass on to Visa or MasterCard.

After it has been validated that the data key is is enrolled in 3-D Secure, a window appears in which the customer can enter the 3-D Secure password. The merchant may initiate the forming of the validation form `getMpiInLineForm()`.

For more information on integrating with MonerisMPI, refer to 8 MPI

4.3 Vault Administrative Transactions

Administrative transactions allow you to perform such tasks as creating new Vault profiles, deleting existing Vault profiles and updating profile information.

Some Vault Administrative Transactions require the Credential on File object to be sent with the **issuer ID** field only.

4.3.1 Vault Add Credit Card – ResAddCC

Creates a new credit card profile, and generates a unique data key which can be obtained from the Receipt object.

This data key is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

Vault Add Credit Card transaction object definition

```
ResAddCC resaddcc = new ResAddCC();
```

HttpsPostRequest object for Vault Add Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resaddcc);
```

Vault Add Credit Card transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
credit card number	*String* max 20-character alpha-numeric	`resaddcc.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`resaddcc.setExpDate(expiry_date);`
electronic commerce indicator	*String* 1-character alphanumeric	`resaddcc.setCryptType(crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`resaddcc.setCofInfo(cof);`

Vault Add Credit Card transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`resaddcc.setCustId(cust_id);`
AVS Information	*Object* N/A	`resaddcc.setAvsInfo(avsCheck);`
email address	*String* 30-character alphanumeric	`resaddcc.setEmail(email);`

Variable Name	Type and Limits	Set Method
phone number	*String* 30-character alphanumeric	`resaddcc.setPhone(phone);`
note	*String* 30-character alphanumeric	`resaddcc.setNote(note);`
data key format	*String* 2-character alphanumeric	`resaddcc.setDataKeyFormat (data_key_format);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Vault Add Credit Card

```
package Canada;
import JavaAPI.*;
public class TestCanadaResAddCC
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String pan = "4242424242424242";
String expdate = "1912";
String phone = "0000000000";
String email = "bob@smith.com";
String note = "my note";
String cust_id = "customer1";
String crypt_type = "7";
String data_key_format = "0";
String processing_country_code = "CA";
boolean status_check = false;
```

Sample Vault Add Credit Card

```
    AvsInfo avsCheck = new AvsInfo();
    avsCheck.setAvsStreetNumber("212");
    avsCheck.setAvsStreetName("Payton Street");
    avsCheck.setAvsZipCode("M1M1M1");
    ResAddCC resaddcc = new ResAddCC();
    resaddcc.setPan(pan);
    resaddcc.setExpdate(expdate);
    resaddcc.setCryptType(crypt_type);
    resaddcc.setCustId(cust_id);
    resaddcc.setPhone(phone);
    resaddcc.setEmail(email);
    resaddcc.setNote(note);
    resaddcc.setAvsInfo(avsCheck);
    //resaddcc.setDataKeyFormat(data_key_format); //optional
    //Mandatory - Credential on File details
    CofInfo cof = new CofInfo();
    cof.setIssuerId("139X3130ASCXAS9"); //can be obtained by performing card verification

    resaddcc.setCofInfo(cof);

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resaddcc);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("MaskedPan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.1.1 Vault Data Key

The ResAddCC sample code includes the following instruction from the Receipt object:

```
System.out.println("DataKey = " + receipt.getDataKey());
```

The data key response field is populated when you send a Vault Add Credit Card – ResAddCC (page 59), Vault Encrypted Add Credit Card – EncResAddCC (page 63), Vault Tokenize Credit Card – ResTokenizeCC (page 86), Vault Temporary Token Add – ResTempAdd (page 65) or Vault Add Token – ResAddToken (page 82) transaction. It is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

The data key is a maximum 28-character alphanumeric string.

4.3.1.2 Vault Encrypted Add Credit Card – EncResAddCC

Vault Encrypted Add Credit Card transaction object definition

```
EncResAddCC encresaddcc = new EncResAddCC();
```

HttpsPostRequest object for Vault Encrypted Add Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(encresaddcc);
```

Vault Encrypted Add Credit Card transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
encrypted track 2 data	*String* 40-character numeric	`encresaddcc.setEncTrack2 (enc_track2);`
device type	*String* 30-character alphanumeric	`encresaddcc.setDeviceType (device_type);`
electronic commerce indicator	*String* 1-character alphanumeric	`encresaddcc.setCryptType (crypt);`

Vault Encrypted Add Credit Card transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`encresaddcc.setCustId(cust_id);`
AVS Information	*Object* N/A	`encresaddcc.setAvsInfo(avsCheck);`
email address	*String* 30-character alphanumeric	`encresaddcc.setEmail(email);`
phone number	*String* 30-character alphanumeric	`encresaddcc.setPhone(phone);`
note	*String* 30-character alphanumeric	`encresaddcc.setNote(note);`
data key format	*String* 2-character alphanumeric	`encresaddcc.setDataKeyFormat(data_key_format);`

Sample Vault Encrypted Add Credit Card

```
package Canada;
import JavaAPI.*;
public class TestCanadaEncResAddCC
{
public static void main(String args[])
{
String store_id = "moneris";
String api_token = "hurgle";
String enc_track2 = "ENCRYPTEDTRACK2DATA";
String device_type = "idtech_bdk";
String phone = "55555555555";
String email = "test.user@moneris.com";
String note = "my note";
String cust_id = "customer2";
String crypt = "7";
String processing_country_code = "CA";

AvsInfo avsCheck = new AvsInfo();
avsCheck.setAvsStreetNumber("212");
avsCheck.setAvsStreetName("Payton Street");
avsCheck.setAvsZipcode("M1M1M1");
EncResAddCC enc_res_add_cc = new EncResAddCC ();
enc_res_add_cc.setEncTrack2(enc_track2);
enc_res_add_cc.setDeviceType(device_type);
enc_res_add_cc.setCryptType(crypt);
```

Sample Vault Encrypted Add Credit Card

```
    enc_res_add_cc.setCustId(cust_id);
    enc_res_add_cc.setPhone(phone);
    enc_res_add_cc.setEmail(email);
    enc_res_add_cc.setNote(note);
    //enc_res_add_cc.setAvsInfo(avsCheck);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(enc_res_add_cc);
    mpgReq.send();

    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType() + "\n");
    //Contents of ResolveData
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("MaskedPan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpDate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.2 Vault Temporary Token Add – ResTempAdd

Creates a new temporary token credit card profile. This transaction requires a duration to be set to indicate how long the temporary token is to be stored for.

During the lifetime of this temporary token, it may be used for any other Vault transaction before it is permanently deleted from the system.

Things to Consider:

- The duration, or lifetime, of the temporary token can be set to be a maximum of 15 minutes.

Vault Temporary Token Add transaction object definition

```
ResTempAdd resTempAdd = new ResTempAdd();
```

HttpsPostRequest object for Vault Temporary Token Add transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resTempAdd);
```

Vault Temporary Token Add transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
credit card number	*String* max 20-character alpha-numeric	`resTempAdd.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`resTempAdd.setExpDate (expiry_date);`
duration	*String* 3-character numeric maximum 15 minutes	`resTempAdd.setDuration (duration);`
electronic commerce indicator	*String* 1-character alphanumeric	`resTempAdd.setCryptType (crypt);`

Vault Temporary Token Add transaction request fields – Optional

Variable Name	Type and Limits	Set Method
data key format	*String*	`resTempAdd.setDataKeyFormat`

Variable Name	Type and Limits	Set Method
	2-character alphanumeric	`(data_key_format);`

Sample Vault Temporary Token Add

```
package Canada;
import JavaAPI.*;
public class TestCanadaResTempAdd
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String pan = "5454545454545454";
String expdate = "1901"; //YYMM format
String crypt_type = "7";
String duration = "900";
String processing_country_code = "CA";
boolean status_check = false;
ResTempAdd resTempAdd = new ResTempAdd();
resTempAdd.setPan(pan);
resTempAdd.setExpdate(expdate);
resTempAdd.setDuration(duration);
resTempAdd.setCryptType(crypt_type);
// resTempAdd.setDataKeyFormat("0U");

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resTempAdd);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.3 Vault Update Credit Card – ResUpdateCC

Updates an existing Vault profile (referencing the profile's unique **data key**) with cardholder information.

Information contained within a credit card profile is updated as indicated by the submitted fields; if any field representing an item of cardholder information is not sent in this request, that item will remain unchanged in the profile.

If the Vault profile is being updated with a new credit card number, then you first need to send a Purchase, Pre-Authorization or Card Verification transaction, with the Credential on File Info object included, before performing Vault Update Credit Card. If the credit card number is not one of the profile items being updated, this step is not required.

Things to Consider:

- To update a specific element in the profile, set that element using the corresponding set method
- When updating a credit card number, first send a Purchase, Pre-Authorization, or Card Verification with the Credential on File Info object before sending this transaction; send the issuer ID received in the response in the subsequent Vault Update Credit Card request
- If the credit card number is not one of the profile items being updated, the Credential on File info object is not required

Vault Update Credit Card transaction object definition

```
ResUpdateCC resUpdateCC = new ResUpdateCC();
```

HttpsPostRequest object for Vault Update Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(resUpdateCC);
```

Vault Update Credit Card transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resUpdateCC.setData(data_key);`

Optional values that are submitted to the ResUpdateCC object are updated. Unsubmitted optional values (with one exception) remain unchanged. This allows you to change only the fields you want.

If a profile contains AVS information, but a Vault Update Credit Card transaction is submitted without an AVS Info object, the existing AVS Info details are deactivated and the new credit card information is registered without AVS.

Vault Update Credit Card transaction request fields – Optional

Variable Name	Type and Limits	Set Method
credit card number	*String* 20-character alphanumeric	`resUpdateCC.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`resUpdateCC.setExpDate (expiry_date);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`resUpdateCC.setCryptType (crypt);`
customer ID	*String* 30-character alphanumeric	`resUpdateCC.setCustId(cust_ id);`
AVS Information	*String* N/A	`resUpdateCC.setAvsInfo (avsCheck);`
email address	*String* 30-character alphanumeric	`resUpdateCC.setEmail(email);`
Phone number	*String* 30-character alphanumeric	`resUpdateCC.setPhone(phone);`
Note	*String* 30-character alphanumeric	`resUpdateCC.setNote(note);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`resUpdateCC.setCofInfo(cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Vault Update Credit Card

```
package Canada;
import JavaAPI.*;
public class TestCanadaResUpdateCC
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String data_key = "vthBJyN1BicbRkdWFZ9flyDP2";
String pan = "4242424242424242";
String expdate = "1901";
String phone = "0000000000";
String email = "bob@smith.com";
String note = "my note";
String cust_id = "customer1";
String crypt_type = "7";
String processing_country_code = "CA";
boolean status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.setAvsStreetNumber("212");
avsCheck.setAvsStreetName("Payton Street");
avsCheck.setAvsZipCode("M1M1M1");
//Credential on File details
CofInfo cof = new CofInfo();
cof.setIssuerId("139X3130ASCXAS9");

ResUpdateCC resUpdateCC = new ResUpdateCC();
resUpdateCC.setData(data_key);
resUpdateCC.setAvsInfo(avsCheck);
resUpdateCC.setCustId(cust_id);
resUpdateCC.setPan(pan);
resUpdateCC.setExpdate(expdate);
resUpdateCC.setPhone(phone);
resUpdateCC.setEmail(email);
resUpdateCC.setNote(note);
resUpdateCC.setCryptType(crypt_type);
resUpdateCC.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
```

Sample Vault Update Credit Card

```
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resUpdateCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.3.1 Vault Encrypted Update CC – EncResUpdateCC

Vault Encrypted Update CC transaction object definition

```
EncResUpdateCC enc_res_update_cc = new EncResUpdateCC ();
```

HttpsPostRequest object for Vault Encrypted Update CC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(enc_res_update_cc);
```

Vault Encrypted Update CC transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`enc_res_update_cc.setData (data_key);`
encrypted track 2 data	*String* 40-character numeric	`enc_res_update_ cc.setEncTrack2(enc_track2);`
device type	*String* 30-character alphanumeric	`enc_res_update_ cc.setDeviceType(device_ type);`

Optional values that are submitted to the ResUpdateCC object are updated, while unsubmitted optional values (with one exception) remain unchanged. This allows you to change only the fields you want.

The exception is that if you are making changes to the payment type, **all** of the variables in the optional values table below must be submitted.

If you update a profile to a different payment type, it is automatically deactivated and a new credit card profile is created and assigned to the data key. The only values from the prior profile that will remain unchanged are the customer ID, phone number, email address, and note.

EXAMPLE: If a profile contains AVS information, but a ResUpdateCC transaction is submitted without an AVSInfo object, the existing AVSInfo details are deactivated and the new credit card information is registered without AVS.

Vault Encrypted Update CC transaction request fields – Optional

Variable Name	Type and Limits	Set Method
electronic commerce indicator	*String* 1-character alphanumeric	`enc_res_update_ cc.setCryptType(crypt);`
customer ID	*String* 30-character alphanumeric	`enc_res_update_cc.setCustId (cust_id);`
electronic commerce indicator	*String* 1-character alphanumeric	`enc_res_update_ cc.setCryptType(crypt);`
email address	*String* 30-character alphanumeric	`enc_res_update_cc.setEmail (email);`
phone number	*String*	`enc_res_update_cc.setPhone`

Variable Name	Type and Limits	Set Method
	30-character alphanumeric	`(phone);`
note	*String* 30-character alphanumeric	`enc_res_update_cc.setNote (note);`

Sample Vault Encrypted Update CC

```
package Canada;
import JavaAPI.*;
public class TestCanadaEncResUpdateCC
{
public static void main(String args[])
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "PHTM1pun7VOaSCFM2xdeP2Sim";
String enc_track2 = "ENCRYPTEDTRACK2DATA";
String device_type = "idtech_bdk";
String phone = "55555555555";
String email = "test.user@moneris.com";
String note = "my note";
String cust_id = "customer2";
String crypt = "7";
String processing_country_code = "CA";
AvsInfo avsinfo = new AvsInfo();
avsinfo.setAvsStreetNumber("212");
avsinfo.setAvsStreetName("Smith Street");
avsinfo.setAvsZipcode("M1M1M1");
EncResUpdateCC enc_res_update_cc = new EncResUpdateCC ();
enc_res_update_cc.setDataKey(data_key);
enc_res_update_cc.setAvsInfo(avsinfo);
enc_res_update_cc.setCustId(cust_id);
enc_res_update_cc.setEncTrack2(enc_track2);
enc_res_update_cc.setDeviceType(device_type);
enc_res_update_cc.setPhone(phone);
enc_res_update_cc.setEmail(email);
enc_res_update_cc.setNote(note);
enc_res_update_cc.setCryptType(crypt);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(enc_res_update_cc);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
```

Sample Vault Encrypted Update CC

```
System.out.println("PaymentType = " + receipt.getPaymentType() + "\n");
//Contents of ResolveData
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpDate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.4 Vault Delete – ResDelete

NOTE: After a profile has been deleted, the details can no longer be retrieved.

Vault Delete transaction object definition

```
ResDelete resDelete = new ResDelete (data_key);
```

HttpsPostRequest object for Vault Delete transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resDelete);
```

Vault Delete transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resDelete.setData(data_key);`

Sample Vault Delete

```
package Canada;
import JavaAPI.*;
public class TestCanadaResDelete
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String data_key = "DxwdemrvfnoXO1HhmRikfw3gA";
String processing_country_code = "CA";
boolean status_check = false;
ResDelete resDelete = new ResDelete(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resDelete);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
//ResolveData
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.5 Vault Lookup Full – ResLookupFull

Vault Lookup Full transaction object definition

```
ResLookupFull resLookupFull = new ResLookupFull(data_key);
```

HttpsPostRequest object for Vault Lookup Full transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resLookupFull);
```

Vault Lookup Full transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resLookupFull.setData(data_key);`

Sample Vault Lookup Full

```
package Canada;
import JavaAPI.*;
public class TestCanadaResLookupFull
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "pi3ZMZoTTM8pLM9wuwws2KBxw";
String processing_country_code = "CA";
boolean status_check = false;
ResLookupFull resLookupFull = new ResLookupFull(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resLookupFull);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
```

Sample Vault Lookup Full

```
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Pan = " + receipt.getResPan());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.6 Vault Lookup Masked – ResLookupMasked

Vault Lookup Masked transaction object definition

```
ResLookupMasked resLookupMasked = new ResLookupMasked();
```

HttpsPostRequest object for Vault Lookup Masked transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resLookupMasked);
```

Vault Lookup Masked transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resLookupMasked.setData (data_key);`

Sample Vault Lookup Masked

```
package Canada;
import JavaAPI.*;
public class TestCanadaResLookupMasked
{
```

Sample Vault Lookup Masked

```
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "pi3ZMZoTTM8pLM9wuwws2KBxw";
String processing_country_code = "CA";
boolean status_check = false;
ResLookupMasked resLookupMasked = new ResLookupMasked();
resLookupMasked.setData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resLookupMasked);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.7 Vault Get Expiring – ResGetExpiring

Vault Get Expiring transaction object definition

```
ResGetExpiring resGetExpiring = new ResGetExpiring();
```

HttpsPostRequest object for Vault Get Expiring transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resGetExpiring);
```

Vault Get Expiring transaction request fields – Required

This transaction has no required request fields.

Sample Vault Get Expiring

```
package Canada;
import JavaAPI.*;
public class TestCanadaResGetExpiring
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String processing_country_code = "CA";
boolean status_check = false;
ResGetExpiring resGetExpiring = new ResGetExpiring();
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resGetExpiring);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
//ResolveData
for (int index =0; index < receipt.getExpiredCardCount(); index++)
{
System.out.println("\nDataKey = " + index);
System.out.println("Payment Type = " + receipt.getExpPaymentType(index));
System.out.println("Cust ID = " + receipt.getExpCustId(index));
System.out.println("Phone = " + receipt.getExpPhone(index));
System.out.println("Email = " + receipt.getExpEmail(index));
System.out.println("Note = " + receipt.getExpNote(index));
System.out.println("Masked Pan = " + receipt.getExpMaskedPan(index));
System.out.println("Exp Date = " + receipt.getExpExpdate(index));
System.out.println("Crypt Type = " + receipt.getExpCryptType(index));
System.out.println("Avs Street Number = " + receipt.getExpAvsStreetNumber(index));
System.out.println("Avs Street Name = " + receipt.getExpAvsStreetName(index));
System.out.println("Avs Zipcode = " + receipt.getExpAvsZipCode(index));
}
}
catch (Exception e)
```

Sample Vault Get Expiring

```
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.8 Vault Is Corporate Card - ResIscorporateCard

Vault Is Corporate Card transaction object definition

```
ResIscorporatecard resIscorporatecard = new ResIscorporatecard();
```

HttpsPostRequest object for Vault Is Corporate Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(ResIscorporateCard);
```

Vault Is Corporate Card transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resIscorporatecard.setData (data_key);`

Sample Vault Is Corporate Card

```
    package Canada;
    import JavaAPI.*;
    public class TestCanadaResIscorporatecard
    {
    public static void main(String[] args)
    {
    String store_id = "store1";
    String api_token = "yesguy";
    String data_key = "eLqsADfwqHDxIpJG9vLnELx01";
    String processing_country_code = "CA";
    boolean status_check = false;
    ResIscorporatecard resIscorporatecard = new ResIscorporatecard();
    resIscorporatecard.setData(data_key);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
```

Sample Vault Is Corporate Card

```
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resIscorporatecard);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("CorporateCard = " + receipt.getCorporateCard());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.9 Vault Add Token – ResAddToken

This transaction is used to convert a temporary token into a permanent token for storage in the Moneris Vault

Things to Consider:

-
- If you intend to store the token for use in future transactions (i.e., Credential on File transactions), **first** you must send either a Vault financial transaction (Purchase with Vault or Pre-Authorization with Vault) or a Card Verification with Vault in order to get the Issuer ID
- The Vault Add Token request uses the Issuer ID to indicate that it is referencing stored credentials

Vault Add Token transaction object definition

```
ResAddToken resAddToken = new ResAddToken();
```

HttpsPostRequest object for Vault Add Token transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(resAddToken);
```

Vault Add Token transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 1: Vault Add Token transaction object mandatory values

Value	Limits	Set method
data key	*String* 25-character alphanumeric	`resAddToken.setData(data_key);`
electronic commerce indicator	*String* 1-character alphanumeric	`resAddToken.setCryptType(crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`cof.setCofInfo(cof);`

Vault Add Token transaction request fields – Optional

Table 2: Vault Add Token transaction optional values

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`resAddToken.setCustId(cust_id);`
AVS Information	*Object* N/A	`resAddToken.setAvsInfo(avsCheck);`
email address	*String*	`resAddToken.setEmail(email);`

Variable Name	Type and Limits	Set Method
	30-character alphanumeric	
phone number	*String* 30-character alphanumeric	`resAddToken.setPhone(phone);`
note	*String* 30-character alphanumeric	`resAddToken.setNote(note);`
data key format	*String* 2-character alphanumeric	`resAddToken.setDataKeyFormat (data_key_format);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Vault Add Token

```
package Canada;
import JavaAPI.*;
public class TestCanadaResAddToken
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "ot-545454ucx87A5454";
String expdate = "2001";
String phone = "0000000000";
String email = "bob@smith.com";
String note = "my note";
String cust_id = "customer1";
String crypt_type = "7";
String data_key_format = "0";
String processing_country_code = "CA";
```

Sample Vault Add Token

```
    boolean status_check = false;
    AvsInfo avsCheck = new AvsInfo();
    avsCheck.setAvsStreetNumber("212");
    avsCheck.setAvsStreetName("Payton Street");
    avsCheck.setAvsZipCode("M1M1M1");

    //Credential on File details
    CofInfo cof = new CofInfo();
    cof.setIssuerId("139X3130ASCXAS9");
    ResAddToken resAddToken = new ResAddToken();
    resAddToken.setDataKey(data_key);
    resAddToken.setCryptType(crypt_type);
    resAddToken.setExpdate(expdate);
    resAddToken.setCustId(cust_id);
    resAddToken.setPhone(phone);
    resAddToken.setEmail(email);
    resAddToken.setNote(note);
    resAddToken.setAvsInfo(avsCheck);
    resAddToken.setCofInfo(cof);
    //resAddToken.setDataKeyFormat(data_key_format); //optional
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resAddToken);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("MaskedPan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.3.10 Vault Tokenize Credit Card – ResTokenizeCC

Creates a new credit card profile using the credit card number, expiry date and e-commerce indicator that were submitted in a previous financial transaction. Previous transactions to be tokenized must have included the Credential on File Info object.

The Issuer ID received in the previous transaction response is sent in the Vault Tokenize Credit Card request to reference that this is a stored credential.

Basic transactions that can be tokenized are:

- Purchase
- Pre-Authorization
- Card Verification

The tokenization process is outlined below :

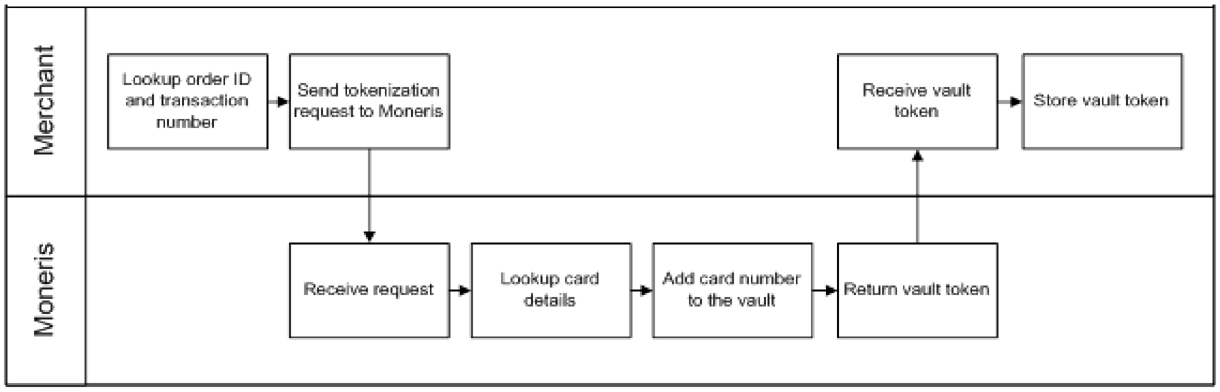


Figure 1: Tokenize process diagram

Vault Tokenize Credit Card transaction object definition

```
ResTokenizeCC resTokenizeCC = new ResTokenizeCC();
```

HttpsPostRequest object for Vault Tokenize Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resTokenizeCC);
```

Vault Tokenize Credit Card transaction request fields – Required

These mandatory values reference a previously processed credit card financial transaction. The credit card number, expiry date, and e-commerce indicator from the original transaction are registered in the Vault for future financial Vault transactions.

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resTokenizeCC.setOrderId (order_id);`
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores variable length	`resTokenizeCC.setTxnNumber (txn_number);`

Vault Tokenize Credit Card transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`resTokenizeCC.setCustId (cust_id);`
email address	*String* 30-character alphanumeric	`resTokenizeCC.setEmail (email);`
phone number	*String* 30-character alphanumeric	`resTokenizeCC.setPhone (phone);`
note	*String* 30-character alphanumeric	`resTokenizeCC.setNote(note);`
AVS Information	*Object* N/A	`resTokenizeCC.setAvsInfo (avsCheck);`
data key format	*String* 2-character alphanumeric	`resTokenizeCC .setDataKeyFormat(data_key_ format);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and	*Object* N/A	`cof.setCofInfo(cof);`

Variable Name	Type and Limits	Set Method
required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.		

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Any field that is not set in the tokenize request is not stored with the transaction. That is, Moneris Gateway does not automatically take the optional information that was part of the original transaction.

The ResolveData that is returned in the response fields indicates what values were registered for this profile.

Sample Vault Tokenize Credit Card

```
package Canada;
import JavaAPI.*;
public class TestCanadaResTokenizeCC
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String order_id = "mvt3212954335";
String txn_number = "199999-0_10";
String phone = "0000000000";
String email = "bob@smith.com";
String note = "my note";
String cust_id = "customer1";
String data_key_format = "0";
String processing_country_code = "CA";
```

Sample Vault Tokenize Credit Card

```
boolean status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.setAvsStreetNumber("212");
avsCheck.setAvsStreetName("Payton Street");
avsCheck.setAvsZipCode("M1M1M1");

//Credential on File details
CofInfo cof = new CofInfo();
cof.setIssuerId("139X3130ASCXAS9");
ResTokenizeCC resTokenizeCC = new ResTokenizeCC();
resTokenizeCC.setOrderId(order_id);
resTokenizeCC.setTxnNumber(txn_number);
resTokenizeCC.setCustId(cust_id);
resTokenizeCC.setPhone(phone);
resTokenizeCC.setEmail(email);
resTokenizeCC.setNote(note);
resTokenizeCC.setAvsInfo(avsCheck);
resTokenizeCC.setCofInfo(cof);
//resTokenizeCC.setDataKeyFormat(data_key_format); //optional

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resTokenizeCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
//ResolveData
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("MaskedPan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.4 Vault Financial Transactions

After a financial transaction is complete, the response fields indicate all the values that are currently saved under the profile that was used.

4.4.1 Customer ID Changes

Some financial transactions take the customer ID as an optional value. The customer ID may or may not already be in the Vault profile when the transaction is sent. Therefore, it is possible to change the value of the customer ID by performing a financial transaction

The table below shows what the customer ID will be in the response field after a financial transaction is performed.

Table 3: Customer ID use in response fields

Already in profile?	Passed in?	Version used in response
No	No	Customer ID not used in transaction
No	Yes	Passed in
Yes	No	Profile
Yes	Yes	Passed in

4.4.2 Purchase with Vault – ResPurchaseCC

Purchase with Vault transaction object definition

```
ResPurchaseCC resPurchaseCC = new ResPurchaseCC();
```

HttpsPostRequest object for Purchase with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resPurchaseCC);
```

Purchase with Vault transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resPurchaseCC.setData(data_key);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resPurchaseCC.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resPurchaseCC.setAmount(amount);`
electronic commerce indicator	*String* 1-character alphanumeric	`resPurchaseCC.setCryptType(crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`cof.setCofInfo(cof);`

Purchase with Vault transaction request fields – Optional

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Variable Name	Type and Limits	Set Method
expiry date	*String* 4-character numeric YYMM format. (Note that this is reversed from the date displayed on the card, which is MMYY)	`resPurchaseCC.setExpDate (expiry_date);`
customer ID	*String* 30-character alphanumeric	`resPurchaseCC.setCustId (cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`resPurchaseCC .setDynamicDescriptor (dynamic_descriptor);`
Customer Information	*Object* N/A	`resPurchaseCC.setCustInfo (customer);`
AVS Information	*Object* N/A	`resPurchaseCC.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`resPurchaseCC.setCvdInfo (cvdCheck);`
Recurring Billing	*Object* N/A	`resPurchaseCC.setRecurInfo (recurInfo);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Purchase with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaResPurchaseCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "8OOXGiwxgvfbZngigVFeld9d2";
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String crypt_type = "1";
String descriptor = "my descriptor";
String processing_country_code = "CA";
String expdate = "1512"; //For Temp Token
```

Sample Purchase with Vault

```
    boolean status_check = false;
    ResPurchaseCC resPurchaseCC = new ResPurchaseCC();
    resPurchaseCC.setData(data_key);
    resPurchaseCC.setOrderId(order_id);
    resPurchaseCC.setCustId(cust_id);
    resPurchaseCC.setAmount(amount);
    resPurchaseCC.setCryptType(crypt_type);
    //resPurchaseCC.setDynamicDescriptor(descriptor);
    //resPurchaseCC.setExpDate(expdate); //Temp Tokens only
    //Mandatory - Credential on File details
    CofInfo cof = new CofInfo();
    cof.setPaymentIndicator("U");
    cof.setPaymentInformation("2");
    cof.setIssuerId("139X3130ASCXAS9");

    resPurchaseCC.setCofInfo(cof);

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resPurchaseCC);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("Masked Pan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
```

Sample Purchase with Vault

```
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.4.3 Pre-Authorization with Vault – ResPreauthCC

Pre-Authorization with Vault transaction object definition

```
ResPreauthCC resPreauthCC = new ResPreauthCC();
```

HttpsPostRequest object for Pre-Authorization with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(resPreauthCC);
```

Pre-Authorization with Vault transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resPreauthCC.setData(data_key);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resPreauthCC.setOrderId(order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resPreauthCC.setAmount(amount);`
electronic commerce indic-	*String*	`resPreauthCC.setCryptType`

Variable Name	Type and Limits	Set Method
ator	1-character alphanumeric	`(crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`resPreauthCC.setCofInfo (cof);`

Pre-Authorization with Vault transaction request fields – Optional

Value	Limits	Set method
status check	*Boolean* true/false	`mpgReq.setStatusCheck (status_check);`
expiry date	*String* 4-character alphanumeric YYMM	`resPreauthCC.setExpDate (expiry_date);`
customer ID	*String* 30-character alphanumeric	`resPreauthCC.setCustId(cust_ id);`
Customer Information	*Object* N/A	`resPreauthCC.setCustInfo (customer);`
AVS Information	*Object* N/A	`resPreauthCC.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent trans-	*Object* N/A	`resPreauthCC.setCvdInfo (cvdCheck);`

Value	Limits	Set method
actions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.		

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Pre-Authorization with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaResPreauthCC
{
```

Sample Pre-Authorization with Vault

```
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "rS7DbroQHJmJxdBfXFXiauQc4";
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String crypt_type = "1";
String dynamic_descriptor = "my descriptor";
String processing_country_code = "CA";
String expdate = "1712"; //For Temp Token
boolean status_check = false;
ResPreauthCC resPreauthCC = new ResPreauthCC();
resPreauthCC.setData(data_key);
resPreauthCC.setOrderId(order_id);
resPreauthCC.setCustId(cust_id);
resPreauthCC.setAmount(amount);
resPreauthCC.setCryptType(crypt_type);
resPreauthCC.setDynamicDescriptor(dynamic_descriptor);
//resPreauthCC.setExpDate(expdate); //Temp Tokens only

//Mandatory - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

resPreauthCC.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resPreauthCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
```

Sample Pre-Authorization with Vault

```
    System.out.println("IsCorporate = " + receipt.getCorporateCard());
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("Masked Pan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.4.4 Vault Independent Refund CC – ResIndRefundCC

Vault Independent Refund transaction object definition

```
ResIndRefundCC resIndRefundCC = new ResIndRefundCC();
```

HttpsPostRequest object for Vault Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resIndRefundCC);
```

Vault Independent Refund transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resIndRefundCC.setData(data_key);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resIndRefundCC.setOrderId(order_id);`

Variable Name	Type and Limits	Set Method
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resIndRefundCC.setAmount (amount);`
electronic commerce indicator	*String* 1-character alphanumeric	`resIndRefundCC.setCryptType (crypt);`

Vault Independent Refund transaction request fields – Optional

Value	Limits	Set method
customer ID	*String* 30-character alphanumeric	`resIndRefundCC.setCustId (cust_id);`
expiry date	*String* 4-character alphanumeric YYMM	`resIndRefundCC.setExpDate (expiry_date);`
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`resIndRefundCC .setDynamicDescriptor (dynamic_descriptor);`

Sample Vault Independent Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaResIndRefundCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "moneris";
String api_token = "hurgle";
String data_key = "eRNr6lU1RD6jmgS9OPqmmbVrk";
String amount = "1.00";
String cust_id = "customer1";
String crypt_type = "1";
String processing_country_code = "CA";
boolean status_check = false;
ResIndRefundCC resIndRefundCC = new ResIndRefundCC();
resIndRefundCC.setOrderId(order_id);
resIndRefundCC.setCustId(cust_id);
resIndRefundCC.setAmount(amount);
resIndRefundCC.setCryptType(crypt_type);
resIndRefundCC.setData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resIndRefundCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Masked Pan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
```

Sample Vault Independent Refund

```
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

4.4.5 Force Post with Vault – ResForcePostCC

Force Post with Vault transaction object definition

```
ResForcePostCC resForcePostCC = new ResForcePostCC();
```

HttpsPostRequest object for Force Post with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resForcePostCC);
```

Force Post with Vault transaction request fields – Required

Variable Name	Type and Limits	Set Method
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resForcePostCC.setAmount (amount);`
data key	*String* 25-character alphanumeric	`resForcePostCC.setData(data_key);`
authorization code	*String* 8-character alphanumeric	`resForcePostCC.setAuthCode (auth_code);`

Variable Name	Type and Limits	Set Method
electronic commerce indicator	*String* 1-character alphanumeric	`resForcePostCC.setCryptType (crypt);`

Force Post with Vault transaction object definition

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`resForcePostCC.setCustId (cust_id);`
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`resForcePostCC .setDynamicDescriptor (dynamic_descriptor);`
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`

Sample Force Post with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaResForcePostCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "uroyVNSxzjk5hHoT0kpQDBCw4";
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String crypt_type = "7";
String auth_code = "124424";
String descriptor = "my descriptor";
String processing_country_code = "CA";
boolean status_check = false;
ResForcePostCC resForcePostCC = new ResForcePostCC();
resForcePostCC.setOrderId(order_id);
resForcePostCC.setCustId(cust_id);
```

Sample Force Post with Vault

```
    resForcePostCC.setAmount(amount);
    resForcePostCC.setDataKey(data_key);
    resForcePostCC.setAuthCode(auth_code);
    resForcePostCC.setCryptType(crypt_type);
    resForcePostCC.setDynamicDescriptor(descriptor);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resForcePostCC);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("Masked Pan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

4.4.6 Card Verification with Vault – ResCardVerificationCC

Things to Consider:

- This transaction type only applies to Visa, Mastercard and Discover transactions
- The card number and expiry date for this transaction are passed using a token, as represented by the data key value
- When using a temporary token (e.g., such as with Hosted Tokenization) **and** you intend to store the cardholder credentials, this transaction must be run prior to running the Vault Add Token transaction

Card Verification with Vault object definition

```
ResCardVerificationCC resCardVerificationCC = new ResCardVerificationCC();
```

HttpsPostRequest object for Card Verification with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resCardVerificationCC);
```

Card Verification with Vault transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 4: Card Verification with Vault transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`resCardVerificationCC .setOrderId(order_id);`
Data key	String	25-character alpha-numeric	`resCardVerificationCC .setDataKeyFormat(data_key_ format);`
E-commerce indicator	String	1-character alpha-numeric	`resCardVerificationCC .setCryptType(crypt);`

Table 4: Card Verification with Vault transaction object mandatory values

Value	Type	Limits	Set method
AVS	Object	N/A	`resCardVerificationCC .setAvsInfo(avsCheck);`
CVD	Object	N/A	`resCardVerificationCC .setCvdInfo(cvdCheck);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	Object	N/A	`resCardVerificationCC .setCofInfo(cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Variable Name	Type and Limits	Set Method
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Card Verification with Vault

```
package Canada;
import java.io.*;
import JavaAPI.*;
public class TestCanadaResCardVerificationCC
{
public static void main(String args[]) throws IOException
{
String store_id = "store5";
String api_token = "yesguy";
String data_key = "AoG4zAFzlFFfxcVmzWAZVQuhj";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String crypt_type = "7";
String processing_country_code = "CA";
boolean status_check = false;

/********************** Efraud Variables ************************/
AvsInfo avs = new AvsInfo ();
avs.setAvsStreetName("test ave");
avs.setAvsStreetNumber("123");
avs.setAvsZipcode("123456");
CvdInfo cvd = new CvdInfo ("1", "099");
/*********************** Transaction Object *******************************/
ResCardVerificationCC resCardVerificationCC = new ResCardVerificationCC();
resCardVerificationCC.setDataKey(data_key);
resCardVerificationCC.setOrderId(order_id);
resCardVerificationCC.setCryptType(crypt_type);
resCardVerificationCC.setAvsInfo(avs);
resCardVerificationCC.setCvdInfo(cvd);
//resCardVerificationCC.setExpdate("1412"); //For Temp Tokens only

//Mandatory - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

resCardVerificationCC.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resCardVerificationCC);
```

Sample Card Verification with Vault

```
mpgReq.setStatusCheck(status_check);
mpgReq.send();
/************************ Receipt Object ******************************/
try
{
Receipt resreceipt = mpgReq.getReceipt();
System.out.println("DataKey = " + resreceipt.getDataKey());
System.out.println("ReceiptId = " + resreceipt.getReceiptId());
System.out.println("ReferenceNum = " + resreceipt.getReferenceNum());
System.out.println("ResponseCode = " + resreceipt.getResponseCode());
System.out.println("AuthCode = " + resreceipt.getAuthCode());
System.out.println("ISO = " + resreceipt.getISO());
System.out.println("Message = " + resreceipt.getMessage());
System.out.println("TransDate = " + resreceipt.getTransDate());
System.out.println("TransTime = " + resreceipt.getTransTime());
System.out.println("TransType = " + resreceipt.getTransType());
System.out.println("Complete = " + resreceipt.getComplete());
System.out.println("TransAmount = " + resreceipt.getTransAmount());
System.out.println("CardType = " + resreceipt.getCardType());
System.out.println("TxnNumber = " + resreceipt.getTxnNumber());
System.out.println("TimedOut = " + resreceipt.getTimedOut());
System.out.println("ResSuccess = " + resreceipt.getResSuccess());
System.out.println("PaymentType = " + resreceipt.getPaymentType() + "\n");
System.out.println("IssuerId = " + resreceipt.getIssuerId());
//Contents of ResolveData
System.out.println("Cust ID = " + resreceipt.getResCustId());
System.out.println("Phone = " + resreceipt.getResPhone());
System.out.println("Email = " + resreceipt.getResEmail());
System.out.println("Note = " + resreceipt.getResNote());
System.out.println("Masked Pan = " + resreceipt.getResMaskedPan());
System.out.println("Exp Date = " + resreceipt.getResExpdate());
System.out.println("Crypt Type = " + resreceipt.getResCryptType());
System.out.println("Avs Street Number = " + resreceipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + resreceipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + resreceipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
} // end TestResCardVerificationCC
```

4.5 Hosted Tokenization

Moneris Hosted Tokenization is a solution for online e-commerce merchants who do not want to handle credit card numbers directly on their websites, yet want the ability to fully customize their check-out web page appearance.

When an hosted tokenization transaction is initiated, the Moneris Gateway displays (on the merchant's behalf) a single text box on the merchant's checkout page. The cardholder can then securely enter the credit card information into the text box. Upon submission of the payment information on the checkout page, Moneris Gateway returns a temporary token representing the credit card number to the merchant. This is then used in an API call to process a financial transaction directly with Moneris to charge the card. After receiving a response to the financial transaction, the merchant generates a receipt and allows the cardholder to continue with online shopping.

For more details on how to implement the Moneris Hosted Tokenization feature, see the Hosted Solutions Integration Guide. The guide can be downloaded from the Moneris Developer Portal at

developer.moneris.com

5 INTERAC® Online Payment

5.1 About INTERAC® Online Payment Transactions

The INTERAC® Online Payment method offers cardholders the ability to pay using online banking. This payment method can be combined with the Moneris GatewayJava API solution to allow online payments using credit and debit cards.

INTERAC® Online Payment transactions via the Java API require two steps:

1. The cardholder guarantees the funds for the purchase amount using their online banking process.
2. The merchant confirms the payment by sending an INTERAC® Online Payment purchase request to Moneris using the Java API.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 18.5 Processing a Transaction.

INTERAC® Online Payment transactions are available to **Canadian integrations** only.

5.2 Other Documents and References

INTERAC® Online Payment is offered by Acxsys Corporation, which is also a licensed user of the *Interac* logo. Refer to the following documentation and websites for additional details.

INTERAC® Online PaymentMerchant Guideline

Visit the Moneris Developer Portal (https://developer.moneris.com) to access the latest documentation and downloads.

This details the requirements for each page consumers visit on a typical INTERAC® Online Payment merchant website. It also details the requirements that can be displayed on any page (that is, requirements that are not page-specific).

Logos

Visit the Moneris Developer Portal (https://developer.moneris.com) to access the logos and downloads.

5.3 Website and Certification Requirements

5.3.1 Things to provide to Moneris

Refer to the Merchant Guidelines referenced in Section 5.2 for instructions on proper use of logos and the term "INTERAC® Online Payment". You need to provide Moneris with the following registration information:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
  - In both French and English
  - 120 × 30 pixels
  - Only PNG format is supported.

- Merchant business name
  - In both English and French
  - Maximum 30 characters.

- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

Note that if your test and production environments are different, provide the above information for both environments.

5.3.2 Certification process

Test cases

All independent merchants and third-party service/shopping cart providers must pass the certification process by conducting all the test cases outlined in Appendix E (page 512) and "Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing" on page 516 respectively. This is required after you have completed all of your testing.

Any major changes to your website after certification (with respect to the INTERAC® Online Payment functionality) require the site to be re-certified by completing the test cases again.

Appendix H (page 524) is the Certification Test Case Detail showing all the information and requirements for each test case.

Screenshots

You must provide Moneris with screenshots of your check-out process showing examples of approved and declined transactions using the INTERAC® Online Payment service.

Checklists

To consistently portray the INTERAC Online service as a secure payment option, you must complete the respective Merchant Requirement checklist inAppendix E (page 512) or Appendix F (page 516)accordingly. The detailed descriptions of the requirements in these checklists can be found in the INTERAC® Online Payment Merchant Guidelines document referred to in 5.2 (page 110). If any item does not apply, mark it as "N/A".

After completion, fax or email the results to the Moneris Integration Support help desk for review before implementing the change into the production environment.

5.3.3 Client Requirements

Checklists

As a merchant using an INTERAC® Online Payment-certified third-party solution, your clients must complete the Merchant Checklists for INTERAC® Online Payment Certification form (Appendix G, page 521). They will **not** be required to complete any of the test cases.

Your clients must also complete the Merchant Requirement checklist (Appendix G, page 521). Ensure that your product documentation properly instructs your clients to fax or email the results to the Moneris Integration Support helpdesk for registration purposes.

Screenshots

Your clients must provide Moneris with screenshots of their check-out process that show examples of approved and declined transactions using INTERAC® Online Payment.

5.3.4 Delays

Note that merchants that fall under the following category codes listed in Table 5 may experience delays in the certification or registration process of up to 7 days.

Table 5: Category codes that might introduce certification/registration delays

Category code	Merchant type/name
4812	Telecommunication equipment including telephone sales
4829	Money transfer—merchant
5045	Computers, computer peripheral equipment, software
5732	Electronic sales
6012	Financial institution—merchandise and services
6051	Quasi cash—merchant

Category code	Merchant type/name
6530	Remote stored value load—merchant
6531	Payment service provider—money transfer for a purchase
6533	Payment service provider—merchant—payment transaction

5.4 Transaction Flow for INTERAC® Online Payment

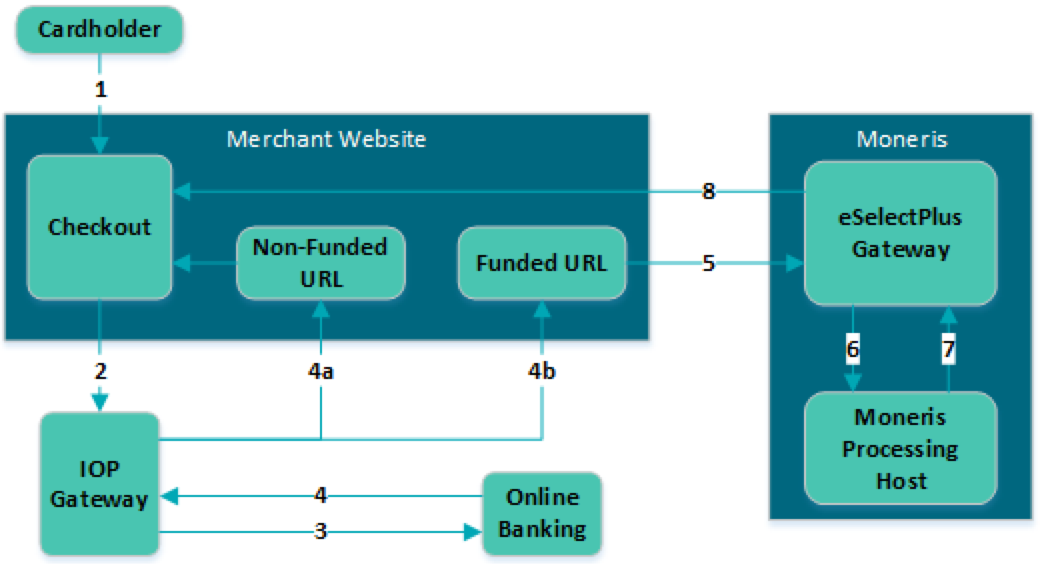


Figure 2: INTERAC® Online Payment transaction flow diagram

1. Customer selects the INTERAC® Online Payment option on the merchant's web store.
2. Merchant redirects the customer to the IOP gateway to select a financial institution (issuer) of choice. This step involves form-posting the following required variables over the HTTPS protocol:

   - IDEBIT_MERCHNUM
   - IDEBIT_AMOUNT[1]
   - IDEBIT_CURRENCY
   - IDEBIT_FUNDEDURL
   - IDEBIT_NOTFUNDEDURL
   - IDEBIT_MERCHLANG
   - IDEBIT_VERSIONIDEBIT_TERMID - optional
   - IDEBIT_INVOICE - optional
   - IDEBIT_MERCHDATA - optional

3. Customer selects an issuer, and is directed to the online banking site. Customer completes the online banking process and guarantees the funds for the purchase.

[1]This value is expressed in cents. Therefore, $1 is input as 100

4. Depending on the results of step 5.4, the issuer re-directs the customer through the IOP Gateway to either the merchant's non-funded URL (4a) or funded URL (4b). Both URLs can appear on the same page. The funded/non-funded URLs must validate the variables posted back according to 5.8 (page 120) before continuing.

   5.4 shows the variables that are posted back in the re-direction.

   If the customer is directed to the non-funded URL, return to step 5.4 and ask for another means of payment.

   If the customer is directed to the funded URL, continue to the next step.
5. Merchant sends an INTERAC® Online Payment purchase request to Moneris Gateway while displaying the "Please wait...." message to the customer. This should be done within 30 minutes of receiving the response in step 5.4.
6. Moneris' processing host sends a request for payment confirmation to the issuer.
7. The issuer sends a response (either approved or declined) to Moneris host.
8. Moneris Gateway relays the response back to the merchant. If the payment was approved, the merchant fulfills the order.

Table 6: Funded and non-funded URL variables

To funded URL only	To funded and non-funded URL
IDEBIT_TRACK2	IDEBIT_VERSION
IDEBIT_ISSCONF	IDEBIT_ISSLANG
IDEBIT_ISSNAME	IDEBIT_TERMID (optional)
	IDEBIT_INVOICE (optional)
	IDEBIT_MERCHDATA (optional)

5.5 Sending an INTERAC® Online Payment Purchase Transaction

5.5.1 Fund-Guarantee Request

After choosing to pay by INTERAC® Online Payment, the customer is redirected using an HTML form post to the INTERAC® Online PaymentGateway page. Below is a sample code that is used to post the request to the Gateway.

```
<form action='from Section 9' method='post'>
<input type='text' name='IDEBIT_INVOICE' value='your unique invoice number'>
    <input type='text' name='IDEBIT_AMOUNT' value='100'> <!— ($1.00) use cent values instead of
        dollar.cent format ->
<input type='text' name='IDEBIT_MERCHNUM' value='from Moneris Solutions'>
<input type='text' name='IDEBIT_CURRENCY' value='CA'>
<input type='text' name='IDEBIT_FUNDEDURL' value='your funded url'>
<input type='text' name='IDEBIT_NOTFUNDEDURL' value='your not funded url'>
<input type='text' name='IDEBIT_ISSLANG' value='en'>
<input type='text' name='IDEBIT_VERSION' value='1'>
<input type="submit" name="Submit" value="Submit to Gateway">
</form>
```

5.5.2 Online Banking Response and Fund-Confirmation Request

The response variables are posted back in an HTML form to either the funded or non-funded URL that was provided to INTERAC®.

The following variables must be validated (5.8, page 120):

- IDEBIT_TRACK2
- IDEBIT_ISSCONF
- IDEBIT_ISSNAME
- IDEBIT_VERSION
- IDEBIT_ISSLANG
- IDEBIT_INVOICE

Note that IDEBIT_ISSCONF and IDEBIT_ISSNAME must be displayed on the client's receipt that is generated by the merchant.

After validation, IDEBIT_TRACK2 is used to form an IDebitPurchase transaction that is sent to Moneris Gateway to confirm the fund.

If the validation fails, redirect the client to the main page and ask for a different means of payment.

If the validation passes, an IDebitPurchase transaction can be sent to Moneris Gateway.

5.6 INTERAC® Online Payment Purchase

INTERAC® Online Payment Purchase transaction object definition

```
IDebitPurchase IOP_Txn = new IDebitPurchase();
```

HttpsPostRequest object for INTERAC® Online Payment Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(IOP_Txn);
```

INTERAC® Online Payment Purchase transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 7: INTERAC® Online Payment transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`IOP_Txn.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`IOP_Txn.setAmount(amount);`
Track2 data	String	40-character alpha-numeric	`IOP_Txn.setTrack2(track2);`

Table 8: INTERAC® Online Payment Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`IOP_Txn.setCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric	`IOP_Txn.setDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	Not applicable. Click hereSee Section 15 (page 407).	`IOP_Txn.setCustInfo(customer);`

Sample INTERAC® Online Payment Purchase

```
package Canada;
import JavaAPI.*;
public class TestCanadaIDebitPurchase
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "Lance_Briggs_55";
String amount = "5.00";
String track2 = "5268051119993326=0609AAAAAAAAAAAAA000";
String processing_country_code = "CA";
boolean status_check = false;
/******************** Billing/Shipping Variables ***************************/
String first_name = "Bob";
```

Sample INTERAC® Online Payment Purchase

```
String last_name = "Smith";
String company_name = "ProLine Inc.";
String address = "623 Bears Ave";
String city = "Chicago";
String province = "Illinois";
String postal_code = "M1M2M1";
String country = "Canada";
String phone = "777-999-7777";
String fax = "777-999-7778";
String tax1 = "10.00";
String tax2 = "5.78";
String tax3 = "4.56";
String shipping_cost = "10.00";
/********************** Order Line Item Variables *****************************/
String[] item_description = new String[] { "Chicago Bears Helmet", "Soldier Field Poster" };
String[] item_quantity = new String[] { "1", "1" };
String[] item_product_code = new String[] { "CB3450", "SF998S" };
String[] item_extended_amount = new String[] { "150.00", "19.79" };
/*********************** Customer Information Object **************************/
CustInfo customer = new CustInfo();
/*********************** Set Customer Billing Information **********************/
customer.setBilling(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/******************** Set Customer Shipping Information ***********************/
customer.setShipping(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/***************************** Order Line Items ******************************/
customer.setItem(item_description[0], item_quantity[0],
item_product_code[0], item_extended_amount[0]);
customer.setItem(item_description[1], item_quantity[1],
item_product_code[1], item_extended_amount[1]);
/*************************** Request *************************/
IDebitPurchase IOP_Txn = new IDebitPurchase();
IOP_Txn.setOrderId(order_id);
IOP_Txn.setCustId(cust_id);
IOP_Txn.setAmount(amount);
IOP_Txn.setIdebitTrack2(track2);
IOP_Txn.setCustInfo(customer);
//IOP_Txn.setDynamicDescriptor("dynamicdescriptor1");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(IOP_Txn);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
```

Sample INTERAC® Online Payment Purchase

```
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

5.7 INTERAC® Online Payment Refund

To process this transaction, you need the order ID and transaction number from the original INTERAC® Online Payment Purchase transaction.

INTERAC® Online Payment Refund transaction object definition

```
IDebitRefund refund = new IDebitRefund();
```

HttpsPostRequest object for INTERAC® Online Payment Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(refund);
```

INTERAC® Online Payment Refund transaction object values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 9: INTERAC® Online Payment Refund transaction object mandatory variables

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`refund.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`refund.setAmount(amount);`
Transaction number	String	255-character alpha-numeric	`refund.setTxnNumber(txn_number);`

Table 10: INTERAC® Online Payment Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`refund.setCustId(cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample code

Sample INTERAC® Online Payment Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaIDebitRefund
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test1435508096214";
String amount = "5.00";
String txn_number = "116181-0_10";
String processing_country_code = "CA";
String cust_id = "my customer id";
boolean status_check = false;
IDebitRefund refund = new IDebitRefund();
refund.setOrderId(order_id);
```

Sample INTERAC® Online Payment Refund

```
refund.setAmount(amount);
refund.setTxnNumber(txn_number);
refund.setCustId(cust_id);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(refund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

5.8 INTERAC® Online Payment Field Definitions

Table 11: Field Definitions

Value	Characters	Limits
	Description	
IDEBIT_ MERCHNUM	5-14	Numbers and uppercase letters
	This field is provided by Moneris. For example, 0003MONMPGXXXX.	
IDEBIT_TERMID	8	Numbers and uppercase letters
	Optional field	

Table 11: Field Definitions (continued)

Value	Characters	Limits
	Description	
IDEBIT_ AMOUNT	1-12	Numbers
	Amount expressed in cents (for example, 1245 for $12.45) to charge to the card.	
IDEBIT_ CURRENCY	3	"CAD" or "USD"
	National currency of the transaction.	
IDEBIT_INVOICE	1-20	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ '
	Optional field Can be the Order ID when used with Moneris Gateway fund confirmation transactions.	
IDEBIT_ MERCHDATA	1024	ISO-8859-1 restricted to single-byte codes, hex 20 to 7E (consistent with US-ASCII and ISO-8859-1 Latin-1). Note that the following character combinations may not be accepted in the IDEBIT_MERCHDATA field: • "/..", "/%2E.", "/.%2E", "/%2E%2E", "\\%2E%2E", "\\%2E.", "\\.%2E", "\\%2E%2E", "&#", "<", "%3C", ">", "%3E"
	Free form data provided by the merchant that will be passed back unchanged to the merchant once the payment has been guaranteed in online banking. This may be used to identify the customer, session or both.	
IDEBIT_ FUNDEDURL	1024	ISO-8859-1 restricted to single-byte codes, restricted to: • Uppercase and lowercase letters • Numbers • ; / ? : @ & = + $, - _ . ! ~ * ‘ ( ) %
	Https address to which the issuer will redirect cardholders after guaranteeing the fund through online banking.	

Table 11: Field Definitions (continued)

Value	Characters	Limits
	Description	
IDEBIT_ NOTFUNDEDURL	1024	ISO-8859-1, restricted to single-byte codes, restricted to: • Uppercase and lowercase letters • Numbers • ; / ? : @ & = + $, - _ . ! ~ * ‘ ( ) %
	Https address to which the issuer redirects cardholders after failing or canceling the online banking process.	
IDEBIT_ MERCHLANG	2	“en” or “fr”
	Customer's current language at merchant.	
IDEBIT_VERSION	3	Numbers
	Initially, the value is 1.	
IDEBIT_ISSLANG	2	“en” or “fr”
	Customer’s current language at issuer.	
IDEBIT_TRACK2	37	ISO-8859-1 (restricted to single-byte codes), hex 20 to 7E (consistent with US-ASCII and ISO-8859-1 Latin-1)
	Value returned by the issuer. It includes the PAN, expiry date, and transaction ID.	
IDEBIT_ISSCONF	15	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase letters • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ '
	Confirmation number returned from the issuer to be displayed on the merchant’s confirmation page and on the receipt.	
IDEBIT_ ISSNAME	30	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase letters • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ • '
	Issuer name to be displayed on the merchant’s confirmation page and on the receipt.	

6 Mag Swipe Transaction Set

Mag Swipe transactions allow customers to swipe a credit card and submit the Track2 details.

These transactions support the submission of Track2 as well as a manual entry of the credit card number and expiry date. If all three fields are submitted, the Track2 details are used to process the transaction.

6.1 Mag Swipe Transaction Type Definitions

Purchase

Verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

Pre-Authorization

Verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant's account, a Completion transaction must be performed. A Pre-Authorization may only be "completed" once.

Completion

Retrieves funds that have been locked (by a Mag Swipe Pre-Authorization transaction), and prepares them for settlement into the merchant's account.

Force Post

Retrieves the locked funds and prepares them for settlement into the merchant's account.

This is used when a merchant obtains the authorization number directly from the issuer by a third-party authorization method (such as by phone).

Purchase Correction

Restores the **full** amount of a previous Mag Swipe Purchase or Mag Swipe Completion transaction to the cardholder's card, and removes any record of it from the cardholder's statement. The order ID and transaction number from the original transaction are required, but the credit card does not need to be re-swiped.

This transaction can be used against a Purchase or Completion transaction that occurred same day provided that the batch containing the original transaction remains open. When using the automated closing feature, Batch Close occurs daily between 10 and 11 pm Eastern Time.

This transaction is sometimes referred to as "void".

Refund

Restores all or part of the funds from a Mag Swipe Purchase or Mag Swipe Completion transaction to the cardholder's card. Unlike a Purchase Correction, there is a record of the refund.

Independent Refund

Credits a specified amount to the cardholder’s credit card.

This does not require a previous transaction (such as Mag Swipe Purchase) to be logged in the Moneris Gateway. However, a credit card must be swiped to provide the Track2 data.

6.1.1 Encrypted Mag Swipe Transactions

Encrypted Mag Swipe transactions allow the customer to swipe or key in a credit card using a Moneris-provided encrypted mag swipe reader, and submit the encrypted Track2 details.

The encrypted mag swipe reader can be used for processing:

- Swiped card-present transactions
- Manually keyed card-present transactions
- Manually keyed card-not-present transactions.

Encrypted Mag Swipe transactions are identical to the regular Mag Swipe transactions from the customer's perspective. However, the card data must be swiped or keyed in via a Moneris-provided encrypted mag swipe reader. Contact Moneris for more details.

Only Mag Swipe Purchase and Mag Swipe Pre-Authorization have encrypted versions. Their explanations appear in this document as subsections of the regular (unencrypted) Mag Swipe Purchase and Mag Swipe Pre-Authorization transactions respectively.

6.2 Mag Swipe Purchase

Mag Swipe Purchase transaction object definition

```
Track2Purchase track2purchase = new Track2Purchase();
```

HttpsPostRequest object for Mag Swipe Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2purchase);
```

Mag Swipe Purchase transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 12: Mag Swipe Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2purchase.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2purchase.setAmount (amount);`

Credit card number
 	
OR
 	
Track2 data
	String	20-character numeric
		
		OR
		
		40-character numeric
			`track2purchase.setPan(pan);`
			
			OR
			
			`track2purchase.setTrack2 (track2);`

Expiry date	String	4-character alpha-numeric (YYMM format)	`track2purchase.setExpDate (expiry_date);`
POS code	String	2-character numeric	`track2purchase.setPosCode (pos_code);`

Table 13: Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
AVS information	Object	N/A	`track2purchase.setAvsInfo (avsCheck);`
Customer ID	String	50-character alpha-numeric	`track2purchase.setCustId (cust_id);`
CVD information	Object	N/A	`track2purchase.setCvdInfo (cvdCheck);`

Table 13: Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
Dynamic descriptor	String	20-character alpha-numeric	`track2purchase .setDynamicDescriptor (dynamic_descriptor);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample Mag Swipe Purchase

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2Purchase
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "LBriggs";
String amount = "1.00";
String track2 = ";5258968987035454=06061015454001060101?";
String pan = "";
String exp = ""; //must send '0000' if swiped
String pos_code = "00";
String processing_country_code = "CA";
boolean status_check = false;
Track2Purchase track2purchase = new Track2Purchase();
track2purchase.setOrderId(order_id);
track2purchase.setCustId(cust_id);
track2purchase.setAmount(amount);
track2purchase.setTrack2(track2);
track2purchase.setPan(pan);
track2purchase.setExpdate(exp);
track2purchase.setPosCode(pos_code);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2purchase);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("BankTotals = " + receipt.getBankTotals());
```

Sample Mag Swipe Purchase

```
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.2.1 Encrypted Mag Swipe Purchase

Encrypted Mag Swipe Purchase transaction object definition

```
EncTrack2Purchase encpurchase = new EncTrack2Purchase();
```

HttpsPostRequest object for Encrypted Mag Swipe Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(encpurchase);
```

Encrypted Mag Swipe Purchase transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 14: Encrypted Mag Swipe Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`encpurchase.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`encpurchase.setAmount(amount);`
Encrypted Track2 data	String	n/a	`encpurchase.setEncTrack2(enc_track2);`
POS code	String	2-character numeric	`encpurchase.setPosCode(pos_code);`
Device type	String	30-character alpha-numeric	`encpurchase.setDeviceType(device_type);`

Table 15: Encrypted Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`encpurchase.setCustId(cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
AVS information	Object	n/a	`encpurchase.setAvsInfo(avsCheck);`
Dynamic descriptor	String	20-character alpha-numeric	`encpurchase.setDynamicDescriptor(dynamic_descriptor);`

Sample Encrypted Mag Swipe Purchase

```
package Canada;
import JavaAPI.*;
```

Sample Encrypted Mag Swipe Purchase

```
public class TestCanadaEncTrack2Purchase
{
public static void main(String args[])
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "moneris";
String api_token = "hurgle";
String amount = "1.00";
String enc_track2 = "ENCRYPTEDTRACK2DATA";
String pan = "";
String expdate = "";
String pos_code = "00";
String device_type = "idtech_bdk";
String processing_country_code = "CA";
EncTrack2Preauth enc_track2_preauth = new EncTrack2Preauth ();
enc_track2_preauth.setOrderId(order_id);
enc_track2_preauth.setAmount(amount);
enc_track2_preauth.setEncTrack2(enc_track2);
enc_track2_preauth.setPan(pan);
enc_track2_preauth.setExpdate(expdate);
enc_track2_preauth.setPosCode(pos_code);
enc_track2_preauth.setDeviceType(device_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(enc_track2_preauth);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.3 Mag Swipe Pre-Authorization

Mag Swipe Pre-Authorization transaction object definition

```
Track2PreAuth track2preauth = new Track2PreAuth();
```

HttpsPostRequest object for Mag Swipe Pre-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2preauth);
```

Mag Swipe Pre-Authorization transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 16: Mag Swipe Pre-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2preauth.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2preauth.setAmount (amount);`
Credit card number OR Track2 data	String	20-character numeric OR 40-character numeric	`track2preauth.setPan(pan);` OR `track2preauth.setPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`track2preauth.setExpDate (expiry_date);`
POS code	String	2-character numeric	`track2preauth.setPosCode(pos_code);`

Table 17: Mag Swipe Pre-Authorization transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2preauth.setCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`track2preauth .setDynamicDescriptor (dynamic_descriptor);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample Mag Swipe Pre-Authorization

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2PreAuth
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "LBriggs";
String amount = "5.00";
String track2 = ";5258968987035454=06061015454001060101?";
String pan = "";
String exp = "0000"; //must send '0000' if swiped
String pos_code = "00";
String processing_country_code = "CA";
boolean status_check = false;
Track2PreAuth track2preauth = new Track2PreAuth();
track2preauth.setOrderId(order_id);
track2preauth.setCustId(cust_id);
track2preauth.setAmount(amount);
track2preauth.setTrack2(track2);
track2preauth.setPan(pan);
track2preauth.setExpdate(exp);
track2preauth.setPosCode(pos_code);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2preauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
```

Sample Mag Swipe Pre-Authorization

```
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.3.1 Encrypted Mag Swipe Pre-Authorization

Encrypted Mag Swipe Pre-Authorization transaction object definition

`EncTrack2Preauth enc_track2_preauth = new EncTrack2Preauth ();`

HttpsPostRequest object for Encrypted Mag Swipe Pre-Authorization transaction

`HttpsPostRequest mpgReq = new HttpsPostRequest();`

`mpgReq.setTransaction(enc_track2_preauth);`

Encrypted Mag Swipe Pre-Authorization transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 18: Encrypted Mag Swipe Pre-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`enc_track2_preauth.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`enc_track2_preauth.setAmount (amount);`

Credit card number
 	
OR
 	
Encrypted Track2
	String	20-character numeric
		
		OR
		
		n/a
			`enc_track2_preauth.setPan (pan);`
			
			OR
			
			`enc_track2_preauth.setEncTrack2(enc_track2);`

POS code	String	2-character numeric	`enc_track2_preauth.setPosCode (pos_code);`
Device type	String	30-character alpha-numeric	`enc_track2_preauth.setDeviceType(device_type);`

Table 19: Encrypted Mag Swipe Pre-Authorization transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`enc_track2_preauth.setCustId (cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample Encrypted Mag Swipe Pre-Authorization

```
    package Canada;
    import JavaAPI.*;
    public class TestCanadaEncTrack2Preauth
    {
```

Sample Encrypted Mag Swipe Pre-Authorization

```
public static void main(String args[])
{
String host = "esqa.moneris.com";
String store_id = "store1";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "1.00";
String enc_track2 ="ENCRYPTEDTRACK2DATA";
String pan = "";
String expdate = "";
String pos_code = "00";
String device_type = "idtech_bdk";
String processing_country_code = "CA";
EncTrack2Preauth enc_track2_preauth = new EncTrack2Preauth ();
enc_track2_preauth.setOrderId(order_id);
enc_track2_preauth.setAmount(amount);
enc_track2_preauth.setEncTrack2(enc_track2);
enc_track2_preauth.setPan(pan);
enc_track2_preauth.setExpdate(expdate);
enc_track2_preauth.setPosCode(pos_code);
enc_track2_preauth.setDeviceType(device_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(enc_track2_preauth);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
receipt = null;
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.4 Mag Swipe Completion

Mag Swipe Completion transaction object definition

```
Track2Completion track2completion = new Track2Completion();
```

HttpsPostRequest object for Mag Swipe Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2completion);
```

Mag Swipe Completion transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 20: Mag Swipe Completion transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2completion.setOrderId (order_id);`
Transaction number	String	255-character variable character	`track2completion.setTxnNumber (txn_number);`
Completion Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2completion.setCompAmount (comp_amount);`
POS code	String	2-character numeric	`track2completion.setPosCode (pos_code);`

Table 21: Mag Swipe Completion transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2completion.setCustId (cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-	`track2completion .setDynamicDescriptor`

Table 21: Mag Swipe Completion transaction optional values (continued)

Value	Type	Limits	Set method
		numeric	(dynamic_descriptor);

Sample Mag Swipe Completion

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2Completion
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String order_id = "Test1432091015817";
String txn_number = "16540-0_10";
String amount = "1.00";
String pos_code = "00";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
boolean status_check = false;
Track2Completion track2completion = new Track2Completion();
track2completion.setOrderId(order_id);
track2completion.setTxnNumber(txn_number);
track2completion.setAmount(amount);
track2completion.setPosCode(pos_code);
track2completion.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2completion);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
```

Sample Mag Swipe Completion

```
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

6.5 Mag Swipe Force Post

Mag Swipe Force Post transaction object definition

`Track2ForcePost track2forcePost = new Track2ForcePost();`

HttpsPostRequest object for Mag Swipe Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2forcePost);
```

Mag Swipe Force Post transaction mandatory arguments

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 22: Mag Swipe Force Post transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2forcePost.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2forcePost.setAmount (amount);`

Credit card number
 	
OR
 	
Track2 data
	String	20-character numeric
		
		OR
		
		40-character numeric
			`track2forcePost.setPan(pan);`
			
			OR
			
			`track2forcePost.setTrack2 (track2);`

Expiry date	String	4-character alpha-numeric	`track2forcePost.setExpDate (expiry_date);`

Table 22: Mag Swipe Force Post transaction object mandatory values

Value	Type	Limits	Set method
		(YYMM format)	
POS code	String	2-character numeric	`track2forcePost.setPosCode (pos_code);`
Authorization code	String	8-character alpha-numeric	`track2forcePost.setAuthCode (auth_code);`

Table 23: Mag Swipe Force Post transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2forcePost.setCustId (cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-numeric	`track2forcePost .setDynamicDescriptor (dynamic_descriptor);`

Sample Mag Swipe Force Post

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2ForcePost
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "1.00";
String track2 = ";5258968987035454=06061015454001060101?";
String auth_code = "123456";
String processing_country_code = "CA";
boolean status_check = false;
Track2ForcePost track2forcePost = new Track2ForcePost();
track2forcePost.setOrderId(order_id);
track2forcePost.setAmount(amount);
track2forcePost.setTrack2(track2);
track2forcePost.setAuthCode(auth_code);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
```

Sample Mag Swipe Force Post

```
    mpgReq.setTransaction(track2forcePost);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    //System.out.println("StatusCode = " + receipt.getStatusCode());
    //System.out.println("StatusMessage = " + receipt.getStatusMessage());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

6.5.1 Encrypted Mag Swipe Force Post

The Encrypted Mag Swipe Force Post is used when a merchant obtains the authorization number directly from the issuer using a phone or any third-party authorization method. This transaction does not require that an existing order be logged in the Moneris Gateway. However, the credit card must be swiped or keyed in using a Moneris-provided encrypted mag swipe reader, and the encrypted Track2 details must be submitted. There are also optional fields that may be submitted such as `cust_id` and `dynamic_descriptor`.

To complete the transaction, the authorization number obtained from the issuer must be entered.

Encrypted Mag Swipe Force Post transaction object definition

```
EncTrack2Forcepost enctrack2fp = new EncTrack2Forcepost();
```

HttpsPostRequest object for Encrypted Mag Swipe Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(enctrack2fp);
```

Encrypted Mag Swipe Force Post transaction object values

Table 1 Encrypted Mag Swipe Force Post transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`enctrack2fp.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`enctrack2fp.setAmount(amount);`
Encrypted Track2 data	String	n/a	`enctrack2fp.setEncTrack2(enc_track2);`
POS Code	String	2-character numeric	`enctrack2fp.setPosCode(pos_code);`
Device type	String	30-character alpha-numeric	`enctrack2fp.setDeviceType(device_type);`
Authorization Code	String	8-character alpha-numeric	`enctrack2fp.setAuthCode(auth_code);`

Table 2 Encrypted Mag Swipe Force Post transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Customer ID	String	50-character alpha-numeric	`enctrack2fp.setCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`enctrack2fp.setDynamicDescriptor(dynamic_descriptor);`

Sample Encrypted Mag Swipe Force Post

```
package Canada;
import JavaAPI.*;
public class TestCanadaEncTrack2Forcepost
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "my customer id";
String amount = "5.00";
String pos_code = "00";
String device_type = "idtech_bdk";
String auth_code = "123456";
String processing_country_code = "CA";
boolean status_check = false;
String descriptor = "my descriptor";
String enc_track2 = "ENCRYPTEDTRACK2DATA";
EncTrack2Forcepost enctrack2fp = new EncTrack2Forcepost();
enctrack2fp.setOrderId(order_id);
enctrack2fp.setCustId(cust_id);
enctrack2fp.setAmount(amount);
enctrack2fp.setEncTrack2(enc_track2);
enctrack2fp.setPosCode(pos_code);
enctrack2fp.setDeviceType(device_type);
enctrack2fp.setAuthCode(auth_code);
enctrack2fp.setDynamicDescriptor(descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(enctrack2fp);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("MaskedPan = " + receipt.getMaskedPan());
System.out.println("CardLevelResult = " + receipt.getCardLevelResult());
}
catch (Exception e)
```

Sample Encrypted Mag Swipe Force Post

```
    {
    e.printStackTrace();
    }
    }
    }
```

6.6 Mag Swipe Purchase Correction

Mag Swipe Purchase Correction transaction object definition

```
Track2PurchaseCorrection track2purchasecorrection = new
Track2PurchaseCorrection();
```

HttpsPostRequest object for Mag Swipe Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2purchasecorrection);
```

Mag Swipe Purchase Correction transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 24: Mag Swipe Purchase Correction transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2void.setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`track2void.setTxnNumber(txn_number);`

Table 25: Mag Swipe Purchase Correction transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2void.setCustId(cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample Mag Swipe Purchase Correction

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2PurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String order_id = "Test1432090631783";
String txn_number = "16522-0_10";
String cust_id = "my customer id";
String processing_country_code = "CA";
boolean status_check = false;
Track2PurchaseCorrection track2void = new Track2PurchaseCorrection();
track2void.setOrderId(order_id);
track2void.setCustId(cust_id);
track2void.setTxnNumber(txn_number);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2void);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.7 Mag Swipe Refund

Mag Swipe Refund transaction object definition

```
Track2Refund track2refund = new Track2Refund();
```

HttpsPostRequest object for Mag Swipe Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2refund);
```

Mag Swipe Refund transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 26: Mag Swipe Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2refund.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2refund.setAmount (amount);`
Transaction number	String	255-character alpha-numeric	`track2refund.setTxnNumber (txn_number);`

Table 27: Mag Swipe Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2refund.setCustId(cust_id);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-numeric	`track2refund .setDynamicDescriptor (dynamic_descriptor);`

Sample Mag Swipe Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2Refund
```

Sample Mag Swipe Refund

```
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String order_id = "Test1432090722923"; //will prompt user for input
String txn_number = "16524-0_10";
String amount = "1.00";
String dynamic_descriptor = "123456";
String cust_id = "customer id";
String processing_country_code = "CA";
boolean status_check = false;
Track2Refund track2refund = new Track2Refund();
track2refund.setOrderId(order_id);
track2refund.setAmount(amount);
track2refund.setCustId(cust_id);
track2refund.setTxnNumber(txn_number);
track2refund.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2refund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

6.8 Mag Swipe Independent Refund

NOTE: If you receive a TRANSACTION NOT ALLOWED error, it may mean the Mag Swipe Independent Refund transaction is not supported on your account. Contact Moneris to have it temporarily (re-)enabled.

Mag Swipe Independent Refund transaction object definition

```
Track2IndependentRefund track2indrefund = new Track2IndependentRefund();
```

HttpsPostRequest object for Mag Swipe Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(track2indrefund);
```

Mag Swipe Independent Refund transaction values

Table 28: Mag Swipe Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2indrefund.setOrderId (order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`track2indrefund.setAmount (amount);`
Credit card number	String	20-character numeric	`track2indrefund.setPan(pan);`
Track2 data	String	40-character numeric	`track2indrefund.setTrack2 (track2);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`track2indrefund.setExpDate (expiry_date);`
POS code	String	2-character numeric	`track2indrefund.setPosCode (pos_code);`

Table 29: Mag Swipe Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2indrefund.setCustId (cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`track2indrefund .setDynamicDescriptor(dynamic_ descriptor);`
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_ check);`

Sample Mag Swipe Independent Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaTrack2IndependentRefund
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "Ced_Benson32";
String amount = "5.00";
String track2 = ";5258968987035454=06061015454001060101?";
String pan = "";
String exp_date = "0000";
String pos_code = "00";
String processing_country_code = "CA";
String dynamic_descriptor = "my descriptor";
boolean status_check = false;
Track2IndependentRefund track2indrefund = new Track2IndependentRefund();
track2indrefund.setOrderId(order_id);
track2indrefund.setCustId(cust_id);
track2indrefund.setAmount(amount);
track2indrefund.setTrack2(track2);
track2indrefund.setPan(pan);
track2indrefund.setExpdate(exp_date);
track2indrefund.setPosCode(pos_code);
track2indrefund.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(track2indrefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
```

Sample Mag Swipe Independent Refund

```
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    //System.out.println("StatusCode = " + receipt.getStatusCode());
    //System.out.println("StatusMessage = " + receipt.getStatusMessage());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

6.8.1 Encrypted Mag Swipe Independent Refund

The Encrypted Mag Swipe Independent Refund credits a specified amount to the cardholder's credit card. The Encrypted Mag Swipe Independent Refund does not require an existing order to be logged in the Moneris Gateway. However, the credit card must be swiped using the Moneris-provided encrypted mag swipe reader to provide the encrypted track2 details.

There are also optional fields that may be submitted such as `cust_id` and `dynamic_descriptor`. The transaction format is almost identical to Encrypted Mag Swipe Purchase and Encrypted Mag Swipe PreAuth.

NOTE:
The Encrypted Mag Swipe Independent Refund transaction may not be supported on your account. This may yield a TRANSACTION NOT ALLOWED error when attempting the transaction.

To temporarily enable (or re-enable) the Independent Refund transaction type, contact Moneris

Encrypted Mag Swipe Independent Refund transaction object definition

```
EncTrack2IndependentRefund encindrefund = new EncTrack2IndependentRefund();
```

HttpsPostRequest object for Encrypted Mag Swipe Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(encindrefund);
```

Encrypted Mag Swipe Independent Refund transaction object values

Table 1 Encrypted Mag Swipe Independent Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`encindrefund.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`encindrefund.setAmount(amount);`
Encrypted Track 2 data	String	n/a	`encindrefund.setEncTrack2(enc_track2);`
Device Type	String	30-character alpha-numeric	`encindrefund.setDeviceType(device_type);`
POS Code	String	2-character numeric	`encindrefund.setPosCode(pos_code);`

Table 2 Encrypted Mag Swipe Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Customer ID	String	50-character alpha-numeric	`encindrefund.setCustId(cust_id);`

Sample Encrypted Mag Swipe Independent Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaEncTrack2IndependentRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
```

Sample Encrypted Mag Swipe Independent Refund

```
    java.util.Date createDate = new java.util.Date();
    String order_id = "Test"+createDate.getTime();
    String cust_id = "my customer id";
    String amount = "5.00";
    String pos_code = "00";
    String device_type = "idtech_bdk";
    String processing_country_code = "CA";
    String enc_track2 = "ENCRYPTEDTRACK2DATA";
    EncTrack2IndependentRefund encindrefund = new EncTrack2IndependentRefund();
    encindrefund.setOrderId(order_id);
    encindrefund.setCustId(cust_id);
    encindrefund.setAmount(amount);
    encindrefund.setEncTrack2(enc_track2);
    encindrefund.setPosCode(pos_code);
    encindrefund.setDeviceType(device_type);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(encindrefund);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("MaskedPan = " + receipt.getMaskedPan());
    System.out.println("CardLevelResult = " + receipt.getCardLevelResult());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

7 Level 2/3 Transactions

7.1 About Level 2/3 Transactions

The Moneris Gateway API supports passing Level 2/3 purchasing card transaction data for Visa, MasterCard and American Express corporate cards.

All Level 2/3 transactions use the same Pre-Authorization transaction as described in the topic Pre-Authorization (page 18).

7.2 Level 2/3 Visa Transactions

7.2.1 Level 2/3 Transaction Types for Visa

This transaction set includes a suite of corporate card financial transactions as well as a transaction that allows for the passing of Level 2/3 data. Please ensure that Visa Level 2/3 support is enabled on your merchant account. Batch Close, Open Totals and Pre-authorization are identical to the transactions outlined in the section Basic Transaction Set (page 12).

- When the Pre-authorization response contains CorporateCard equal to true then you can submit the Visa transactions.
- If CorporateCard is false then the card does not support Level 2/3 data and non Level 2/3 transaction are to be used. If the card is not a corporate card, please refer to the section 2 Basic Transaction Set for the appropriate non-corporate card transactions.

NOTE: This transaction set is intended for transactions where Corporate Card is true and Level 2/3 data will be submitted. If the credit card is found to be a corporate card but you do not wish to send any Level 2/3 data then you may submit Visa transactions using the basic transaction set outlined in 2 Basic Transaction Set.

Pre-authorization– (authorization/pre-authorization)
Pre-authorization verifies and locks funds on the customer’s credit card. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from a preauth so that they may be settled in the merchant account a capture must be performed. CorporateCard will return as true if the card supports Level 2/3.

VS Completion – (Capture/Pre-authorization Completion)
Once a Pre-authorization is obtained the funds that are locked need to be retrieved from the customer’s credit card. The capture retrieves the locked funds and readies them for settlement into the merchant account. Prior to performing a VS Completion, a Pre-authorization must be performed. Once the transaction is completed, VS Corpais must be used to process the Level 2/3 data.

VS Force Post – (Force Capture/Pre-authorization Completion)
This transaction is an alternative to VS Completion to obtain the funds locked on Pre-auth obtained from IVR or equivalent terminal. The VS Force Post retrieves the locked funds and readies them for settlement in to the merchant account. Once the transaction is completed, VS Corpais must be used to process the Level 2/3 data.

VS Purchase Correction (Void, Correction)
VS Completion and VS Force Post can be voided the same day* that they occur. A VS Purchase Correction must be for the full amount of the transaction and will remove any record of it from the cardholder statement.

VS Refund – (Credit)
A VS Refund can be performed against a VS Completion to refund any part or all of the transaction. Once the transaction is completed, VS Corpais must be used to process the Level 2/3 data.

VS Independent Refund – (Credit)
A VS Independent Refund can be performed against a purchase or a capture to refund any part, or all of the transaction. Independent refund is used when the originating transaction was not performed through Moneris Gateway. Once the transaction is completed, VS Corpais must be used to process the Level 2/3 data.

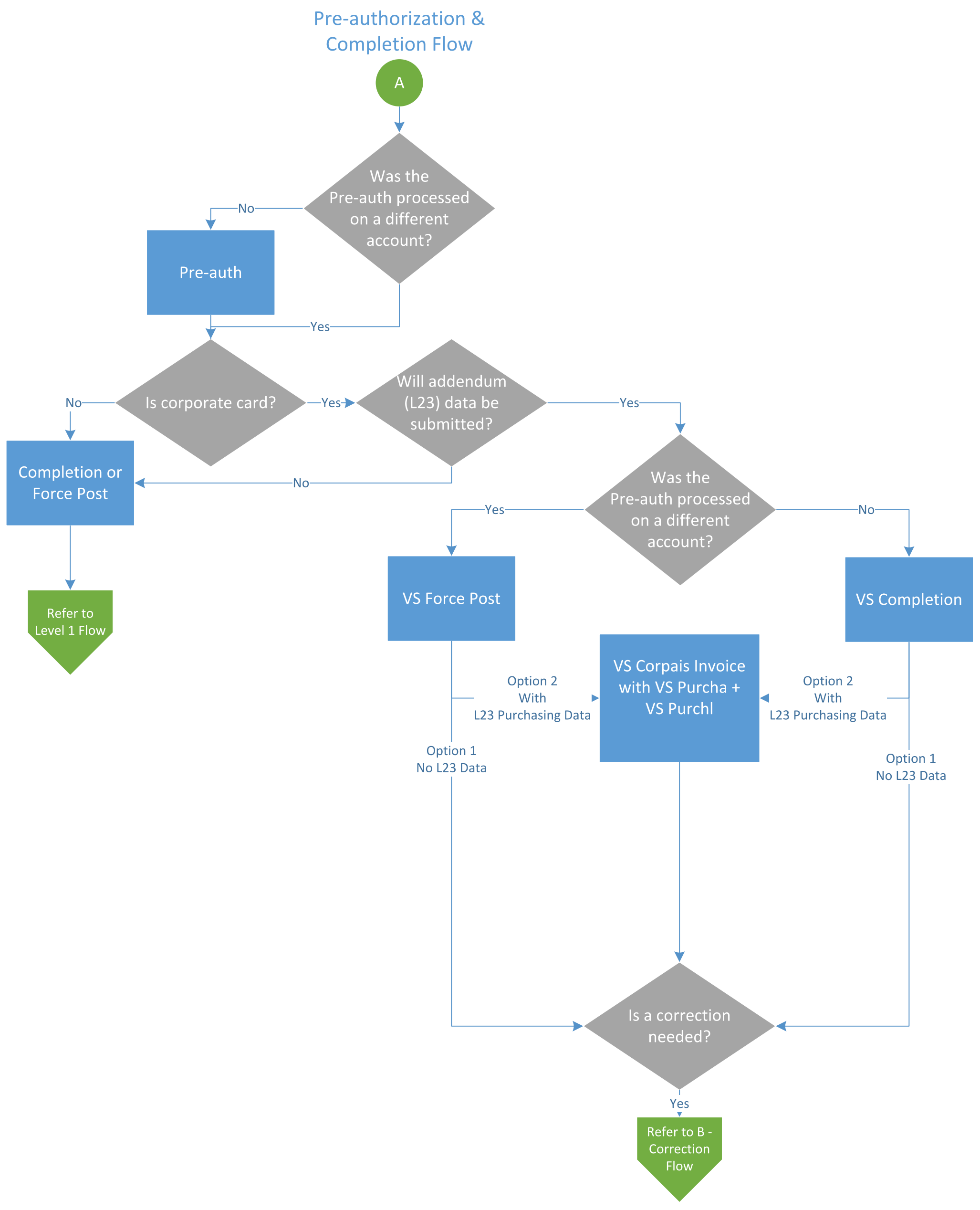
NOTE: the Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an independent refund, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-866-319-7450.

VS Corpais – (Level 2/3 Data)
VS Corpais will contain all the required and optional data fields for Level 2/3 Business to Business data. VS Corpais data can be sent when the card has been identified in the Pre-authorization transaction request as being a corporate card.

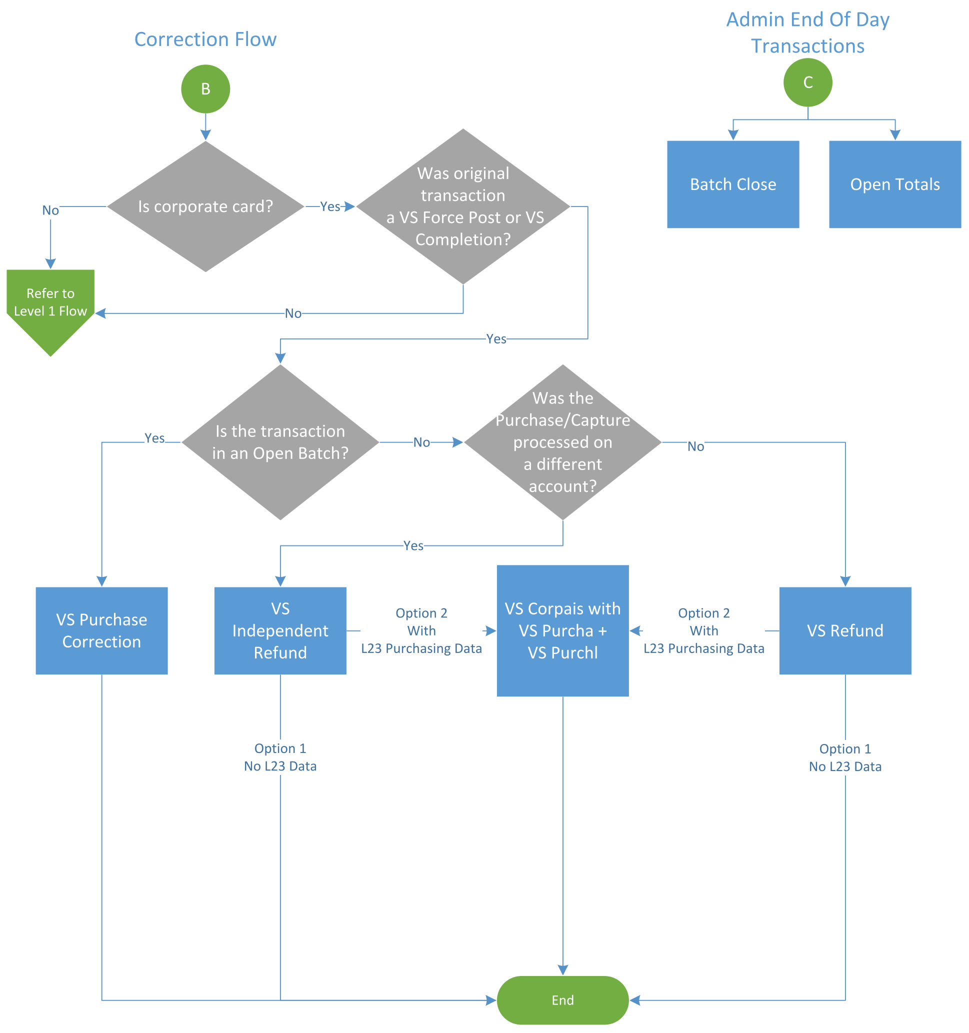
* A VS Purchase Correction can be performed against a transaction as long as the batch that contains the original transaction remains open. When using the automated closing feature, the batch close occurs daily between 10 – 11 pm EST.

7.2.2 Level 2/3 Transaction Flow for Visa

Pre-authorization/Completion Transaction Flow


Pre-authorization & Completion Flow
A
Was the Pre-auth processed on a different account?
No
Pre-auth
Yes
Is corporate card?
No
Yes
Will addendum (L23) data be submitted?
Yes
No
Completion or Force Post
Refer to Level 1 Flow
Was the Pre-auth processed on a different account?
Yes
No
VS Force Post
VS Completion
VS Corpais Invoice with VS Purcha + VS Purchl
Option 2 With L23 Purchasing Data
Option 2 With L23 Purchasing Data
Option 1 No L23 Data
Option 1 No L23 Data
Is a correction needed?
Yes
Refer to B - Correction Flow

Purchase Correction Transaction Flow


Correction Flow
B
Is corporate card?
No
Yes
Refer to Level 1 Flow
Was original transaction a VS Force Post or VS Completion?
No
Yes
Is the transaction in an Open Batch?
Yes
No
Was the Purchase/Capture processed on a different account?
No
Yes
VS Purchase Correction
VS Independent Refund
Option 2 With L23 Purchasing Data
VS Corpais with VS Purcha + VS Purchl
Option 2 With L23 Purchasing Data
VS Refund
Option 1 No L23 Data
Option 1 No L23 Data
End
Admin End Of Day Transactions
C
Batch Close
Open Totals

7.2.3 VS Completion

Once a Pre-authorization is obtained, the funds that are locked need to be retrieved from the customer's credit card. This VS Completion transaction is used to secure the funds locked by a pre-authorization transaction and readies them for settlement into the merchant account.

NOTE: Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to VS Corpais.

VS Completion transaction object definition

```
VsCompletion vsCompletion = new VsCompletion();
```

HttpsPostRequest object for VS Completion transaction object

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsCompletion);
```

VS Completion transaction object values

Table 1 VS Completion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsCompletion.setOrderId (order_id);`
Completion amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vsCompletion.setCompAmount (comp_amount);`
Transaction number	String	255-character alpha-numeric	`vsCompletion.setTxnNumber (txn_number);`
E-Commerce Indicator	String	1-character alpha-numeric	`vsCompletion.setCryptType (crypt);`

Table 2 Visa - Corporate Card Common Data - Level 2 Request Fields

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`vsCompletion .setNationalTax (national_tax);`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`vsCompletion .setMerchantVatNo (merchant_vat_no);`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`vsCompletion .setLocalTax (local_tax);`	Must reflect the amount of Local Tax (PST or QST) appear-ing on the invoice If Local Tax included then must not be all spaces or all zer-oes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01

Req*	Value	Limits	Set Method	Description
				Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alpha-numeric	`vsCompletion .setLocalTaxNo (local_tax_no);`	Merchant's Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alpha-numeric	`vsCompletion .setCustomerVatNo (customer_vat_no);`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alpha-numeric	`vsCompletion .setCri(cri);`	Value which the customer may choose to provide to the supplier at the point of sale – must be provided if given by the customer

Req*	Value	Limits	Set Method	Description
N	Customer Code	17-character alpha-numeric	`vsCompletion .setCustomerCode (customer_code);`	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alpha-numeric	`vsCompletion .setInvoiceNumber (invoice_number);`	Optional invoice number field that will not be passed along to Visa, but will be included on Moneris reporting

*Y = Required, N = Optional, C = Conditional

Sample VS Completion

```
package Level23;
import JavaAPI.*;
public class TestVsCompletion
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="ord-210916-15:14:46";
String comp_amount="5.00";
String txn_number = "19002-0_11";
String crypt="7";
String national_tax = "1.23";
String merchant_vat_no = "gstno111";
String local_tax = "2.34";
String customer_vat_no = "gstno999";
String cri = "CUST-REF-002";
String customer_code="ccvsfp";
String invoice_number="invsfp";
String local_tax_no="ltaxno";
VsCompletion vsCompletion = new VsCompletion();
vsCompletion.setOrderId(order_id);
vsCompletion.setCompAmount(comp_amount);
vsCompletion.setTxnNumber(txn_number);
vsCompletion.setCryptType(crypt);
vsCompletion.setNationalTax(national_tax);
```

Sample VS Completion

```
    vsCompletion.setMerchantVatNo(merchant_vat_no);
    vsCompletion.setLocalTax(local_tax);
    vsCompletion.setCustomerVatNo(customer_vat_no);
    vsCompletion.setCri(cri);
    vsCompletion.setCustomerCode(customer_code);
    vsCompletion.setInvoiceNumber(invoice_number);
    vsCompletion.setLocalTaxNo(local_tax_no);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(vsCompletion);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.2.4 VS Purchase Correction

The VS Purchase Correction (also known as a "void") transaction is used to cancel a transaction that was performed in the current batch. No amount is required because a void is always for 100% of the original transaction. The only transaction that can be voided using VS Purchase Correction is a VS Completion or VS Force Post. To send a void the order_id and txn_number from the VS Completion/VS Force Post are required.

VS Purchase Correction transaction object definition

```
VsPurchaseCorrection vsPurchaseCorrection = new VsPurchaseCorrection();
```

HttpsPostRequest object for VS Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsPurchaseCorrection);
```

VS Purchase Correction transaction object values

Table 1 VS Purchase Correction transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsPurchaseCorrection .setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`vsPurchaseCorrection .setTxnNumber(txn_number);`
E-Commerce Indicator	String	1-character alpha-numeric	`vsPurchaseCorrection .setCryptType(crypt);`

Sample VS Purchase Correction

```
package Level23;
import JavaAPI.*;

public class TestVsPurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485208113189";
String txn_number = "39793-0_11";
String crypt="7";
VsPurchaseCorrection vsPurchaseCorrection = new VsPurchaseCorrection();
vsPurchaseCorrection.setOrderId(order_id);
vsPurchaseCorrection.setTxnNumber(txn_number);
vsPurchaseCorrection.setCryptType(crypt);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vsPurchaseCorrection);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
```

Sample VS Purchase Correction

```
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.2.5 VS Force Post

The VS Force Post transaction is used to secure the funds locked by a pre-authorization transaction performed over IVR or equivalent terminal. When sending a force post request, you will need Order ID, Amount, Credit Card Number, Expiry Date, E-commerce Indicator and the Authorization Code received in the pre-authorization response.

NOTE: Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to VS Corpais.

VS Force Post transaction object definition

```
VsForcePost vsForcePost = new VsForcePost();
```

HttpsPostRequest object for VS Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsForcePost);
```

VS Force Post transaction object values

Table 1 VS Force Post transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsForcePost.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vsForcePost.setAmount(amount);`
Credit card number	String	20-character numeric	`vsForcePost.setPan(pan);`
Expiry Date	String	4-character numeric YYMM format	`vsForcePost.setExpDate(expiry_date);`
Authorization code	String	8-character alpha-numeric	`vsForcePost.setAuthCode(auth_code);`
E-commerce Indicator	String	1-character alpha-numeric	`vsForcePost.setCryptType(crypt);`

Table 2 VS Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vsForcePost.setCustId(cust_id);`

Table 3 Visa - Corporate Card Common Data - Level 2 Request Fields

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`vsForcePost.setNationalTax(national_tax);`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice.

Req*	Value	Limits	Set Method	Description
				Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`vsForcePost .setMerchantVatNo (merchant_vat_no);`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`vsForcePost .setLocalTax (local_tax);`	Must reflect the amount of Local Tax (PST or QST) appear-ing on the invoice If Local Tax included then must not be all spaces or all zer-oes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST)	15-character alpha-	`vsForcePost .setLocalTaxNo`	Merchant's

Req*	Value	Limits	Set Method	Description
	Registration Number	numeric	`(local_tax_no);`	Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alphanumeric	`vsForcePost .setCustomerVatNo (customer_vat_no);`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	`vsForcePost .setCri(cri);`	Value which the customer may choose to provide to the supplier at the point of sale – must be provided if given by the customer
N	Customer Code	17-character alphanumeric	`vsForcePost .setCustomerCode (customer_code);`	Optional customer code field that will not be passed

Req*	Value	Limits	Set Method	Description
				along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alphanumeric	`vsForcePost.setInvoiceNumber(invoice_number);`	Optional invoice number field that will not be passed along to Visa, but will be included on Moneris reporting

*Y = Required, N = Optional, C = Conditional

Sample VS Force Post

```
package Level23;
import JavaAPI.*;

public class TestVsForcePost
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id="CUST13343";
String amount="5.00";
String pan="4242424254545454";
String expiry_date="2012"; //YYMM
String auth_code="123456";
String crypt="7";
String national_tax = "1.23";
String merchant_vat_no = "gstno111";
String local_tax = "2.34";
String customer_vat_no = "gstno999";
String cri = "CUST-REF-002";
String customer_code="ccvsfp";
String invoice_number="invsfp";
String local_tax_no="ltaxno";
VsForcePost vsForcePost = new VsForcePost();
vsForcePost.setOrderId(order_id);
vsForcePost.setCustId(cust_id);
vsForcePost.setAmount(amount);
vsForcePost.setPan(pan);
vsForcePost.setExpDate(expiry_date);
```

Sample VS Force Post

```
    vsForcePost.setAuthCode(auth_code);
    vsForcePost.setCryptType(crypt);
    vsForcePost.setNationalTax(national_tax);
    vsForcePost.setMerchantVatNo(merchant_vat_no);
    vsForcePost.setLocalTax(local_tax);
    vsForcePost.setCustomerVatNo(customer_vat_no);
    vsForcePost.setCri(cri);
    vsForcePost.setCustomerCode(customer_code);
    vsForcePost.setInvoiceNumber(invoice_number);
    vsForcePost.setLocalTaxNo(local_tax_no);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(vsForcePost);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.2.6 VS Refund

VS Refund will credit a specified amount to the cardholder’s credit card. A refund can be sent up to the full value of the original VS Completion or VS Force Post. To send a VS Refund you will require the Order ID and Transaction Number from the original VS Completion or VS Force Post.

NOTE: Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to VS Corpais.

VS Refund transaction object definition

```
VsRefund vsRefund = new VsRefund();
```

HttpsPostRequest object for VS Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsRefund);
```

VS Refund transaction object values

Table 1 VS Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsRefund.setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`vsRefund.setTxnNumber(txn_number);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vsRefund.setAmount(amount);`
E-Commerce Indicator	String	1-character alpha-numeric	`vsRefund.setCryptType(crypt);`

Table 2 Visa - Corporate Card Common Data - Level 2 Request Fields

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`vsRefund .setNationalTax (national_tax);`	Must reflect the amount of National Tax

Req*	Value	Limits	Set Method	Description
				(GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`vsRefund .setMerchantVatNo (merchant_vat_no);`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`vsRefund .setLocalTax (local_tax);`	Must reflect the amount of Local Tax (PST or QST) appear-ing on the invoice If Local Tax included then must not be all spaces or all zer-oes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01 Maximum = 999999.99 Must have 2

Req*	Value	Limits	Set Method	Description
				decimal places
C	Local Tax (PST or QST) Registration Number	15-character alpha-numeric	`vsRefund .setLocalTaxNo (local_tax_no);`	Merchant's Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alpha-numeric	`vsRefund .setCustomerVatNo (customer_vat_no);`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alpha-numeric	`vsRefund .setCri(cri);`	Value which the customer may choose to provide to the supplier at the point of sale – must be provided if given by the customer
N	Customer Code	17-character alpha-numeric	`vsRefund .setCustomerCode`	Optional cus-

Req*	Value	Limits	Set Method	Description
			`(customer_code);`	tomer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alphanumeric	`vsRefund .setInvoiceNumber (invoice_number);`	Optional invoice number field that will not be passed along to Visa, but will be included on Moneris reporting

*Y = Required, N = Optional, C = Conditional

Sample VS Refund

```
package Level23;
import JavaAPI.*;
public class TestVsRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485208133961";
String amount="5.00";
String txn_number = "39795-0_11";
String crypt="7";
String national_tax = "1.23";
String merchant_vat_no = "gstno111";
String local_tax = "2.34";
String customer_vat_no = "gstno999";
String cri = "CUST-REF-002";
String customer_code="ccvsfp";
String invoice_number="invsfp";
String local_tax_no="ltaxno";
VsRefund vsRefund = new VsRefund();
vsRefund.setOrderId(order_id);
vsRefund.setAmount(amount);
vsRefund.setTxnNumber(txn_number);
vsRefund.setCryptType(crypt);
vsRefund.setNationalTax(national_tax);
vsRefund.setMerchantVatNo(merchant_vat_no);
```

Sample VS Refund

```
    vsRefund.setLocalTax(local_tax);
    vsRefund.setCustomerVatNo(customer_vat_no);
    vsRefund.setCri(cri);
    vsRefund.setCustomerCode(customer_code);
    vsRefund.setInvoiceNumber(invoice_number);
    vsRefund.setLocalTaxNo(local_tax_no);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(vsRefund);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.2.7 VS Independent Refund

VS Independent Refund will credit a specified amount to the cardholder’s credit card. The independent refund does not require an existing order to be logged in the Moneris Gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a pre-authorization.

NOTE: Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to VS Corpais.

VS Independent Refund transaction object definition

```
VsIndependentRefund vsIndependentRefund = new VsIndependentRefund();
```

HttpsPostRequest object for VS Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsIndependentRefund);
```

VS Independent Refund transaction object values

Table 1 VS Independent Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsIndependentRefund .setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vsIndependentRefund.setAmount (amount);`
Credit card number	String	20-character numeric	`vsIndependentRefund.setPan (pan);`
Expiry date	String	4-character numeric YYMM format	`vsIndependentRefund .setExpDate(expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`vsIndependentRefund .setCryptType(crypt);`

Table 2 VS Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vsIndependentRefund.setCustId (cust_id);`

Table 3 Visa - Corporate Card Common Data - Level 2 Request Fields

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`vsIndependentRefund.setNationalTax(national_tax);`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`vsIndependentRefund.setMerchantVatNo(merchant_vat_no);`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`vsIndependentRefund.setLocalTax(local_tax);`	Must reflect the amount of Local Tax (PST or QST) appearing on the invoice If Local Tax included then must not be all spaces or all zeroes; Must be provided if Local Tax (PST or QST) applies Minimum =

Req*	Value	Limits	Set Method	Description
				0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alpha-numeric	`vsIndependentRefund.setLocalTaxNo(local_tax_no);`	Merchant's Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alpha-numeric	`vsIndependentRefund.setCustomerVatNo(customer_vat_no);`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alpha-numeric	`vsIndependentRefund.setCri(cri);`	Value which the customer may choose to provide to the supplier at the point of sale – must be

Req*	Value	Limits	Set Method	Description
				provided if given by the customer
N	Customer Code	17-character alpha-numeric	`vsIndependentRefund.setCustomerCode(customer_code);`	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alpha-numeric	`vsIndependentRefund.setInvoiceNumber(invoice_number);`	Optional invoice number field that will not be passed along to Visa, but will be included on Moneris reporting

*Y = Required, N = Optional, C = Conditional

Sample VS Independent Refund

```
package Level23;
import JavaAPI.*;
public class TestVsIndependentRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id="CUST13343";
String amount="5.00";
String pan="4242424254545454";
String expiry_date="2012"; //YYMM
String crypt="7";
String national_tax = "1.23";
String merchant_vat_no = "gstno111";
String local_tax = "2.34";
String customer_vat_no = "gstno999";
String cri = "CUST-REF-002";
String customer_code="ccvsfp";
```

Sample VS Independent Refund

```
String invoice_number="invsfp";
String local_tax_no="ltaxno";
VsIndependentRefund vsIndependentRefund = new VsIndependentRefund();
vsIndependentRefund.setOrderId(order_id);
vsIndependentRefund.setCustId(cust_id);
vsIndependentRefund.setAmount(amount);
vsIndependentRefund.setPan(pan);
vsIndependentRefund.setExpDate(expiry_date);
vsIndependentRefund.setCryptType(crypt);
vsIndependentRefund.setNationalTax(national_tax);
vsIndependentRefund.setMerchantVatNo(merchant_vat_no);
vsIndependentRefund.setLocalTax(local_tax);
vsIndependentRefund.setCustomerVatNo(customer_vat_no);
vsIndependentRefund.setCri(cri);
vsIndependentRefund.setCustomerCode(customer_code);
vsIndependentRefund.setInvoiceNumber(invoice_number);
vsIndependentRefund.setLocalTaxNo(local_tax_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vsIndependentRefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.2.8 VS Corpais

VS Corpais will contain all the required and optional data fields for Level 2/3 Purchasing Card Addendum data. VS Corpais data can be sent when the card has been identified in the Pre-authorization transaction request as being a corporate card.

In addition to the Order ID and Transaction number, this transaction also contains two objects:

- VS Purcha – Corporate Card Common Data
- VS Purchl – Line Item Details

VS Corpais request must be preceded by a financial transaction (VS Completion, VS Force Post, VS Refund, VS Independent Refund) and the Corporate Card flag must be set to "true" in the Pre-authorization response.

VS Corpais transaction object definition

```
VsCorpais vsCorpais = new VsCorpais();
```

HttpsPostRequest object for VS Corpais transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(vsCorpais);
```

VS Corpais transaction object values

Table 1 VS Corpais transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vsCorpais.setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`vsCorpais.setTxnNumber(txn_number);`

vsPurcha
 	
For a list of the variables that appear in this object, see the table below
	Object	n/a	`VsPurcha vsPurcha = new VsPurcha();`
			
			`vsCorpais.setVsPurch(vsPurcha,vsPurchl);`

vsPurchl
 	
For a list of the variables that appear in this object, see the table below
	Object	n/a	`VsPurchl vsPurchl = new VsPurchl();`
			
			`vsCorpais.setVsPurch(vsPurcha,vsPurchl);`

*Y = Required, N = Optional, C = Conditional

7.2.8.1 VS Purcha - Corporate Card Common Data

VS Corpais transactions use the VS Purcha object to contain Level 2 data.

Table 1 Corporate Card Common Data - Level 2 Request Fields - VSPurcha

Req*	Value	Limits	Set Method	Description
C	Buyer Name	30-character alphanumeric	`vsPurcha.setBuyerName (buyer_name);`	Buyer/Recipient Name **NOTE:** Name required by CRA on transactions >$150
C	Local Tax Rate	4-character decimal	`vsPurcha .setLocalTaxRate(local_ tax_rate);.`	Indicates the detailed tax rate applied in relationship to a local tax amount **EXAMPLE:** 8% PST should be 8.0 Minimum = 0.01 Maximum = 99.99 **NOTE:** Must be provided if Local Tax (PST or QST) applies.
N	Duty Amount	9-character decimal	`vsPurcha.setDutyAmount (duty_amount);`	Duty on total purchase amount A minus sign means 'amount is a credit', plus sign or no sign means 'amount is a debit' maximum without sign is 999999.99
N	Invoice Discount Treatment	1-character numeric	`vsPurcha .setDiscountTreatment (discount_treatment);`	Indicates how the merchant is managing discounts Must be one of the following values: 0 - if no invoice level discounts apply for this

Req*	Value	Limits	Set Method	Description
				invoice 1 - if Tax was calculated on Post-Discount totals 2 - if Tax was calculated on Pre-Discount totals
N	Invoice Level Discount Amount	9-character decimal	`vsPurcha.setDiscountAmt (discount_amt);`	Amount of discount (if provided at the invoice level according to the Invoice Discount Treatment) Must be non-zero if Invoice Discount Treatment is 1 or 2 Minimum amount is 0.00 and maximum is 999999.99
C	Ship To Postal Code / Zip Code	10-character alphanumeric	`vsPurcha .setShipToPostalCode (ship_to_pos_code);`	The postal code or zip code for the destination where goods will be delivered **NOTE:** Required if shipment is involved Full alpha postal code - Valid ANA<space>NAN format required if shipping to an address within Canada
C	Ship From Postal Code / Zip Code	10-character alphanumeric	`vsPurcha .setShipFromPostalCode (ship_from_pos_code);`	The postal code or zip code from which items were shipped For Canadian

Req*	Value	Limits	Set Method	Description
				addresses,requires full alpha postal code for the merchant with Valid ANA<space>NAN format
C	Destination Country Code	2-character alphanumeric	`vsPurcha.setDesCouCode(des_cou_code);`	Code of country where purchased goods will be delivered Use ISO 3166-1 alpha-2 format **NOTE:** Required if it appears on the invoice for an international transaction
Y	Unique VAT Invoice Reference Number	25-character alphanumeric	`vsPurcha.setVatRefNum(vat_ref_num);`	Unique Value Added Tax Invoice Reference Number Must be populated with the invoice number and this cannot be all spaces or zeroes
Y	Tax Treatment	1-character alphanumeric	`vsPurcha.setTaxTreatment(tax_treatment);`	Must be one of the following values: 0 = Net Prices with tax calculated at line item level; 1 = Net Prices with tax calculated at invoice level; 2 = Gross prices given with tax information provided at line item level; 3 = Gross prices given with tax information provided at invoice level;

Req*	Value	Limits	Set Method	Description
				4 = No tax applies (small merchant) on the invoice for the transaction
N	Freight/Shipping Amount (Ship Amount)	9-character decimal	`vsPurcha .setFreightAmount (freight_amount);`	Freight charges on total purchase If shipping is not provided as a line item it must be provided here, if applicable Signed monetary amount: Minus (-) sign means 'amount is a credit', Plus (+) sign or no sign means 'amount is a debit' Maximum without sign is 999999.99
C	GST HST Freight Rate	4-character decimal	`vsPurcha .setGstHstFreightRate (gst_hst_freight_rate);`	Rate of GST (excludes PST) or HST charged on the shipping amount (in accordance with the Tax Treatment) If Freight/Shipping Amount is provided then this (National GST or HST) tax rate must be provided. Monetary amount, maximum is 99.99. Such as 13% HST is 13.00
C	GST HST Freight Amount	9-character decimal	`vsPurcha .setGstHstFreightAmount (gst_hst_freight_`	Amount of GST (excludes PST) or HST charged on the

Req*	Value	Limits	Set Method	Description
			`amount);`	shipping amount If Freight/Shipping Amount is provided then this (National GST or HST) tax amount must be provided if taxTreatment is 0 or 2 Signed monetary amount: maximum without sign is 999999.99.

7.2.8.2 VS Purchl - Line Item Details

VS Corpais transactions use the VS Purchl object to contain Level 3 data.

Line Item Details for VS Purchl

```
String[] item_com_code = {"X3101", "X84802"};

String[] product_code = {"CHR123", "DDSK200"};

String[] item_description = {"Office Chair", "Disk Drive"};

String[] item_quantity = {"3", "1"};

String[] item_uom = {"EA", "EA"};

String[] unit_cost = {"0.20", "0.40"};

String[] vat_tax_amt = {"0.00", "0.00"};

String[] vat_tax_rate = {"13.00", "13.00"};

String[] discount_treatmentL = {"0", "0"};

String[] discount_amtL = {"0.00", "0.00"};
```

Setting VS Purchl Line Item Details

```
vsPurchl.setVsPurchl(item_com_code[0], product_code[0], item_description[0],
item_quantity[0], item_uom[0], unit_cost[0], vat_tax_amt[0], vat_tax_rate[0],
discount_treatmentL[0], discount_amtL[0]);

vsPurchl.setVsPurchl(item_com_code[1], product_code[1], item_description[1],
item_quantity[1], item_uom[1], unit_cost[1], vat_tax_amt[1], vat_tax_rate[1],
discount_treatmentL[1], discount_amtL[1]);
```

Table 1 Corporate Card Common Data - Level 3 Request Fields - VSPurchl

Req*	Value	Limits	Variable/Field	Description
C	Item Commodity Code	12-character alpha-numeric	item_com_code	Line item Comod-ity Code (if this field is not sent, then Product Code must be sent)
Y	Product Code	12-character alpha-numeric	product_code	Product code for this line item – mer-chant's product code, man-ufacturer's product code or buyer's product code Typically this will be the SKU or iden-tifier by which the merchant tracks and prices the item or service This should always be provided for every line item
Y	Item Description	35-character alpha-numeric	item_description	Line item descrip-tion

Y	Item Quantity	12-character decimal	item_quantity	Quantity invoiced for this line item
				
				Up to 4 decimal places supported, whole numbers are accepted
				
				Minimum = 0.0001
				
				Maximum = 999999999999

Y	Item Unit of Measure	2-character alpha-numeric	item_uom	Unit of measure Use ANSI X-12 EDI Allowable Units of Measure and

Req*	Value	Limits	Variable/Field	Description
				Codes

Y	Item Unit Cost	12-character decimal	unit_cost	Line item cost per unit
				
				2-4 decimal places accepted
				
				Minimum = 0.0001
				
				Maximum = 999999.9999

N	VAT Tax Amount	12-character decimal	vat_tax_amt	Any value-added tax or other sales tax amount
				
				Must have 2 decimal places
				
				Minimum = 0.01
				
				Maximum = 999999.99

N	VAT Tax Rate	4-character decimal	vat_tax_rate	Sales tax rate **EXAMPLE:** 8% PST should be 8.0 maximum 99.99
Y	Discount Treat-ment	1-character numeric	discount_treatmentL	Must be one of the following values: 0 if no invoice level dis-counts apply for this invoice 1 if Tax was calculated on Post-Discount totals 2 if Tax was calculated on Pre-Discount totals
C	Discount Amount	12-character decimal	discount_amtL	Amount of dis-count, if provided for this line item according to the Line Item Discount

Req*	Value	Limits	Variable/Field	Description
				Treatment Must be non-zero if Line Item Discount Treatment is 1 or 2 Must have 2 decimal places Minimum = 0.01 Maximum = 999999.99

7.2.8.3 Sample Code for VS Corpais

Sample VS Corpais

```
package Level23;
import JavaAPI.*;

public class TestVsCorpais
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485208069127";
String txn_number="39791-0_11";
String buyer_name = "Buyer Manager";
String local_tax_rate = "13.00";
String duty_amount = "0.00";
String discount_treatment = "0";
String discount_amt = "0.00";
String freight_amount = "0.20";
String ship_to_pos_code = "M8X 2W8";
String ship_from_pos_code = "M1K 2Y7";
String des_cou_code = "CAN";
String vat_ref_num = "VAT12345";
String tax_treatment = "3";//3 = Gross prices given with tax information provided at invoice level
String gst_hst_freight_amount = "0.00";
String gst_hst_freight_rate = "13.00";
String[] item_com_code = {"X3101", "X84802"};
String[] product_code = {"CHR123", "DDSK200"};
String[] item_description = {"Office Chair", "Disk Drive"};
String[] item_quantity = {"3", "1"};
String[] item_uom = {"EA", "EA"};
String[] unit_cost = {"0.20", "0.40"};
String[] vat_tax_amt = {"0.00", "0.00"};
String[] vat_tax_rate = {"13.00", "13.00"};
```

Sample VS Corpais

```
String[] discount_treatmentL = {"0", "0"};
String[] discount_amtL = {"0.00", "0.00"};
//Create and set VsPurcha
VsPurcha vsPurcha = new VsPurcha();
vsPurcha.setBuyerName(buyer_name);
vsPurcha.setLocalTaxRate(local_tax_rate);
vsPurcha.setDutyAmount(duty_amount);
vsPurcha.setDiscountTreatment(discount_treatment);
vsPurcha.setDiscountAmt(discount_amt);
vsPurcha.setFreightAmount(freight_amount);
vsPurcha.setShipToPostalCode(ship_to_pos_code);
vsPurcha.setShipFromPostalCode(ship_from_pos_code);
vsPurcha.setDesCouCode(des_cou_code);
vsPurcha.setVatRefNum(vat_ref_num);
vsPurcha.setTaxTreatment(tax_treatment);
vsPurcha.setGstHstFreightAmount(gst_hst_freight_amount);
vsPurcha.setGstHstFreightRate(gst_hst_freight_rate);
//Create and set VsPurchl
VsPurchl vsPurchl = new VsPurchl();
vsPurchl.setVsPurchl(item_com_code[0], product_code[0], item_description[0], item_quantity[0],
item_uom[0], unit_cost[0], vat_tax_amt[0], vat_tax_rate[0], discount_treatmentL[0], discount_amtL
[0]);
vsPurchl.setVsPurchl(item_com_code[1], product_code[1], item_description[1], item_quantity[1],
item_uom[1], unit_cost[1], vat_tax_amt[1], vat_tax_rate[1], discount_treatmentL[1], discount_amtL
[1]);

VsCorpais vsCorpais = new VsCorpais();
vsCorpais.setOrderId(order_id);
vsCorpais.setTxnNumber(txn_number);
vsCorpais.setVsPurch(vsPurcha, vsPurchl);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vsCorpais);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
```

Sample VS Corpais

```
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.3 Level 2/3 MasterCard Transactions

7.3.1 Level 2/3 Transaction Types for MasterCard

This transaction set includes a suite of corporate card financial transactions as well as a transaction that allows for the passing of Level 2/3 data. Please ensure MC Level 2/3 processing support is enabled on your merchant account. Batch Close, Open Totals and Pre-authorization are identical to the transactions outlined in the section Basic Transaction Set (page 12).

When the Preauth response contains CorporateCard equal to true then you can submit the MC transactions.

If CorporateCard is false then the card does not support Level 2/3 data and non Level 2/3 transaction are to be used. If the card is not a corporate card, please refer to section 4 for the appropriate non-corporate card transactions.

NOTE: This transaction set is intended for transactions where Corporate Card is true and Level 2/3 data will be submitted. If the credit card is found to be a corporate card but you do not wish to send any Level 2/3 data then you may submit MC transactions using the transaction set outlined in Basic Transaction Set (page 12).

Pre-auth – (authorization/pre-authorization)
The pre-auth verifies and locks funds on the customer’s credit card. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from a pre-auth so that they may be settled in the merchant account a capture must be performed. Level 2/3 data submission is not supported as part of a pre-auth as a pre-auth is not settled. When CorporateCard is returned true then Level 2/3 data may be submitted.

MC Completion – (Capture/Preauth Completion)

Once a Pre-authorization is obtained the funds that are locked need to be retrieved from the customer's credit card. The capture retrieves the locked funds and readies them for settlement in to the merchant account. Prior to performing an MCCompletion a Pre-auth must be performed.

MC Force Post – (Force Capture/Preauth Completion)

This transaction is an alternative to MC Completion to obtain the funds locked on Preauth obtained from IVR or equivalent terminal. The MC Force Post requires that the original Pre-authorization's auth code is provided and it retrieves the locked funds and readies them for settlement in to the merchant account.

MC Purchase Correction – (Void, Correction)

MC Completions can be voided the same day* that they occur. A void must be for the full amount of the transaction and will remove any record of it from the cardholder statement. * An MC Purchase Correction can be performed against a transaction as long as the batch that contains the original transaction remains open. When using the automated closing feature batch close occurs daily between 10 – 11 pm EST.

MC Refund – (Credit)

A MC Refund can be performed against an MC Completion or MC Force Post to refund an amount less than or equal to the amount of the original transaction.

MC Independent Refund – (Credit)

A MC Indpendent Refund can be performed against an completion to refund any part, or all of the transaction. Independent refund is used when the originating transaction was not performed through Moneris Gateway. Please note, the MC Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an MC Independent Refund, it may mean the transaction is not supported on your account. If you wish to have the MC Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-866-319-7450.

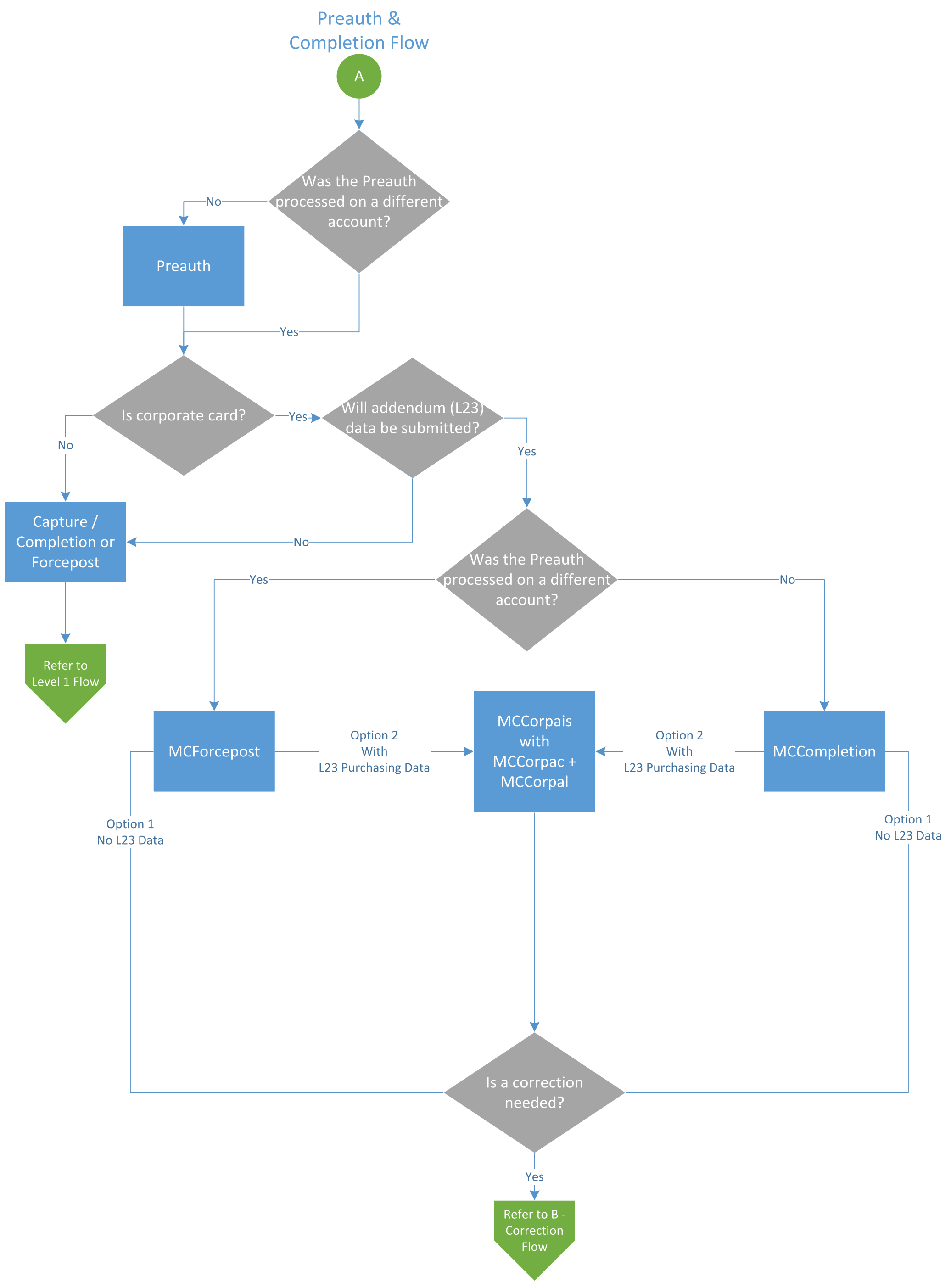
MC Corpais Common Line Item – (Level 2/3 Data)

MC Corpais Common Line Item will contain the entire required and optional data field for Level 2/3 data. MCCorpais Common Line Item data can be sent when the card has been identified in the transaction request as being a corporate card. This transaction supports multiple data types and combinations:

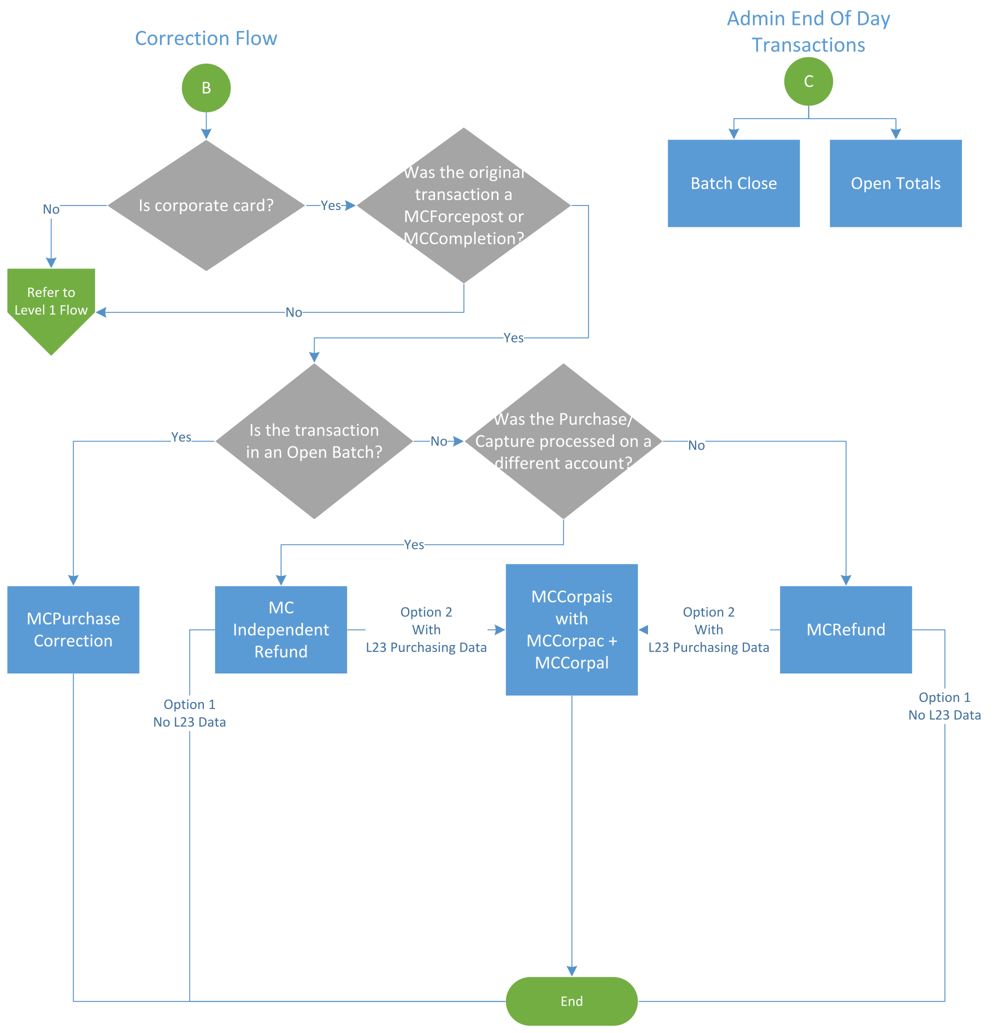
- Purchasing Card Data:
  - Corporate card common data with Line Item Details

7.3.2 Level 2/3 Transaction Flow for MasterCard

Pre-authorization/Completion Transaction Flow


Preauth & Completion Flow
A
Was the Preauth processed on a different account?
No
Preauth
Yes
Is corporate card?
Yes
Will addendum (L23) data be submitted?
No
Yes
Capture / Completion or Forcepost
No
Refer to Level 1 Flow
Was the Preauth processed on a different account?
Yes
No
MCForcepost
Option 2 With L23 Purchasing Data
MCCorpais with MCCorpac + MCCorpal
Option 2 With L23 Purchasing Data
MCCompletion
Option 1 No L23 Data
Option 1 No L23 Data
Is a correction needed?
Yes
Refer to B - Correction Flow

Purchase Correction Transaction Flow


Correction Flow
B
Is corporate card?
No
Yes
Refer to Level 1 Flow
Was the original transaction a MCForcepost or MCCompletion?
No
Yes
Is the transaction in an Open Batch?
Yes
No
Was the Purchase/ Capture processed on a different account?
No
Yes
MCPurchase Correction
MC Independent Refund
Option 2 With L23 Purchasing Data
MCCorpais with MCCorpac + MCCorpal
Option 2 With L23 Purchasing Data
MCRefund
Option 1 No L23 Data
Option 1 No L23 Data
End
Admin End Of Day Transactions
C
Batch Close
Open Totals

7.3.3 MC Completion

The MC Completion transaction is used to secure the funds locked by a pre-authorization transaction. When sending a capture request you will need two pieces of information from the original pre-authorization– the Order ID and the transaction number from the returned response.

Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to MC Corpais.

MC Completion transaction object definition

```
McCompletion mcCompletion = new McCompletion();
```

HttpsPostRequest object for MC Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcCompletion);
```

MC Completion transaction object values

Table 1 MC Completion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`mcCompletion.setOrderId (order_id);`
Completion amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`mcCompletion.setCompAmount (comp_amount);`
Transaction number	String	255-character alpha-numeric	`mcCompletion.setTxnNumber (txn_number);`
Merchant reference number	String	19-character alpha-numeric	`mcCompletion.setMerchantRefNo (merchant_ref_no);`
E-commerce indicator	String	1-character alpha-numeric	`mcCompletion.setCryptType (crypt);`

Sample MC Completion

```
package Level23;
import JavaAPI.*;

public class TestMcCompletion
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485206444761";
String comp_amount="1.00";
String txn_number="39777-0_11";
String crypt="7";
String merchant_ref_no = "319038";
McCompletion mcCompletion = new McCompletion();
mcCompletion.setOrderId(order_id);
mcCompletion.setCompAmount(comp_amount);
mcCompletion.setTxnNumber(txn_number);
mcCompletion.setCryptType(crypt);
mcCompletion.setMerchantRefNo(merchant_ref_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcCompletion);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.3.4 MC Force Post

MC Force Post transaction is used to secure the funds locked by a pre-authorization transaction performed over IVR or equivalent terminal`. When sending a force post request, you will need order_id, amount, pan (card number), expiry date, crypt type and the authorization code received in the pre-authorization response.

MC Force Post transaction object definition

```
McForcePost mcforcepost= new McForcePost();
```

HttpsPostRequest object for MC Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcforcepost);
```

MC Force Post transaction object values

Table 1 MC Force Post transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`mcforcepost.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`mcforcepost.setAmount(amount);`
Credit card number	String	20-character alpha-numeric	`mcforcepost.setPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`mcforcepost.setExpDate(expiry_date);`

Value	Type	Limits	Set Method
Authorization code	String	8-character alpha-numeric	`mcforcepost.setAuthCode(auth_code);`
E-commerce indicator	String	1-character alpha-numeric	`mcforcepost.setCryptType(crypt);`
Merchant reference number	String	19-character alpha-numeric	`mcforcepost.setMerchantRefNo(merchant_ref_no);`

Table 2 MC Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`mcforcepost.setCustId(cust_id);`

Sample MC Force Post

```
package Level23;
import JavaAPI.*;

public class TestMcForcePost
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id = "CUST13343";
String amount = "5.00";
String pan = "5454545442424242";
String expiry_date = "1912"; //YYMM
String auth_code = "123456";
String crypt = "7";
String merchant_ref_no = "319038";
McForcePost mcforcepost = new McForcePost();
mcforcepost.setOrderId(order_id);
mcforcepost.setCustId(cust_id);
mcforcepost.setAmount(amount);
mcforcepost.setPan(pan);
mcforcepost.setExpDate(expiry_date);
mcforcepost.setAuthCode(auth_code);
mcforcepost.setCryptType(crypt);
mcforcepost.setMerchantRefNo(merchant_ref_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
```

Sample MC Force Post

```
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(mcforcepost);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.3.5 MC Purchase Correction

The MC Purchase Correction (void) transaction is used to cancel a transaction that was performed in the current batch. No amount is required because a void is always for 100% of the original transaction. The only transaction that can be voided is completion. To send a void, the Order ID and Transaction Number from the MC Completion or MC Force Post are required.

MC Purchase Correction transaction object definition

```
McPurchaseCorrection mcpurchasecorrection = new McPurchaseCorrection();
```

HttpsPostRequest object for MC Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcpurchasecorrection);
```

MC Purchase Correction transaction object values

Table 1 MC Purchase Correction transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`mcpurchasecorrection .setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`mcpurchasecorrection .setTxnNumber(txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`mcpurchasecorrection .setCryptType(crypt);`

Sample MC Purchase Correction

```
package Level23;
import JavaAPI.*;

public class TestMcPurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485207871499";
String txn_number="660117311902017023164431860-0_11";
String crypt="7";
McPurchaseCorrection mcpurchasecorrection = new McPurchaseCorrection();
mcpurchasecorrection.setOrderId(order_id);
mcpurchasecorrection.setTxnNumber(txn_number);
mcpurchasecorrection.setCryptType(crypt);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpurchasecorrection);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
```

Sample MC Purchase Correction

```
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.3.6 MC Refund

The MC Refund will credit a specified amount to the cardholder's credit card. A refund can be sent up to the full value of the original capture. To send a refund you will require the Order ID and Transaction Number from the original MC Completion or MC Force Post.

MC Refund transaction object definition

```
McRefund mcRefund = new McRefund();
```

HttpsPostRequest object for MC Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcRefund);
```

MC Refund transaction object values

Table 1 MC Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`mcRefund.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`mcRefund.setAmount(amount);`
Transaction number	String	255-character alpha-numeric	`mcRefund.setTxnNumber(txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`mcRefund.setCryptType(crypt);`
Merchant reference number	String	19-character alpha-numeric	`mcRefund.setMerchantRefNo(merchant_ref_no);`

Sample MC Refund

```
package Level23;
import JavaAPI.*;
public class TestMcRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485207913048";
String amount="5.00";
String txn_number="660117311902017023164513403-0_11";
String crypt="7";
String merchant_ref_no = "319038";
McRefund mcRefund = new McRefund();
mcRefund.setOrderId(order_id);
mcRefund.setAmount(amount);
```

Sample MC Refund

```
mcRefund.setTxnNumber(txn_number);
mcRefund.setCryptType(crypt);
mcRefund.setMerchantRefNo(merchant_ref_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcRefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.3.7 MC Independent Refund

MC Independent Refund is used when the originating transaction was not performed through Moneris Gateway and does not require an existing order to be logged in the Moneris Gateway; however, the credit card number and the expiry date will need to be passed. The transaction format is almost identical to a purchase or a pre-authorization.

NOTE: Independent refund transactions are not supported on all accounts. If you receive a transaction not allowed error when attempting an independent refund transaction, it may mean the feature is not supported on your account. To have Independent Refund transaction functionality temporarily enabled (or re-enabled), please contact the MonerisCustomer Service Centre at 1-866-319-7450.

Once you have completed this transaction successfully, to submit the complete supplemental level 2/3 data, please proceed to MC Corpais.

MC Independent Refund transaction object definition

```
McIndependentRefund mcindrefund = new McIndependentRefund();
```

HttpsPostRequest object for MC Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcindrefund);
```

MC Independent Refund transaction object values

Table 1 MC Independent Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`mcindrefund.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`mcindrefund.setAmount(amount);`
E-commerce indicator	String	1-character alphanumeric	`mcindrefund.setCryptType(crypt);`
Credit card number	String	20-character numeric	`mcindrefund.setPan(pan);`
Expiry date	String	4-character numeric (YYMM format)	`mcindrefund.setExpDate(expiry_date);`
Merchant reference number	String	19-character alphanumeric	`mcindrefund.setMerchantRefNo(merchant_ref_no);`

Table 2 MC Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`mcindrefund.setCustId(cust_id);`

Sample MC Independent Refund

```
package Level23;
import JavaAPI.*;
public class TestMcIndependentRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id = "CUST13343";
String amount = "5.00";
String pan = "5454545442424242";
String expiry_date = "1912"; //YYMM
String crypt = "7";
String merchant_ref_no = "319038";
McIndependentRefund mcindrefund = new McIndependentRefund();
mcindrefund.setOrderId(order_id);
mcindrefund.setCustId(cust_id);
mcindrefund.setAmount(amount);
mcindrefund.setPan(pan);
mcindrefund.setExpDate(expiry_date);
mcindrefund.setCryptType(crypt);
mcindrefund.setMerchantRefNo(merchant_ref_no);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcindrefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
```

Sample MC Independent Refund

```
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.3.8 MC Corpais - Corporate Card Common Data with Line Item Details

This transaction example includes the following elements for Level 2 and 3 purchasing card corporate card data processing:

- Corporate Card Common Data (MC Corpac)
  - only 1 set of MC Corpac fields can be submitted
  - this data set includes data elements that apply to the overall order, e.g., the total overall taxes
- Line Item Details (MC Corpal)
  - 1-998 counts of MC Corpal line items can be submitted
  - This data set includes the details about each individual item or service purchased

The MC Corpais request must be preceded by a financial transaction (MC Completion, MC Force Post, MC Refund, MC Independent Refund) and the Corporate Card flag must be set to “true” in the Preauthorization response. The MC Corpais request will need to contain the Order ID of the financial transaction as well as the Transaction Number.

In addition, MC Corpais has a tax array object that can be sent via the Tax fields in MC Corpac and MC Corpal. For more about the tax array object, see 7.3.8.3 Tax Array Object - MC Corpais.

For descriptions of the Level 2/3 fields, please see Definition of Request Fields for Level 2/3 - MasterCard (page 472).

MC Corpais transaction object definition

```
McCorpais mcCorpais = new McCorpais();
```

HttpsPostRequest object for MC Corpais transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcCorpais);
```

MC Corpais transaction object values

Table 1 MC Corpais transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`mcCorpais.setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`mcCorpais.setTxnNumber(txn_number);`
MCCorpac	Object	n/a	`mcCorpac.setMcCorpac(mcCorpac);`
MC Corpal	Object	n/a	`mcCorpais.setMcCorpal(mcCorpal);`

*Y = Required, N = Optional, C = Conditional

7.3.8.1 MC Corpac - Corporate Card Common Data

Table 1 Corporate Card Common Data - Level 2 Request Fields - MCCorpac

Req*	Value	Limits	Set Method	Description
N	Austin-Tetra Number	15-character alpha-numeric	`mcCorpac.setAustinTetraNumber(austin_tetra_number);`	The Austin-Tetra Number assigned to the card acceptor
N	NAICS Code	15-character alpha-numeric	`mcCorpac.setNaicsCode(naics_code);`	North American Industry Classification System (NAICS) code assigned to the card acceptor
N	Customer Code	25-character alpha-numeric	`mcCorpac.setCustomerCode1(customer_code1_c);`	A control number, such as purchase order number, project number, department allocation number or name that the purchaser supplied the merchant Left-justified; may be spaces
N	Unique Invoice	17-character alpha-	`mcCorpac.setUniqueInvoiceNumber`	Unique number associated with the individual trans-

Req*	Value	Limits	Set Method	Description
	Number	numeric	`(unique_invoice_number_c);`	action provided by the merchant
N	Commodity Code	15-character alphanumeric	`mcCorpac.setCommodityCode (commodity_code);`	Code assigned by the merchant that best categorizes the item(s) being purchased
N	Order Date	6-character numeric YYMMDD format	`mcCorpac.setOrderDate (order_date_c);`	The date the item was ordered **NOTE:** If present, must contain a valid date
N	Corporation VAT Number	20-character alphanumeric	`mcCorpac .setCorporationVatNumber (corporation_vat_number_c);`	Contains a corporation's value added tax (VAT) number
N	Customer VAT Number	20-character alphanumeric	`mcCorpac .setCustomerVatNumber (customer_vat_number_c);`	Contains the VAT number for the customer / cardholder used to identify the customer when purchasing goods and services from the merchant
N	Freight Amount	12-character decimal	`mcCorpac.setFreightAmount1 (freight_amount_c);`	The freight on the total purchase Must have 2 decimals Minimum = 0.00 Maximum = 999999.99

N	Duty Amount	12-character decimal	`mcCorpac.setDutyAmount1 (duty_amount_c);`	The duty on the total purchase
				
				Must have 2 decimals
				
				Minimum = 0.00
				
				Maximum = 999999.99

N	Destination State /	3-character alphanumeric	`mcCorpac .setDestinationProvinceCode (destination_province_`	State or Province of the country where the goods will be delivered

Req*	Value	Limits	Set Method	Description
	Province Code		`code);`	Left justified with trailing spaces **EXAMPLE:** ONT = Ontario
N	Destination Country Code	3-character alpha-numeric ISO 3166-1 alpha-3 format	`mcCorpac .setDestinationCountryCode (destination_country_code);`	The country code where goods will be delivered Left justified with trailing spaces ISO 3166-1 alpha-3 format **EXAMPLE:** CAN = Canada
N	Ship From Postal Code	10-character alpha-numeric ANA NAN format	`mcCorpac.setShipFromPosCode (ship_from_pos_code);`	The postal code or zip code from which items were shipped Full alpha postal code - Valid ANA<space>NAN format
N	Destination Postal Code	10-character alpha-numeric	`mcCorpac.setShipToPosCode (ship_to_pos_code_c);`	The postal code or zip code where goods will be delivered Full alpha postal code - Valid ANA<space>NAN format if shipping to an address within Canada
N	Authorized Contact Name	36-character alpha-numeric	`mcCorpac .setAuthorizedContactName (authorized_contact_name_ c);`	Name of an individual or company contacted for company authorized purchases
N	Authorized Contact Phone	17-character alpha-numeric	`mcCorpac .setAuthorizedContactPhone (authorized_contact_phone);`	Phone number of an individual or company contacted for company authorized purchases
N	Additional Card Acceptor	40-character alpha-numeric	`mcCorpac .setAdditionalCardAcceptorD ata(additional_card_`	Information pertaining to the card acceptor

Req*	Value	Limits	Set Method	Description
	Data		`acceptor_data);`	
N	Card Acceptor Type	8-character alpha-numeric	`mcCorpac .setCardAcceptorType(card_ acceptor_type);`	Various classifications of business ownership characteristics This field takes 8 characters. Each character represents a different component, as follows: 1st character represents ‘Business Type’ and contains a code to identify the specific classification or type of business: 1. Corporation 2. Not known 3. Individual/Sole Proprietorship 4. Partnership 5. Association/Estate/Trust 6. Tax Exempt Organizations (501C) 7. International Organization 8. Limited Liability Company (LLC) 9. Government Agency 2nd character represents 'Business Owner Type'. Contains a code to identify specific characteristics about the business owner. 1 - No application classification 2 - Female business owner 3 - Physically handicapped female

Req*	Value	Limits	Set Method	Description
				business owner 4 - Physically handicapped male business owner 0 - Unknown 3rd character represents 'Business Certification Type'. Contains a code to identify specific characteristics about the business certification type, such as small business, disadvantaged, or other certification type: 1 - Not certified 2 - Small Business Administration (SBA) certification small business 3 - SBA certification as small disadvantaged business 4 - Other government or agency-recognized certification (such as Minority Supplier Development Council) 5 - Self-certified small business 6 - SBA certification as small and other government or agency-recognized certification 7 - SBA certification as small disadvantaged busi-

Re-q*	Value	Limits	Set Method	Description
				ness and other government or agency-recognized certification 8 - Other government or agency-recognized certification and self-certified small business A - SBA certification as 8(a) B - Self-certified small disadvantaged business (SDB) C - SBA certification as HUBZone 0 - Unknown 4th character represents 'Business Racial/Ethnic Type'. Contains a code identifying the racial or ethnic type of the majority owner of the business. 1 - African American 2 - Asian Pacific American 3 - Subcontinent Asian American 4 - Hispanic American 5 - Native American Indian 6 - Native Hawaiian 7 - Native Alaskan 8 - Caucasian 9 - Other 0 - Unknown 5th character represents 'Business Type Provided Code'

Req*	Value	Limits	Set Method	Description
				Y - Business type is provided. N - Business type was not provided. R - Card acceptor refused to provide business type 6th character represents 'Business Owner Type Provided Code' Y - Business owner type is provided. N - Business owner type was not provided. R - Card acceptor refused to provide business type 7th character represents 'Business Certification Type Provided Code' Y - Business certification type is provided. N - Business certification type was not provided. R - Card acceptor refused to provide business type 8th character represents 'Business Racial/Ethnic Type' Y - Business racial/ethnic type is provided. N - Business

Req*	Value	Limits	Set Method	Description
				racial/ethnic type was not provided. R - Card acceptor refused to provide business racial/ethnic type
N	Card Acceptor Tax ID	20-character alphanumeric	`mcCorpac.setCardAcceptorTaxTd(card_acceptor_tax_id_c);`	US federal tax ID number or value-added tax (VAT) ID
N	Card Acceptor Reference Number	25-character alphanumeric	`mcCorpac.setCardAcceptorReferenceNumber(card_acceptor_reference_number);`	Code that facilitates card acceptor/corporation communication and record keeping
N	Card Acceptor VAT Number	20-character alphanumeric	`mcCorpac.setCardAcceptorVatNumber(card_acceptor_vat_number_c);`	Value added tax (VAT) number for the card acceptor location Used to identify the card acceptor when collecting and reporting taxes
C	Tax	Up to 6 arrays	`mcCorpac.setTax(tax_c);`	Can have up to 6 arrays containing different tax details **NOTE:** If you use this variable, you must fill in all the fields of tax array mentioned below.

7.3.8.2 MC Corpal - Line Item Details

MC Corpal Object - Line Item Details

```
mcCorpal.setMcCorpal(customer_code1_l[0], line_item_date_l[0], ship_date_l[0],
order_date1_l[0], medical_services_ship_to_health_industry_number_l[0],
contract_number_l[0],medical_services_adjustment_l[0], medical_services_
product_number_qualifier_l[0], product_code1_l[0], item_description_l[0],
item_quantity_l[0], unit_cost_l[0], item_unit_measure_l[0], ext_item_amount_l
[0], discount_amount_l[0], commodity_code_l[0], type_of_supply_l[0], vat_ref_
num_l[0], tax_l[0]);
```

Table 1 Line Item Details - Level 3 Request Fields - MC Corpal

Req*	Value	Limits	Variable	Description
N	Customer Code	25-character alpha-numeric	`customer_code1_l`	A control number, such as purchase order number, project number, department allocation number or name that the purchaser supplied the merchant
N	Line Item Date	6-character numeric YYMMDD format	`line_item_date_l`	The purchase date of the line item referenced in the associated Corporate Card Line Item Detail Fixed length 6 Numeric, in YYMMDD format
N	Ship Date	6-character numeric YYMMDD format	`ship_date_l`	The date the merchandise was shipped to the destination Fixed length 6 Numeric, in YYMMDD format
N	Order Date	6-character numeric YYMMDD format	`order_date1_ll`	The date the item was ordered Fixed length 6-character numeric, in YYMMDD format
Y	Product Code	12-character alpha-numeric	`product_code1_ll`	Line item Product Code

Req*	Value	Limits	Variable	Description
				Contains the non-fuel related product code of the individual item purchased
Y	Item Description	35-character alpha-numeric	`item_description_ll`	Line Item description Contains the description of the individual item purchased
Y	Item Quantity	12-character alpha-numeric	`item_quantity_ll`	Quantity of line item Up to 5 decimal places supported Minimum amount is 0.0 and maximum is 9999999.99999
Y	Unit Cost	12-character decimal	`unit_cost_ll`	Line item cost per unit. Must contain a minimum of 2 decimal places, up to 5 decimal places supported. Minimum amount is 0.00001 and maximum is 999999.99999
Y	Item Unit Measure	12-character alpha-numeric	`item_unit_measure_ll`	The line item unit of measurement code ANSI X-12 EDI Allowable Units of Measure and Codes

Req*	Value	Limits	Variable	Description
Y	Extended Item Amount	9-character decimal	`ext_item_amount_ll`	Contains the individual item amount that is normally calculated as price multiplied by quantity Must contain 2 decimal places Minimum amount is 0.00 and maximum is 999999.99
N	Discount Amount	9-character decimal	`discount_amount_ll`	Contains the item discount amount Must contain 2 decimal places Minimum amount is 0.00 and maximum is 999999.99
N	Commodity Code	15-character alphanumeric	`commodity_code_ll`	Code assigned to the merchant that best categorizes the item(s) being purchased
C	Tax	Up to 6 arrays	`tax_l`	Can have up to 6 arrays containing different tax details –see Tax Array Request Fields table below for each field description **NOTE:** If you use this variable, you must fill in all the fields of tax array mentioned below.

7.3.8.3 Tax Array Object - MC Corpais

The tax array object is used when you use the Tax field of both MC Corpac and MC Corpal. If you use the tax array object, all of the array fields must be sent.

Setting the tax array differs slightly between the two objects.

Setting tax array for MC Corpac

```
//Tax Details

String[] tax_amount_c = { "1.19", "1.29"};

String[] tax_rate_c = { "6.0", "7.0"};

String[] tax_type_c = { "GST", "PST"};

String[] tax_id_c = { "gst1298", "pst1298"};

String[] tax_included_in_sales_c = { "Y", "N"};

McTax tax_c = new McTax();

tax_c.setTax(tax_amount_c[0], tax_rate_c[0], tax_type_c[0], tax_id_c[0], tax_
included_in_sales_c[0]);
```

Setting tax array for MC Corpal

```
//Tax Details for Items

String[] tax_amount_l = {"0.52", "1.48"};

String[] tax_rate_l = {"13.0", "13.0"};

String[] tax_type_l = {"HST", "HST"};

String[] tax_id_l = {"hst1298", "hst1298"};

String[] tax_included_in_sales_l = {"Y", "Y"};

McTax[] tax_l = new McTax[2];

tax_l[1].setTax(tax_amount_l[1], tax_rate_l[1], tax_type_l[1], tax_id_l[1],
tax_included_in_sales_l[1]);
```

Table 1 MC Corpais Tax Array Request Fields

Req*	Value	Limits	Variable	Description
Y	Tax Amount	12-character decimal	`tax_amount_c/tax_amount_l`	Contains detail tax amount for purchase of goods or services Must be 2 decimal places. Minimum amount is 0.00 and

Req*	Value	Limits	Variable	Description
				maximum is 999999.99
Y	Tax Rate	5-character decimal	`tax_rate_c/tax_rate_l`	Contains the detailed tax rate applied in relationship to a specific tax amount **EXAMPLE:** 5% GST should be '5.0' or or 9.975% QST should be '9.975' May contain up to 3 decimals, minimum 0.001, maximum up to 9999.9
Y	Tax Type	4-character alphanumeric	`tax_type_c/tax_type_l`	Contains tax type, such as GST,QST,PST,HST
Y	Tax ID	20-character alphanumeric	`tax_id_c/tax_id_l`	Provides an identification number used by the card acceptor with the tax authority in relationship to a specific tax amount, such as GST/HST number
Y	Tax included in sales indicator	1-character alphanumeric	`tax_included_in_sales_c/tax_included_in_sales_l`	This is the indicator used to reflect additional tax capture and reporting Valid values are: Y = Tax included in total purchase amount N = Tax not included in total purchase amount

7.3.8.4 Sample Code for MC Corpais

Sample MC Corpais - Corporate Card Common Data with Line Item Details

```
package Level23;
import JavaAPI.*;
public class TestMcCorpaisCommonLineItem
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485206444761";
String txn_number="39777-1_11";
String customer_code1_c ="CustomerCode123";
String card_acceptor_tax_id_c ="UrTaxId";//Merchant tax id which is mandatory
String corporation_vat_number_c ="cvn123";
String freight_amount_c ="1.23";
String duty_amount_c ="2.34";
String ship_to_pos_code_c ="M1R 1W5";
String order_date_c ="141211";
String customer_vat_number_c ="customervn231";
String unique_invoice_number_c ="uin567";
String authorized_contact_name_c ="John Walker";
//Tax Details
String[] tax_amount_c = { "1.19", "1.29"};
String[] tax_rate_c = { "6.0", "7.0"};
String[] tax_type_c = { "GST", "PST"};
String[] tax_id_c = { "gst1298", "pst1298"};
String[] tax_included_in_sales_c = { "Y", "N"};
//Item Details
String[] customer_code1_l = {"customer code", "customer code2"};
String[] line_item_date_l = {"150114", "150114"};
String[] ship_date_l = {"150120", "150122"};
String[] order_date1_l = {"150114", "150114"};
String[] medical_services_ship_to_health_industry_number_l = {"", ""};
String[] contract_number_l = {"", ""};
String[] medical_services_adjustment_l = {"", ""};
String[] medical_services_product_number_qualifier_l = {"", ""};
String[] product_code1_l = {"pc11", "pc12"};
String[] item_description_l = {"Good item", "Better item"};
String[] item_quantity_l = {"4", "5"};
String[] unit_cost_l ={"1.25", "10.00"};
String[] item_unit_measure_l = {"EA", "EA"};
String[] ext_item_amount_l ={"5.00", "50.00"};
String[] discount_amount_l ={"1.00", "50.00"};
String[] commodity_code_l ={"cCode11", "cCode12"};
String[] type_of_supply_l = {"", ""};
String[] vat_ref_num_l = {"", ""};
//Tax Details for Items
String[] tax_amount_l = {"0.52", "1.48"};
String[] tax_rate_l = {"13.0", "13.0"};
String[] tax_type_l = {"HST", "HST"};
String[] tax_id_l = {"hst1298", "hst1298"};
String[] tax_included_in_sales_l = {"Y", "Y"};
//Create and set Tax for McCorpac
McTax tax_c = new McTax();
tax_c.setTax(tax_amount_c[0], tax_rate_c[0], tax_type_c[0], tax_id_c[0], tax_included_in_sales_c
```

Sample MC Corpais - Corporate Card Common Data with Line Item Details

```
    [0]);
    tax_c.setTax(tax_amount_c[1], tax_rate_c[1], tax_type_c[1], tax_id_c[1], tax_included_in_sales_c
    [1]);
    //Create and set McCorpac for common data - only set values that you know
    McCorpac mcCorpac = new McCorpac();
    mcCorpac.setCustomerCode1(customer_code1_c);
    mcCorpac.setCardAcceptorTaxTd(card_acceptor_tax_id_c);
    mcCorpac.setCorporationVatNumber(corporation_vat_number_c);
    mcCorpac.setFreightAmount1(freight_amount_c);
    mcCorpac.setDutyAmount1(duty_amount_c);
    mcCorpac.setShipToPosCode(ship_to_pos_code_c);
    mcCorpac.setOrderDate(order_date_c);
    mcCorpac.setCustomerVatNumber(customer_vat_number_c);
    mcCorpac.setUniqueInvoiceNumber(unique_invoice_number_c);
    mcCorpac.setAuthorizedContactName(authorized_contact_name_c);
    mcCorpac.setTax(tax_c);
    //Create and set Tax for McCorpal
    McTax[] tax_l = new McTax[2];
    tax_l[0] = new McTax();
    tax_l[0].setTax(tax_amount_l[0], tax_rate_l[0], tax_type_l[0], tax_id_l[0], tax_included_in_sales_
    l[0]);
    tax_l[1] = new McTax();
    tax_l[1].setTax(tax_amount_l[1], tax_rate_l[1], tax_type_l[1], tax_id_l[1], tax_included_in_sales_
    l[1]);
    //Create and set McCorpal for each item
    McCorpal mcCorpal = new McCorpal();
    mcCorpal.setMcCorpal(customer_code1_l[0], line_item_date_l[0], ship_date_l[0], order_date1_l[0],
    medical_services_ship_to_health_industry_number_l[0], contract_number_l[0],
    medical_services_adjustment_l[0], medical_services_product_number_qualifier_l[0], product_code1_l
    [0], item_description_l[0], item_quantity_l[0],
    unit_cost_l[0], item_unit_measure_l[0], ext_item_amount_l[0], discount_amount_l[0], commodity_
    code_l[0], type_of_supply_l[0], vat_ref_num_l[0], tax_l[0]);
    mcCorpal.setMcCorpal(customer_code1_l[1], line_item_date_l[1], ship_date_l[1], order_date1_l[1],
    medical_services_ship_to_health_industry_number_l[1], contract_number_l[1],
    medical_services_adjustment_l[1], medical_services_product_number_qualifier_l[1], product_code1_l
    [1], item_description_l[1], item_quantity_l[1],
    unit_cost_l[1], item_unit_measure_l[1], ext_item_amount_l[1], discount_amount_l[1], commodity_
    code_l[1], type_of_supply_l[1], vat_ref_num_l[1], tax_l[1]);
    McCorpais mcCorpais = new McCorpais();
    mcCorpais.setOrderId(order_id);
    mcCorpais.setTxnNumber(txn_number);
    mcCorpais.setMcCorpac(mcCorpac);
    mcCorpais.setMcCorpal(mcCorpal);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(mcCorpais);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
```

Sample MC Corpais - Corporate Card Common Data with Line Item Details

```
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.4 Level 2/3 American Express Transactions

7.4.1 Level 2/3 Transaction Types for Amex

This transaction set includes a suite of corporate card financial transactions as well as a transaction that allows for the passing of Level 2/3 data. Please ensure American Express Level 2/3 processing support is enabled on your merchant account. Batch Close, Open Totals and Pre-authorization are identical to the transactions outlined in the section Basic Transaction Set (page 12).

- When the Pre-authorization response contains CorporateCard equal to true then you can submit the AX transactions.
- If CorporateCard is false then the card does not support Level 2/3 data and non Level 2/3 transaction are to be used. If the card is not a corporate card, please refer to 2 Basic Transaction Set for the appropriate non-corporate card transactions.

NOTE: This transaction set is intended for transactions where Corporate Card is true and Level 2/3 data will be submitted. If the credit card is found to be a corporate card but you do not wish to send any Level 2/3 data then you may submit AX transactions using the transaction set outlined in the section Basic Transaction Set (page 12).

Pre-authorization – (authorization)
The preauth verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from a pre-auth so that they may be settled in the merchant account a capture must be performed. CorporateCard will return as true if the card supports Level 2/3.

AX Completion – (Capture/Pre-authorization Completion)
Once a Pre-authorization is obtained the funds that are locked need to be retrieved from the customer's credit card. The capture retrieves the locked funds and readies them for settlement in to the merchant account. Prior to performing an AXCompletion a Preauth must be performed.

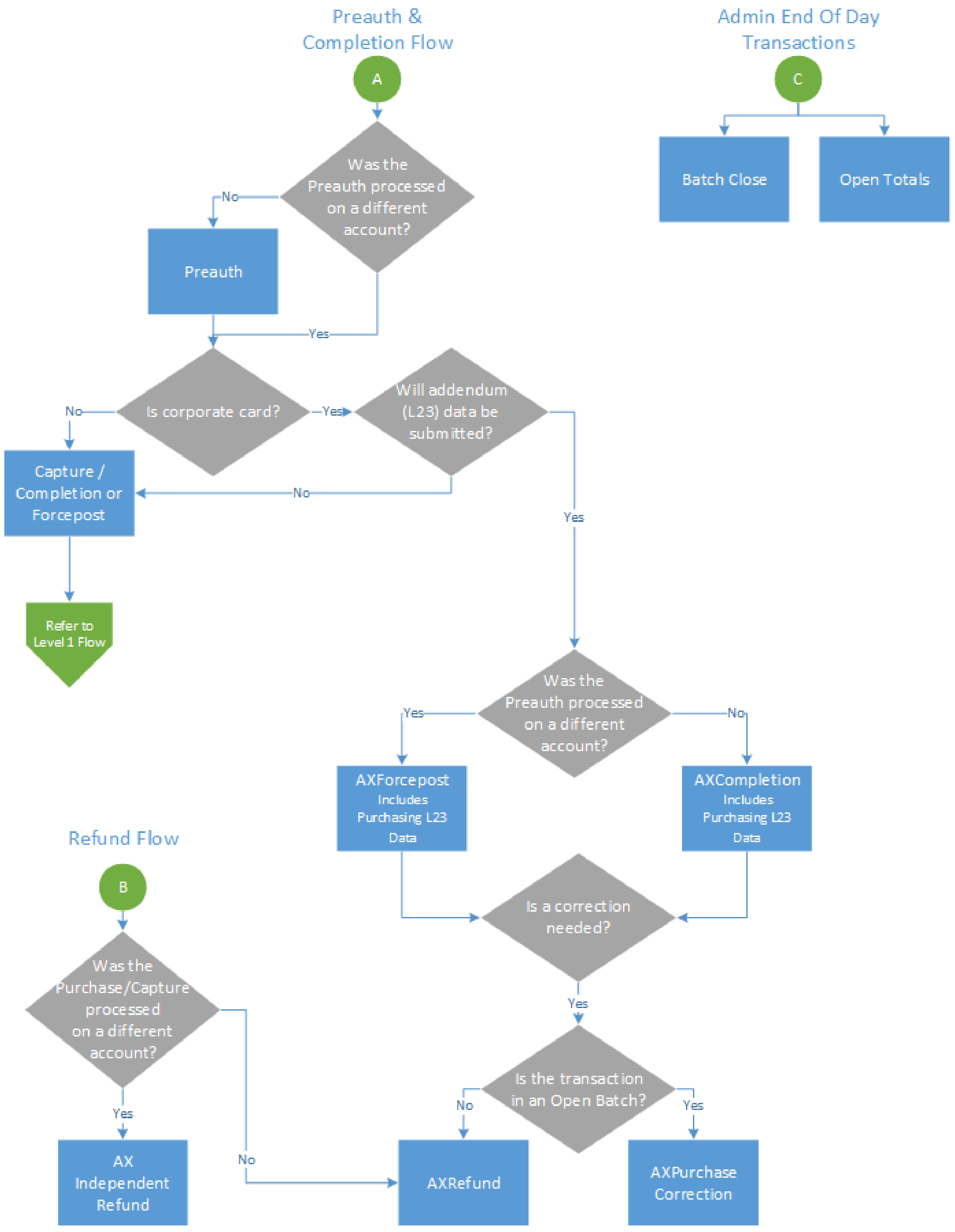
AX Force Post – (Force Capture/Pre-authorization Completion)
This transaction is an alternative to AX Completion to obtain the funds locked on a Pre-authorization obtained from IVR or equivalent terminal. The capture retrieves the locked funds and readies them for settlement in to the merchant account.

AX Purchase Correction – (Void, Correction)
AX Completion and AX Force Post can be voided the same day* that they occur. A void must be for the full amount of the transaction and will remove any record of it from the cardholder statement. * An AX Purchase Correction can be performed against a transaction as long as the batch that contains the original transaction remains open. When using the automated closing feature, the batch close occurs daily between 10 – 11 pm EST.

AX Refund – (Credit)
An AX Refund can be performed against an AX Completion and AX Force Post to refund any part, or all of the transaction.

AX Independent Refund – (Credit)
An AX Independent Refund can be performed against a purchase or a capture to refund any part, or all of the transaction. Independent refund is used when the originating transaction was not performed through Moneris Gateway. Please note, the Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an independent refund, it may mean the transaction is not supported on your account. If you wish to have the AX Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-866-319-7450.

7.4.2 Level 2/3 Transaction Flow for Amex


Preauth & Completion Flow
A
Was the Preauth processed on a different account?
No
Preauth
Yes
Is corporate card?
No
Yes
Will addendum (L23) data be submitted?
Capture / Completion or Forcepost
No
Yes
Refer to Level 1 Flow
Admin End Of Day Transactions
C
Batch Close
Open Totals
Was the Preauth processed on a different account?
Yes
No
AXForcepost Includes Purchasing L23 Data
AXCompletion Includes Purchasing L23 Data
Is a correction needed?
Yes
Is the transaction in an Open Batch?
No
Yes
Refund Flow
B
Was the Purchase/Capture processed on a different account?
Yes
No
AX Independent Refund
AXRefund
AXPurchase Correction

7.4.3 Level 2/3 Data Objects in Amex

7.4.3.1 About the Level 2/3 Data Objects for Amex

Many of the Level 2/3 transaction requests using American Express also include a mandatory data object called AxLevel23. AxLevel23 is also comprised of other objects, also described in this section.

The Level 2/3 data objects within this section apply to all of the following transactions and are passed as part of the transaction request for:

- AX Completion
- AX Force Post
- AX Refund
- AX Independent Refund

Things to Consider:

- Please ensure the addendum data below is complete and accurate.
- Please ensure the math on quantities calculations, amounts, discounts, taxes, etc. properly adds up to the overall transaction amount. Incorrect amounts will cause the transaction to be rejected.

7.4.3.2 Defining the AxLevel23 Object

AxLevel23 object definition

```
AxLevel23 level23 = new AxLevel23();
```

The AXLevel23 object itself has three objects, Table1, Table2 and Table3, all of which are mandatory.

Table 1 AxLevel23 Object

Req*	Value	Limits	Set Method	Description
Y	Table1	Object	`AxTable1 table1 = new AxTable1();` `level23.setTable1`	Refer below for further breakdown and definition of table1

Req*	Value	Limits	Set Method	Description
			`(table1);`	
Y	Table2	Object	`AxTable2 table2 = new AxTable2();` `level23.setTable2 (table2);`	Refer below for further breakdown and definition of table2
Y	Table3	Object	`AxTable3 table3 = new AxTable3();` `level23.setTable3 (table3);`	Refer below for further breakdown and definition of table3

*Y = Required, N = Optional, C = Conditional

Table 1 Object

Table 1 contains the addendum data heading information. Contains information such as identification elements that uniquely identify an invoice (transaction), the customer name and shipping address.

Table 1 object definition

```
AxTable1 table1 = new AxTable1();
```

Table 1 AxLevel23 object - Table 1 object fields

Req*	Value	Limits	Set Method	Description
C	Purchase Order Number	22-character alphanumeric	`table1.setBig04 (big04);`	The cardholder supplied Purchase Order Number, which is entered by the merchant at the point-of-sale This entry is used in the Statement/Reporting process and may include accounting information specific to the client **NOTE:** This element is mandatory, if the merchant's customer provides a Purchase Order Number.

Req*	Value	Limits	Set Method	Description
N	Release Number	30-character alpha-numeric	`table1.setBig05 (big05);`	A number that identifies a release against a Purchase Order previously placed by the parties involved in the transaction
N	Invoice Number	8-character alpha-numeric	`table1.setBig10 (big10);`	Contains the Amex invoice/reference number
N	N1Loop	Object	`table1.setN1Loop (n1Loop)`	Refer below for further breakdown and definition of N1Loop object

*Y = Required, N = Optional, C = Conditional

Table 1 also has its own objects:

- N1Loop object
- AxRef object

Table 1 - Setting the N1Loop Object

The N1Loop data set contains the Requester names. It can also optionally contain the buying group, ship from, ship to and receiver details.

A minimum of at least 1 n1Loop must be set. Up to 5 n1Loop can be set.

N1Loop object definition

```
n1Loop.setN1Loop(n101, n102, n301, n401, n402, n403, axRef1);
```

Table 1 AxLevel23 object - Table 1 object - N1Loop object fields

Req*	Value	Limits	Variable or Set Method	Description
Y	Entity Identifier Code	2-character alpha-numeric	`n101`	Supported values: R6 - Requester (required) BG - Buying Group (optional) SF - Ship From (optional) ST - Ship To (optional)

Req*	Value	Limits	Variable or Set Method	Description
				40 - Receiver (optional)
Y	Name	40-character alpha-numeric	`n102`	**n101 code** / **n102 meaning**: R6 – Requester Name; BG – Buying Group Name; SF – Ship From Name; ST – Ship To Name; 40 – Receiver Name
N	Address	40-character alpha-numeric	`n301`	Address
N	City	30-character alpha-numeric	`n401`	City
N	State or Province	2-character alpha-numeric	`n402`	State or province
N	Postal Code	15-character alpha-numeric	`n403`	Postal Code
N	AxRef	Object	`AxRef axRef1 = new AxRef();`	Refer below for further breakdown and definition of AxRef object. This object contains the customer postal code (mandatory) and customer reference number (optional) A minimum of 1 axRef1 must be set; maximum of 2 axRef1's may be set

*Y = Required, N = Optional, C = Conditional

Table 1 - Setting the AxRef Object

Setting AXRef object

```
AxRef axRef1 = new AxRef();

String[] ref01 = {"4C", "CR"}; //Reference ID Qualifier

String[] ref02 = {"M5T3A5", "16802309004"}; //Reference ID

axRef1.setRef(ref01[0], ref02[0]);

axRef1.setRef(ref01[1], ref02[1]);
```

Table 1 AxLevel23 object - Table 1 object - AxRef object fields

Req*	Value	Limits	Variable	Description
Y	Reference Identification Qualifier	2-character alpha-numeric	ref01	This element may contain the following qualifiers for the corresponding occurrences of the N1Loop:
Y	Reference Identification	15-character alpha-numeric	ref02	This field must be populated for each ref01 provided

n101 value	ref01 denotation
R6	Supported values: 4C - Shipment Destination Code (mandatory) CR - Customer Reference Number (conditional)
BG	n/a
SF	n/a
ST	n/a
40	n/a

Req*	Value	Limits	Variable	Description
				ref01 value — **ref02 denotation** 4C (n101 value = R6) — This element must contain the Amex Ship-to Postal Code of the destination where the commodity was shipped. If the Ship-to Postal Code is unavailable, the postal code of the merchant location where the transaction took place may be substituted. CR (n101 value = R6): — This element must contain the Amex Card member Reference Number (e.g., purchase order, cost center, project number, etc.) that corresponds to this transaction, if provided by the Cardholder. This information may be displayed in the statement/reporting process and may include client-specific accounting information.

*Y = Required, N = Optional, C = Conditional

Table 2 Object

Table 2 includes the transaction's addendum detail. It contains transaction data including reference codes, debit or credit and tax amounts, line item detail descriptions, shipping information and much more. All transaction data in an invoice relate to a single transaction and cardholder account number.

Table 2 object definition

```
AxTable2 table2 = new AxTable2();
```

Table 1 AxLevel23 object - Table 2 object fields

Req*	Value	Limits	Set Method	Description
N	It1loop	Object	`table2.setIt1Loop (it1Loop);`	Refer below for further breakdown and definition of object details.

*Y = Required, N = Optional, C = Conditional

Table 2 - Setting the AxIt1Loop Object

The AxIt1Loop data defines the baseline item data for the invoice. This data is defined for each item/service purchased and included within this invoice. This data set contains basic transaction data, including quantity, unit of measure, unit price and goods/services reference information.

- A minimum of 1 it1Loop required
- A maximum of 999 it1Loop's supported

AxIt1Loop object definition

```
AxIt1Loop it1Loop = new AxIt1Loop();

it1Loop.setIt1Loop(it102[0], it103[0], it104[0], it105[0], it106s[0], txi[0],
pam05[0], pid05[0]);

it1Loop.setIt1Loop(it102[1], it103[1], it104[1], it105[1], it106s[1], txi[1],
pam05[1], pid05[1]);
```

Table 1 AxLevel23 object - Table 2 object - AxIt1Loop object fields

Req*	Value	Limits	Variable	Description
Y	Line Item Quantity Invoiced	10-character decimal	it102	Quantity of line item Up to 2 decimal places supported Minimum amount is 0.0 and maximum is 9999999999
Y	Unit or Basis for Measurement Code	2-character alphanumeric	it103	The line item unit of measurement code Must contain a

Req*	Value	Limits	Variable	Description
				code that specifies the units in which the value is expressed or the manner in which a measurement is taken **EXAMPLE:** EA = each, E5=inches See ANSI X-12 EDI Allowable Units of Measure and Codes for the list of codes
Y	Unit Price	15-character decimal	it104	Line item cost per unit Must contain 2 decimal places Minimum amount is 0.00 and max-imum is 999999.99
N	Basis or Unit Price Code	2-character alpha-numeric	it105	Code identifying the type of unit price for an item **EXAMPLE:** DR = dealer, AP = advise price See ASC X12 004010 Element 639 for list of codes
N	AxIt106s	object	it106s	Refer below for fur-ther breakdown and definition of object details.
N	AxTxi	object	txi	Refer below for fur-ther breakdown and definition of

Req*	Value	Limits	Variable	Description
				object details A maximum of 12 AxTxi (tax information data sets) may be defined **NOTE:** that if line item level tax information is populated in AxTxi in Table2, then tax totals for the entire invoice (transaction) must be entered in Table3.
Y	Line Item Extended Amount	8-character decimal	pam05	Contains the individual item amount that is normally calculated as price multiplied by quantity Must contain 2 decimal places Minimum amount is 0.00 and maximum is 99999.99
Y	Line Item Description	80-character alphanumeric	pid05	Line Item description Contains the description of the individual item purchased This field pertain to each line item in the transaction

*Y = Required, N = Optional, C = Conditional

Table 2 - Setting the AxIt106s Object

```
AxIt106s[] it106s = {new AxIt106s(), new AxIt106s(), new AxIt106s(), new
AxIt106s(), new AxIt106s()};

String[] it10618 = {"MG", "MG", "MG", "MG", "MG"}; //Product/Service ID
qualifier

String[] it10719 = {"DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"};
//Product/Service ID (corresponds to it10618)
```

Table 1 AxLevel23 object - Table 2 object - AxIt106s object fields

Req*	Value	Limits	Set Method	Description

N	Product/Service ID Qualifier	2-character alpha-numeric	`it106s [0].setIt10618 (it10618[0]);`
			
			`it106s [1].setIt10618 (it10618[1]);`
				Supported values:
				
				MG - Manufacturer's Part Number
				
				VC - Supplier Catalog Number
				
				SK - Supplier Stock Keeping Unit Number
				
				UP - Universal Product Code
				
				VP – Vendor Part Num-ber
				
				PO – Purchase Order Number
				
				AN – Client Defined Asset Code

N	Product/Service ID	**it10618** — **it10719 - size/type**
		VC — 20-character alphanumeric
		PO — 22-character alphanumeric
		Other — 30-character alphanumeric
			`it106s [0].setIt10719 (it10719[0]);`
			
			`it106s [1].setIt10719 (it10719[1]);`
				Product/Service ID corresponds to the preceding qualifier defined by it10618
				
				The maximum length depends on the qualifier defined in it10618

*Y = Required, N = Optional, C = Conditional

Table 2 - Setting the AxTxi Object

Table 2 AxiTxi object definition

```
//Create Table 2 with details

String[] txi01_GST = {"GS", "GS", "GS", "GS", "GS"}; //Tax type code
```

```
String[] txi02_GST = {"0.70", "1.75", "1.00", "0.80","0.00"}; //Monetary
amount

String[] txi03_GST = {"5.0", "5.0", "5.0", "5.0","5.0"}; //Percent

String[] txi06_GST = {"2", "2", "2", "2","2"}; //Tax exempt code

String[] txi01_PST = {"PG", "PG", "PG","PG","PG"}; //Tax type code

String[] txi02_PST = {"0.80", "2.00", "1.00", "0.80","0.00"}; //Monetary
amount

String[] txi03_PST = {"7.0", "7.0", "7.0", "7.0","7.0"}; //Percent

String[] txi06_PST = {"2", "2", "2", "2","2"}; //Tax exempt code


AxTxi[] txi = {new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi
()};

txi[0].setTxi(txi01_GST[0], txi02_GST[0], txi03_GST[0], txi06_GST[0]);

txi[0].setTxi(txi01_PST[0], txi02_PST[0], txi03_PST[0], txi06_PST[0]);

txi[1].setTxi(txi01_GST[1], txi02_GST[1], txi03_GST[1], txi06_GST[1]);

txi[1].setTxi(txi01_PST[1], txi02_PST[1], txi03_PST[1], txi06_PST[1]);

txi[2].setTxi(txi01_GST[2], txi02_GST[2], txi03_GST[2], txi06_GST[2]);

txi[2].setTxi(txi01_PST[2], txi02_PST[2], txi03_PST[2], txi06_PST[2]);

txi[3].setTxi(txi01_GST[3], txi02_GST[3], txi03_GST[3], txi06_GST[3]);

txi[3].setTxi(txi01_PST[3], txi02_PST[3], txi03_PST[3], txi06_PST[3]);

txi[4].setTxi(txi01_GST[4], txi02_GST[4], txi03_GST[4], txi06_GST[4]);

txi[4].setTxi(txi01_PST[4], txi02_PST[4], txi03_PST[4], txi06_PST[4]);
```

Table 1 AxLevel23 object - Table 2 object - AxiTxi object fields

Req*	Value	Limits	Variable	Description
C	Tax Type code	txi01	2-character alphanumeric	Tax type code applicable to Canada and US only For Canada, this field must contain a code that specifies the type of tax If txi01 is used,

Req*	Value	Limits	Variable	Description
				then txi02, txi03 or txi06 must be populated Valid codes include the following: CT – County/Tax (optional) CA – City Tax (optional) EV – Environmental Tax (optional) GS – Good and Services Tax (GST) (optional) LS – State and Local Sales Tax (optional) LT – Local Sales Tax (optional) PG – Provincial Sales Tax (PST) (optional) SP – State/Provincial Tax a.k.a. Quebec Sales Tax (QST) (optional) ST – State Sales Tax (optional) TX – All Taxes (required) VA – Value-Added Tax a.k.a. Canadian Harmonized Sales Tax (HST) (optional)
C	Monetary Amount	txi02	6-character decimal	This element may contain the monetary tax amount that corresponds to the Tax Type Code in txi01 **NOTE:** If txi02 is used in mandatory occur-

Req*	Value	Limits	Variable	Description
				rence txi01=TX, txi02 must contain the total tax amount applicable to the entire invoice (transaction) If taxes are not applicable for the entire invoice (transaction), txi02 must be 0.00. The maximum value that can be entered in this field is “9999.99”, which is $9,999.99 (CAD) A debit is entered as: 9999.99 A credit is entered as: –9999.99
C	Percent	txi03	10-character decimal	Contains the tax percentage (in decimal format) that corresponds to the tax type code defined in txi01 Up to 2 decimal places supported
C	Tax Exempt Code	txi06	1-character alphanumeric	This element may contain the Tax Exempt Code that identifies the exemption status from sales and tax that corresponds to the Tax Type Code in txi01 Supported values: 1 – Yes (Tax Exempt) 2 – No (Not Tax Exempt)

Req*	Value	Limits	Variable	Description
				4 – Not Exempt/For Resale A – Labor Taxable, Material Exempt B – Material Taxable, Labor Exempt C – Not Taxable F – Exempt (Goods / Services Tax) G – Exempt (Provincial Sales Tax) L – Exempt Local Service R – Recurring Exempt U – Usage Exempt

*Y = Required, N = Optional, C = Conditional

Table 3 Object

Table 3 includes the transaction addendum summary. It contains the total invoice (transaction) amount, sales tax, freight and/or handling charges and invoice summary information, including total line items, number of segments in the invoice, and the transaction set control number (a.k.a., batch number).

Table 3 object definition

```
AxTable3 table3 = new AxTable3();
```

Table 1 AxLevel23 object - Table 3 object fields

Req*	Value	Limits	Set Method	Description
C	AxTxi	Object	`table3.setTxi (taxTbl3);`	Refer below for further breakdown and definition of object details. **NOTE:** if line item level tax information is populated in AxTxi in Table2, then tax totals for the entire invoice

Req*	Value	Limits	Set Method	Description
				(transaction) must be entered in Table3. A maximum of 10 AxTxi's may be set in Table3.

*Y = Required, N = Optional, C = Conditional

Table 3 - Setting the AxTxi Object

The mandatory tax information data set must contain the total tax amount applicable to the entire invoice (transaction) which includes all line items identified in Table2. If taxes are not applicable for the entire invoice (transaction), then txi02 must be set to 0.00.

Tax totals must be entered in this mandatory tax information segment in Table 3, even if line item detail level tax data is reported in Table 2.

At least one occurrence of txi02, txi03 or txi06 is required.

Table 3 AxiTxi object definition

```
AxTxi taxTbl3 = new AxTxi();

taxTbl3.setTxi("GS", "4.25","",""); //sum of GST taxes

taxTbl3.setTxi("PG", "4.60","",""); //sum of PST taxes

taxTbl3.setTxi("TX", "8.85","",""); //sum of all taxes
```

Table 1 AxLevel23 object - Table 3 object - AxiTxi object fields

Req*	Value	Limits	Variable	Description
C	Tax Type code	txi01	2-character alphanumeric	Tax type code applicable to Canada and US only For Canada, this field must contain a code that specifies the type of tax If txi01 is used, then txi02, txi03 or txi06 must be populated

Req*	Value	Limits	Variable	Description
				Valid codes include the following: CT – County/Tax (optional) CA – City Tax (optional) EV – Environmental Tax (optional) GS – Good and Services Tax (GST) (optional) LS – State and Local Sales Tax (optional) LT – Local Sales Tax (optional) PG – Provincial Sales Tax (PST) (optional) SP – State/Provincial Tax a.k.a. Quebec Sales Tax (QST) (optional) ST – State Sales Tax (optional) TX – All Taxes (required) VA – Value-Added Tax a.k.a. Canadian Harmonized Sales Tax (HST) (optional)
C	Monetary Amount	txi02	6-character decimal	This element may contain the monetary tax amount that corresponds to the Tax Type Code in txi01 **NOTE:** If txi02 is used in mandatory occurrence txi01=TX, txi02 must contain the total tax amount applicable to the entire invoice (transaction)

Req*	Value	Limits	Variable	Description
				If taxes are not applicable for the entire invoice (transaction), txi02 must be 0.00. The maximum value that can be entered in this field is “9999.99”, which is $9,999.99 (CAD) A debit is entered as: 9999.99 A credit is entered as: –9999.99
C	Percent	txi03	10-character decimal	Contains the tax percentage (in decimal format) that corresponds to the tax type code defined in txi01 Up to 2 decimal places supported
C	Tax Exempt Code	txi06	1-character alphanumeric	This element may contain the Tax Exempt Code that identifies the exemption status from sales and tax that corresponds to the Tax Type Code in txi01 Supported values: 1 – Yes (Tax Exempt) 2 – No (Not Tax Exempt) 4 – Not Exempt/For Resale A – Labor Taxable, Material Exempt

Req*	Value	Limits	Variable	Description
				B – Material Taxable, Labor Exempt C – Not Taxable F – Exempt (Goods / Services Tax) G – Exempt (Provincial Sales Tax) L – Exempt Local Ser- vice R – Recurring Exempt U – Usage Exempt

*Y = Required, N = Optional, C = Conditional

7.4.4 AX Completion

The AX Completion transaction is used to secure the funds locked by a pre-authorization transaction. When sending a capture request you will need two pieces of information from the original pre-authorization – the Order ID and the transaction number from the returned response.

AX Completion transaction object definition

```
AxCompletion axCompletion = new AxCompletion()
```

HttpsPostRequest object for AX Completion

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(axCompletion);
```

AX Completion transaction object values

Table 1 AX Completion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`axCompletion.setOrderId (order_id);`
Completion amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`axCompletion.setCompAmount (comp_amount);`
Transaction number	String	255-character alpha-numeric	`axCompletion.setTxnNumber (txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`axCompletion.setCryptType (crypt);`
Level 2/3 Data	Object	n/a	`axCompletion.setAxLevel23 (level23);`

Sample AX Completion

```
package Level23;
import JavaAPI.*;

public class TestAxCompletion
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="ord-210916-12:06:38";
String comp_amount="62.37";
String txn_number = "18924-0_11";
String crypt="7";

//Create Table 1 with details
String n101 = "R6"; //Entity ID Code
String n102 = "Retailing Inc. International"; //Name
String n301 = "919 Oriole Rd."; //Address Line 1
```

Sample AX Completion

```
    String n401 = "Toronto"; //City
    String n402 = "On"; //State or Province
    String n403 = "H1T6W3"; //Postal Code
    String[] ref01 = {"4C", "CR"}; //Reference ID Qualifier
    String[] ref02 = {"M5T3A5", "16802309004"}; //Reference ID
    String big04 = "PO7758545"; //Purchase Order Number
    String big05 = "RN0049858"; //Release Number
    String big10 = "INV99870E"; //Invoice Number
    AxRef axRef1 = new AxRef();
    axRef1.setRef(ref01[0], ref02[0]);
    axRef1.setRef(ref01[1], ref02[1]);
    AxN1Loop n1Loop = new AxN1Loop();
    n1Loop.setN1Loop(n101, n102, n301, n401, n402, n403, axRef1);
    AxTable1 table1 = new AxTable1();
    table1.setBig04(big04);
    table1.setBig05(big05);
    table1.setBig10(big10);
    table1.setN1Loop(n1Loop);

    //Create Table 2 with details
    //the sum of the extended amount field (pam05) must equal the level 1 amount field
    String[] it102 = {"1", "1", "1", "1", "1"}; //Line item quantity invoiced
    String[] it103 = {"EA", "EA", "EA", "EA", "EA"}; //Line item unit or basis of measurement code
    String[] it104 = {"10.00", "25.00", "8.62", "10.00", "-10.00"}; //Line item unit price
    String[] it105 = {"", "", "", "", ""}; //Line item basis of unit price code

    String[] it10618 = {"MG", "MG", "MG", "MG", "MG"}; //Product/Service ID qualifier
    String[] it10719 = {"DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"}; //Product/Service ID
    (corresponds to it10618)

    String[] txi01_GST = {"GS", "GS", "GS", "GS", "GS"}; //Tax type code
    String[] txi02_GST = {"0.70", "1.75", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_GST = {"", "", "", "",""}; //Percent
    String[] txi06_GST = {"", "", "", "",""}; //Tax exempt code

    String[] txi01_PST = {"PG", "PG", "PG","PG","PG"}; //Tax type code
    String[] txi02_PST = {"0.80", "2.00", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_PST = {"", "", "", "",""}; //Percent
    String[] txi06_PST = {"", "", "", "",""}; //Tax exempt code
    String[] pam05 = {"11.50", "28.75", "10.62", "11.50", "-10.00"}; //Extended line-item amount
    String[] pid05 = {"Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"}; //Line item
    description
    AxIt106s[] it106s = {new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s
    ()};

    it106s[0].setIt10618(it10618[0]);
    it106s[0].setIt10719(it10719[0]);

    it106s[1].setIt10618(it10618[1]);
    it106s[1].setIt10719(it10719[1]);

    it106s[2].setIt10618(it10618[2]);
    it106s[2].setIt10719(it10719[2]);

    it106s[3].setIt10618(it10618[3]);
    it106s[3].setIt10719(it10719[3]);

    it106s[4].setIt10618(it10618[4]);
    it106s[4].setIt10719(it10719[4]);
```

Sample AX Completion

```
    AxTxi[] txi = {new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi()};
    txi[0].setTxi(txi01_GST[0], txi02_GST[0], txi03_GST[0], txi06_GST[0]);
    txi[0].setTxi(txi01_PST[0], txi02_PST[0], txi03_PST[0], txi06_PST[0]);
    txi[1].setTxi(txi01_GST[1], txi02_GST[1], txi03_GST[1], txi06_GST[1]);
    txi[1].setTxi(txi01_PST[1], txi02_PST[1], txi03_PST[1], txi06_PST[1]);
    txi[2].setTxi(txi01_GST[2], txi02_GST[2], txi03_GST[2], txi06_GST[2]);
    txi[2].setTxi(txi01_PST[2], txi02_PST[2], txi03_PST[2], txi06_PST[2]);
    txi[3].setTxi(txi01_GST[3], txi02_GST[3], txi03_GST[3], txi06_GST[3]);
    txi[3].setTxi(txi01_PST[3], txi02_PST[3], txi03_PST[3], txi06_PST[3]);
    txi[4].setTxi(txi01_GST[4], txi02_GST[4], txi03_GST[4], txi06_GST[4]);
    txi[4].setTxi(txi01_PST[4], txi02_PST[4], txi03_PST[4], txi06_PST[4]);
    AxIt1Loop it1Loop = new AxIt1Loop();
    it1Loop.setIt1Loop(it102[0], it103[0], it104[0], it105[0], it106s[0], txi[0], pam05[0], pid05[0]);
    it1Loop.setIt1Loop(it102[1], it103[1], it104[1], it105[1], it106s[1], txi[1], pam05[1], pid05[1]);
    it1Loop.setIt1Loop(it102[2], it103[2], it104[2], it105[2], it106s[2], txi[2], pam05[2], pid05[2]);
    it1Loop.setIt1Loop(it102[3], it103[3], it104[3], it105[3], it106s[3], txi[3], pam05[3], pid05[3]);
    it1Loop.setIt1Loop(it102[4], it103[4], it104[4], it105[4], it106s[4], txi[4], pam05[4], pid05[4]);
    AxTable2 table2 = new AxTable2();
    table2.setIt1Loop(it1Loop);
    //Create Table 3 with details
    AxTxi taxTbl3 = new AxTxi();
    taxTbl3.setTxi("GS", "4.25","",""); //sum of GST taxes
    taxTbl3.setTxi("PG", "4.60","",""); //sum of PST taxes
    taxTbl3.setTxi("TX", "8.85","",""); //sum of all taxes
    AxTable3 table3 = new AxTable3();
    table3.setTxi(taxTbl3);

    //Create and set Level23 Object
    AxLevel23 level23 = new AxLevel23();
    level23.setTable1(table1);
    level23.setTable2(table2);
    level23.setTable3(table3);
    AxCompletion axCompletion = new AxCompletion();
    axCompletion.setOrderId(order_id);
    axCompletion.setCompAmount(comp_amount);
    axCompletion.setTxnNumber(txn_number);
    axCompletion.setCryptType(crypt);
    axCompletion.setAxLevel23(level23);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(axCompletion);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
```

Sample AX Completion

```
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.4.5 AX Force Post

The AX Force Post transaction is used to secure the funds locked by a pre-authorization transaction performed over IVR or equivalent terminal. When sending an AX Force Post request, you will need the order ID, amount, credit card number, expiry date, authorization code and e-commerce indicator.

AX Force Post transaction object definition

```
AxForcePost axForcePost = new AxForcePost();
```

HttpsPostRequest object for AX Force Post transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(axForcePost);
```

AX Force Post transaction object values

Table 1 AX Force Post transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`axForcePost.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`axForcePost.setAmount(amount);`
Credit card number	String	20-character alpha-numeric	`axForcePost.setPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`axForcePost.setExpDate(expiry_date);`
Authorization code	String	8-character alpha-numeric	`axForcePost.setAuthCode(auth_code);`
E-commerce indicator	String	1-character alpha-numeric	`axForcePost.setCryptType(crypt);`
Level 2/3 Data	Object	n/a	`axForcePost.setAxLevel23(level23);`

Table 2 AX Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`axForcePost.setCustId(cust_id);`

Sample AX Force Post

```
package Level23;
```

Sample AX Force Post

```
import JavaAPI.*;

public class TestAxForcePost
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id="CUST13343";
String amount="62.37";
String pan="373269005095005";
String expiry_date="2012"; //YYMM
String auth_code="123456";
String crypt="7";

//Create Table 1 with details
String n101 = "R6"; //Entity ID Code
String n102 = "Retailing Inc. International"; //Name
String n301 = "919 Oriole Rd."; //Address Line 1
String n401 = "Toronto"; //City
String n402 = "On"; //State or Province
String n403 = "H1T6W3"; //Postal Code
String[] ref01 = {"4C", "CR"}; //Reference ID Qualifier
String[] ref02 = {"M5T3A5", "16802309004"}; //Reference ID
String big04 = "PO7758545"; //Purchase Order Number
String big05 = "RN0049858"; //Release Number
String big10 = "INV99870E"; //Invoice Number
AxRef axRef1 = new AxRef();
axRef1.setRef(ref01[0], ref02[0]);
axRef1.setRef(ref01[1], ref02[1]);
AxN1Loop n1Loop = new AxN1Loop();
n1Loop.setN1Loop(n101, n102, n301, n401, n402, n403, axRef1);
AxTable1 table1 = new AxTable1();
table1.setBig04(big04);
table1.setBig05(big05);
table1.setBig10(big10);
table1.setN1Loop(n1Loop);

//Create Table 2 with details
//the sum of the extended amount field (pam05) must equal the level 1 amount field

String[] it102 = {"1", "1", "1", "1", "1"}; //Line item quantity invoiced
String[] it103 = {"EA", "EA", "EA", "EA", "EA"}; //Line item unit or basis of measurement code
String[] it104 = {"10.00", "25.00", "8.62", "10.00", "-10.00"}; //Line item unit price
String[] it105 = {"", "", "", "", ""}; //Line item basis of unit price code

String[] it10618 = {"MG", "MG", "MG", "MG", "MG"}; //Product/Service ID qualifier
String[] it10719 = {"DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"}; //Product/Service ID
(corresponds to it10618)

String[] txi01_GST = {"GS", "GS", "GS", "GS", "GS"}; //Tax type code
String[] txi02_GST = {"0.70", "1.75", "1.00", "0.80","0.00"}; //Monetary amount
String[] txi03_GST = {"", "", "", "",""}; //Percent
String[] txi06_GST = {"", "", "", "",""}; //Tax exempt code
```

Sample AX Force Post

```
String[] txi01_PST = {"PG", "PG", "PG","PG","PG"}; //Tax type code
String[] txi02_PST = {"0.80", "2.00", "1.00", "0.80","0.00"}; //Monetary amount
String[] txi03_PST = {"", "", "", "",""}; //Percent
String[] txi06_PST = {"", "", "", "",""}; //Tax exempt code
String[] pam05 = {"11.50", "28.75", "10.62", "11.50", "-10.00"}; //Extended line-item amount
String[] pid05 = {"Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"}; //Line item
description
AxIt106s[] it106s = {new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s
()};

it106s[0].setIt10618(it10618[0]);
it106s[0].setIt10719(it10719[0]);

it106s[1].setIt10618(it10618[1]);
it106s[1].setIt10719(it10719[1]);

it106s[2].setIt10618(it10618[2]);
it106s[2].setIt10719(it10719[2]);

it106s[3].setIt10618(it10618[3]);
it106s[3].setIt10719(it10719[3]);

it106s[4].setIt10618(it10618[4]);
it106s[4].setIt10719(it10719[4]);
AxTxi[] txi = {new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi()};
txi[0].setTxi(txi01_GST[0], txi02_GST[0], txi03_GST[0], txi06_GST[0]);
txi[0].setTxi(txi01_PST[0], txi02_PST[0], txi03_PST[0], txi06_PST[0]);
txi[1].setTxi(txi01_GST[1], txi02_GST[1], txi03_GST[1], txi06_GST[1]);
txi[1].setTxi(txi01_PST[1], txi02_PST[1], txi03_PST[1], txi06_PST[1]);
txi[2].setTxi(txi01_GST[2], txi02_GST[2], txi03_GST[2], txi06_GST[2]);
txi[2].setTxi(txi01_PST[2], txi02_PST[2], txi03_PST[2], txi06_PST[2]);
txi[3].setTxi(txi01_GST[3], txi02_GST[3], txi03_GST[3], txi06_GST[3]);
txi[3].setTxi(txi01_PST[3], txi02_PST[3], txi03_PST[3], txi06_PST[3]);
txi[4].setTxi(txi01_GST[4], txi02_GST[4], txi03_GST[4], txi06_GST[4]);
txi[4].setTxi(txi01_PST[4], txi02_PST[4], txi03_PST[4], txi06_PST[4]);
AxIt1Loop it1Loop = new AxIt1Loop();
it1Loop.setIt1Loop(it102[0], it103[0], it104[0], it105[0], it106s[0], txi[0], pam05[0], pid05[0]);
it1Loop.setIt1Loop(it102[1], it103[1], it104[1], it105[1], it106s[1], txi[1], pam05[1], pid05[1]);
it1Loop.setIt1Loop(it102[2], it103[2], it104[2], it105[2], it106s[2], txi[2], pam05[2], pid05[2]);
it1Loop.setIt1Loop(it102[3], it103[3], it104[3], it105[3], it106s[3], txi[3], pam05[3], pid05[3]);
it1Loop.setIt1Loop(it102[4], it103[4], it104[4], it105[4], it106s[4], txi[4], pam05[4], pid05[4]);
AxTable2 table2 = new AxTable2();
table2.setIt1Loop(it1Loop);
//Create Table 3 with details
AxTxi taxTbl3 = new AxTxi();
taxTbl3.setTxi("GS", "4.25","",""); //sum of GST taxes
taxTbl3.setTxi("PG", "4.60","",""); //sum of PST taxes
taxTbl3.setTxi("TX", "8.85","",""); //sum of all taxes
AxTable3 table3 = new AxTable3();
table3.setTxi(taxTbl3);

AxLevel23 level23 = new AxLevel23();
level23.setTable1(table1);
level23.setTable2(table2);
level23.setTable3(table3);
AxForcePost axForcePost = new AxForcePost();
axForcePost.setOrderId(order_id);
axForcePost.setCustId(cust_id);
```

Sample AX Force Post

```
axForcePost.setAmount(amount);
axForcePost.setPan(pan);
axForcePost.setExpDate(expiry_date);
axForcePost.setAuthCode(auth_code);
axForcePost.setCryptType(crypt);
axForcePost.setAxLevel23(level23);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(axForcePost);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();

System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.4.6 AX Purchase Correction

The AX Purchase Correction (Void) transaction is used to cancel a transaction that was performed in the current batch. No amount is required because a void is always for 100% of the original transaction. The only transaction that can be voided using AX Purchase Correction is AX Completion and AX Force Post. To send an AX Purchase Correction the Order ID and transaction number from the AX Completion or AX Force Post are required.

AX Purchase Correction transaction object definition

```
AxPurchaseCorrection axPurchaseCorrection = new AxPurchaseCorrection();
```

HttpsPostRequest object for AX Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(axPurchaseCorrection);
```

AX Purchase Correction transaction object values

Table 1 AX Purchase Correction transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`axPurchaseCorrection .setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`axPurchaseCorrection .setTxnNumber(txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`axPurchaseCorrection .setCryptType(crypt);`

AX Purchase Correction

```
package Level23;
import JavaAPI.*;

public class TestAxPurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485206180427";
String txn_number = "660117311902017023161620759-0_11";
String crypt="7";
AxPurchaseCorrection axPurchaseCorrection = new AxPurchaseCorrection();
axPurchaseCorrection.setOrderId(order_id);
axPurchaseCorrection.setTxnNumber(txn_number);
axPurchaseCorrection.setCryptType(crypt);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(axPurchaseCorrection);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
```

AX Purchase Correction

```
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

7.4.7 AX Refund

The AX Refund will credit a specified amount to the cardholder's credit card. A refund can be sent up to the full value of the original AX Completion or AX Force Post. To send an AX Refund you will require the Order ID and transaction number from the original AX Completion or AX Force Post.

AX Refund transaction object definition

```
AxRefund axRefund = new AxRefund();
```

HttpsPostRequest object for AX Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(axRefund);
```

AX Refund transaction object values

Table 1 AX Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`axRefund.setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`axRefund.setTxnNumber(txn_number);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`axRefund.setAmount(amount);`
E-commerce indicator	String	1-character alpha-numeric	`axRefund.setCryptType(crypt);`
Level 2/3 Data	Object	n/a	`axRefund.setAxLevel23(level23);`

Sample AX Refund

```
package Level23;
import JavaAPI.*;
public class TestAxRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

String order_id="Test1485206231878";
String amount="62.37";
String txn_number = "660117311902017023161712265-0_11";
String crypt="7";

//Create Table 1 with details
String n101 = "R6"; //Entity ID Code
String n102 = "Retailing Inc. International"; //Name
String n301 = "919 Oriole Rd."; //Address Line 1
String n401 = "Toronto"; //City
```

Sample AX Refund

```
    String n402 = "On"; //State or Province
    String n403 = "H1T6W3"; //Postal Code
    String[] ref01 = {"4C", "CR"}; //Reference ID Qualifier
    String[] ref02 = {"M5T3A5", "16802309004"}; //Reference ID
    String big04 = "PO7758545"; //Purchase Order Number
    String big05 = "RN0049858"; //Release Number
    String big10 = "INV99870E"; //Invoice Number
    AxRef axRef1 = new AxRef();
    axRef1.setRef(ref01[0], ref02[0]);
    axRef1.setRef(ref01[1], ref02[1]);
    AxN1Loop n1Loop = new AxN1Loop();
    n1Loop.setN1Loop(n101, n102, n301, n401, n402, n403, axRef1);
    AxTable1 table1 = new AxTable1();
    table1.setBig04(big04);
    table1.setBig05(big05);
    table1.setBig10(big10);
    table1.setN1Loop(n1Loop);

    //Create Table 2 with details
    //the sum of the extended amount field (pam05) must equal the level 1 amount field
    String[] it102 = {"1", "1", "1", "1", "1"}; //Line item quantity invoiced
    String[] it103 = {"EA", "EA", "EA", "EA", "EA"}; //Line item unit or basis of measurement code
    String[] it104 = {"10.00", "25.00", "8.62", "10.00", "-10.00"}; //Line item unit price
    String[] it105 = {"", "", "", "", ""}; //Line item basis of unit price code

    String[] it10618 = {"MG", "MG", "MG", "MG", "MG"}; //Product/Service ID qualifier
    String[] it10719 = {"DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"}; //Product/Service ID
    (corresponds to it10618)

    String[] txi01_GST = {"GS", "GS", "GS", "GS", "GS"}; //Tax type code
    String[] txi02_GST = {"0.70", "1.75", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_GST = {"", "", "", "",""}; //Percent
    String[] txi06_GST = {"", "", "", "",""}; //Tax exempt code

    String[] txi01_PST = {"PG", "PG", "PG","PG","PG"}; //Tax type code
    String[] txi02_PST = {"0.80", "2.00", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_PST = {"", "", "", "",""}; //Percent
    String[] txi06_PST = {"", "", "", "",""}; //Tax exempt code
    String[] pam05 = {"11.50", "28.75", "10.62", "11.50", "-10.00"}; //Extended line-item amount
    String[] pid05 = {"Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"}; //Line item
    description
    AxIt106s[] it106s = {new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s
    ()};

    it106s[0].setIt10618(it10618[0]);
    it106s[0].setIt10719(it10719[0]);

    it106s[1].setIt10618(it10618[1]);
    it106s[1].setIt10719(it10719[1]);

    it106s[2].setIt10618(it10618[2]);
    it106s[2].setIt10719(it10719[2]);

    it106s[3].setIt10618(it10618[3]);
    it106s[3].setIt10719(it10719[3]);

    it106s[4].setIt10618(it10618[4]);
    it106s[4].setIt10719(it10719[4]);
    AxTxi[] txi = {new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi()};
```

Sample AX Refund

```
    txi[0].setTxi(txi01_GST[0], txi02_GST[0], txi03_GST[0], txi06_GST[0]);
    txi[0].setTxi(txi01_PST[0], txi02_PST[0], txi03_PST[0], txi06_PST[0]);
    txi[1].setTxi(txi01_GST[1], txi02_GST[1], txi03_GST[1], txi06_GST[1]);
    txi[1].setTxi(txi01_PST[1], txi02_PST[1], txi03_PST[1], txi06_PST[1]);
    txi[2].setTxi(txi01_GST[2], txi02_GST[2], txi03_GST[2], txi06_GST[2]);
    txi[2].setTxi(txi01_PST[2], txi02_PST[2], txi03_PST[2], txi06_PST[2]);
    txi[3].setTxi(txi01_GST[3], txi02_GST[3], txi03_GST[3], txi06_GST[3]);
    txi[3].setTxi(txi01_PST[3], txi02_PST[3], txi03_PST[3], txi06_PST[3]);
    txi[4].setTxi(txi01_GST[4], txi02_GST[4], txi03_GST[4], txi06_GST[4]);
    txi[4].setTxi(txi01_PST[4], txi02_PST[4], txi03_PST[4], txi06_PST[4]);
    AxIt1Loop it1Loop = new AxIt1Loop();
    it1Loop.setIt1Loop(it102[0], it103[0], it104[0], it105[0], it106s[0], txi[0], pam05[0], pid05[0]);
    it1Loop.setIt1Loop(it102[1], it103[1], it104[1], it105[1], it106s[1], txi[1], pam05[1], pid05[1]);
    it1Loop.setIt1Loop(it102[2], it103[2], it104[2], it105[2], it106s[2], txi[2], pam05[2], pid05[2]);
    it1Loop.setIt1Loop(it102[3], it103[3], it104[3], it105[3], it106s[3], txi[3], pam05[3], pid05[3]);
    it1Loop.setIt1Loop(it102[4], it103[4], it104[4], it105[4], it106s[4], txi[4], pam05[4], pid05[4]);
    AxTable2 table2 = new AxTable2();
    table2.setIt1Loop(it1Loop);
    //Create Table 3 with details
    AxTxi taxTbl3 = new AxTxi();
    taxTbl3.setTxi("GS", "4.25","",""); //sum of GST taxes
    taxTbl3.setTxi("PG", "4.60","",""); //sum of PST taxes
    taxTbl3.setTxi("TX", "8.85","",""); //sum of all taxes
    AxTable3 table3 = new AxTable3();
    table3.setTxi(taxTbl3);

    //Create and set Level23 Object
    AxLevel23 level23 = new AxLevel23();
    level23.setTable1(table1);
    level23.setTable2(table2);
    level23.setTable3(table3);
    AxRefund axRefund = new AxRefund();
    axRefund.setOrderId(order_id);
    axRefund.setAmount(amount);
    axRefund.setTxnNumber(txn_number);
    axRefund.setCryptType(crypt);
    axRefund.setAxLevel23(level23);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(axRefund);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
```

Sample AX Refund

```
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

7.4.8 AX Independent Refund

The AX Independent Refund will credit a specified amount to the cardholder's credit card. The independent refund does not require an existing order to be logged in the Moneris Gateway; however, the credit card number and expiry date will need to be passed.

AX Independent Refund transaction object definition

```
AxIndependentRefund axIndependentRefund = new AxIndependentRefund();
```

HttpsPostRequest object for AX Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(axIndependentRefund);
```

AX Independent Refund transaction object values

Table 1 AX Independent Refund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`axIndependentRefund .setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`axIndependentRefund.setAmount (amount);`
Credit card number	String	20-character alpha-	`axIndependentRefund.setPan`

Value	Type	Limits	Set Method
		numeric	`(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`axIndependentRefund .setExpDate(expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`axIndependentRefund .setCryptType(crypt);`

Table 2 AX Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`axIndependentRefund.setCustId (cust_id);`

Sample AX Independent Refund

```
package Level23;
import JavaAPI.*;
public class TestAxIndependentRefund
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String processing_country_code = "CA";
boolean status_check = false;

java.util.Date createDate = new java.util.Date();
String order_id="Test"+createDate.getTime();
String cust_id="CUST13343";
String amount="62.37";
String pan="373269005095005";
String expiry_date="2012"; //YYMM
String crypt="7";

//Create Table 1 with details
String n101 = "R6"; //Entity ID Code
String n102 = "Retailing Inc. International"; //Name
String n301 = "919 Oriole Rd."; //Address Line 1
String n401 = "Toronto"; //City
String n402 = "On"; //State or Province
String n403 = "H1T6W3"; //Postal Code
String[] ref01 = {"4C", "CR"}; //Reference ID Qualifier
String[] ref02 = {"M5T3A5", "16802309004"}; //Reference ID
String big04 = "PO7758545"; //Purchase Order Number
String big05 = "RN0049858"; //Release Number
String big10 = "INV99870E"; //Invoice Number
AxRef axRef1 = new AxRef();
```

Sample AX Independent Refund

```
    axRef1.setRef(ref01[0], ref02[0]);
    axRef1.setRef(ref01[1], ref02[1]);
    AxN1Loop n1Loop = new AxN1Loop();
    n1Loop.setN1Loop(n101, n102, n301, n401, n402, n403, axRef1);
    AxTable1 table1 = new AxTable1();
    table1.setBig04(big04);
    table1.setBig05(big05);
    table1.setBig10(big10);
    table1.setN1Loop(n1Loop);

    //Create Table 2 with details
    //the sum of the extended amount field (pam05) must equal the level 1 amount field
    String[] it102 = {"1", "1", "1", "1", "1"}; //Line item quantity invoiced
    String[] it103 = {"EA", "EA", "EA", "EA", "EA"}; //Line item unit or basis of measurement code
    String[] it104 = {"10.00", "25.00", "8.62", "10.00", "-10.00"}; //Line item unit price
    String[] it105 = {"", "", "", "", ""}; //Line item basis of unit price code

    String[] it10618 = {"MG", "MG", "MG", "MG", "MG"}; //Product/Service ID qualifier
    String[] it10719 = {"DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"}; //Product/Service ID
    (corresponds to it10618)

    String[] txi01_GST = {"GS", "GS", "GS", "GS", "GS"}; //Tax type code
    String[] txi02_GST = {"0.70", "1.75", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_GST = {"", "", "", "",""}; //Percent
    String[] txi06_GST = {"", "", "", "",""}; //Tax exempt code

    String[] txi01_PST = {"PG", "PG", "PG","PG","PG"}; //Tax type code
    String[] txi02_PST = {"0.80", "2.00", "1.00", "0.80","0.00"}; //Monetary amount
    String[] txi03_PST = {"", "", "", "",""}; //Percent
    String[] txi06_PST = {"", "", "", "",""}; //Tax exempt code
    String[] pam05 = {"11.50", "28.75", "10.62", "11.50", "-10.00"}; //Extended line-item amount
    String[] pid05 = {"Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"}; //Line item
    description
    AxIt106s[] it106s = {new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s(), new AxIt106s
    ()};

    it106s[0].setIt10618(it10618[0]);
    it106s[0].setIt10719(it10719[0]);

    it106s[1].setIt10618(it10618[1]);
    it106s[1].setIt10719(it10719[1]);

    it106s[2].setIt10618(it10618[2]);
    it106s[2].setIt10719(it10719[2]);

    it106s[3].setIt10618(it10618[3]);
    it106s[3].setIt10719(it10719[3]);

    it106s[4].setIt10618(it10618[4]);
    it106s[4].setIt10719(it10719[4]);
    AxTxi[] txi = {new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi(), new AxTxi()};
    txi[0].setTxi(txi01_GST[0], txi02_GST[0], txi03_GST[0], txi06_GST[0]);
    txi[0].setTxi(txi01_PST[0], txi02_PST[0], txi03_PST[0], txi06_PST[0]);
    txi[1].setTxi(txi01_GST[1], txi02_GST[1], txi03_GST[1], txi06_GST[1]);
    txi[1].setTxi(txi01_PST[1], txi02_PST[1], txi03_PST[1], txi06_PST[1]);
    txi[2].setTxi(txi01_GST[2], txi02_GST[2], txi03_GST[2], txi06_GST[2]);
    txi[2].setTxi(txi01_PST[2], txi02_PST[2], txi03_PST[2], txi06_PST[2]);
    txi[3].setTxi(txi01_GST[3], txi02_GST[3], txi03_GST[3], txi06_GST[3]);
    txi[3].setTxi(txi01_PST[3], txi02_PST[3], txi03_PST[3], txi06_PST[3]);
```

Sample AX Independent Refund

```
    txi[4].setTxi(txi01_GST[4], txi02_GST[4], txi03_GST[4], txi06_GST[4]);
    txi[4].setTxi(txi01_PST[4], txi02_PST[4], txi03_PST[4], txi06_PST[4]);
    AxIt1Loop it1Loop = new AxIt1Loop();
    it1Loop.setIt1Loop(it102[0], it103[0], it104[0], it105[0], it106s[0], txi[0], pam05[0], pid05[0]);
    it1Loop.setIt1Loop(it102[1], it103[1], it104[1], it105[1], it106s[1], txi[1], pam05[1], pid05[1]);
    it1Loop.setIt1Loop(it102[2], it103[2], it104[2], it105[2], it106s[2], txi[2], pam05[2], pid05[2]);
    it1Loop.setIt1Loop(it102[3], it103[3], it104[3], it105[3], it106s[3], txi[3], pam05[3], pid05[3]);
    it1Loop.setIt1Loop(it102[4], it103[4], it104[4], it105[4], it106s[4], txi[4], pam05[4], pid05[4]);
    AxTable2 table2 = new AxTable2();
    table2.setIt1Loop(it1Loop);
    //Create Table 3 with details
    AxTxi taxTbl3 = new AxTxi();
    taxTbl3.setTxi("GS", "4.25","",""); //sum of GST taxes
    taxTbl3.setTxi("PG", "4.60","",""); //sum of PST taxes
    taxTbl3.setTxi("TX", "8.85","",""); //sum of all taxes
    AxTable3 table3 = new AxTable3();
    table3.setTxi(taxTbl3);

    //Create and set Level23 Object
    AxLevel23 level23 = new AxLevel23();
    level23.setTable1(table1);
    level23.setTable2(table2);
    level23.setTable3(table3);
    AxIndependentRefund axIndependentRefund = new AxIndependentRefund();
    axIndependentRefund.setOrderId(order_id);
    axIndependentRefund.setCustId(cust_id);
    axIndependentRefund.setAmount(amount);
    axIndependentRefund.setPan(pan);
    axIndependentRefund.setExpDate(expiry_date);
    axIndependentRefund.setCryptType(crypt);
    axIndependentRefund.setAxLevel23(level23);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(axIndependentRefund);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
```

Sample AX Independent Refund

```
    }
    catch (Exception e)
    {
    System.out.println(e);
    }
    }
    }
```

8 MPI

8.1 About MPI Transactions

The Moneris Gateway can enable transactions using the 3-D Secure protocol via Merchant Plug-In (MPI) and Access Control Server (ACS) .

Moneris Gateway supports the following 3-D Secure implementations:

- Verified by Visa (VbV)
- Mastercard Secure Code (MCSC)
- American Express SafeKey (applies to Canadian integrations only)

8.2 3-D Secure Implementations (VbV, MCSC, SafeKey)

Verified by Visa (VbV), MasterCard Secure Code (MCSC) and American Express SafeKey are programs based on the 3-D Secure Protocol to improve the security of online transactions.

These programs involve authentication of the cardholder during an online e-commerce transaction. Authentication is based on the issuer’s selected method of authentication.

The following are examples of authentication methods:

- Risk-based authentication
- Dynamic passwords
- Static passwords.

Some benefits of these programs are reduced risk of fraudulent transactions and protection against chargebacks for certain fraudulent transactions.

Additional eFraud features

To further decrease fraudulent activity, Moneris also recommends implementing the following features:

- Address Verification Service

- Card Validation Digits (CVD)

8.3 Activating 3-D Secure Functionality

To activate Verified by Visa, Mastercard Secure Code and/or American Express SafeKey transaction functionality, call Moneris Sales Support to have Moneris enroll you in the program(s) and enable the functionality on your account.

8.4 Activating Amex SafeKey

To Activate Amex SafeKey transaction functionality with your system via the Moneris Gateway API:

1. Enroll in the SafeKey program with American Express at: https://network.americanexpress.com/ca/en/safekey/index.aspx
2. Call your Moneris sales centre at 1-855-465-4980 to get Amex SafeKey functionality enabled on your account.

8.5 Transaction Flow for MPI

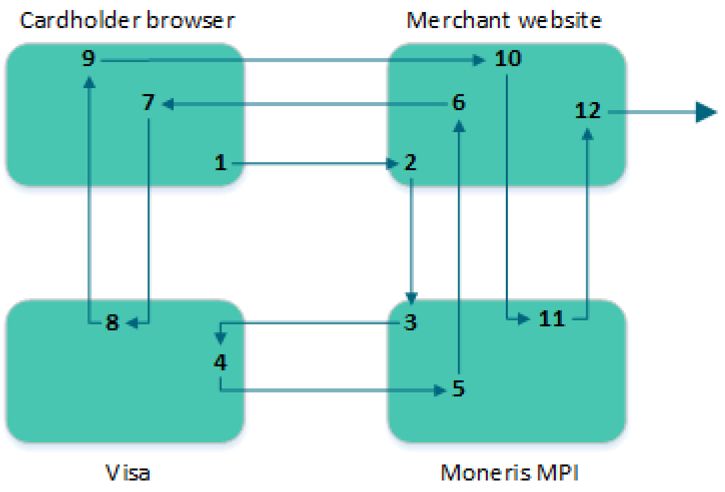


Figure 3: Transaction flow diagram

1. Cardholder enters the credit card number and submits the transaction information to the merchant.
2. Upon receiving the transaction request, the merchant calls the MonerisMPI API and passes a TXN type request. For sample code please refer to MpiTxn Request Transaction (page 264).
3. The Moneris MPI receives the request, authenticates the merchant and sends the transaction information to Visa, MasterCard or American Express.
4. Visa/MasterCard/Amex verifies that the card is enrolled and returns the issuer URL.
5. Moneris MPI receives the response from Visa, MasterCard or Amex and forwards the information to the merchant.
6. The MonerisMPI API installed at the merchant receives the response from the Moneris MPI.

   If the response is "Y" for enrolled, the merchant makes a call to the API, which opens a popup/inline window in the cardholder browser.

If the response is “N” for not enrolled, a transaction could be sent to the processor identifying it as VBV/MCSC attempted with an ECI value of 6; if this response is "N" for American Express SafeKey, the ECI value will be 7.

If the response is “U” for unable to authenticate or the response times out, the transaction can be sent to the processor with an ECI value of 7. The merchant can then choose to continue with the transaction and be liable for a chargeback, or the merchant can choose to end the transaction.

7. The cardholder browser uses the URL that was returned from Visa/MasterCard/Amex via the merchant to communicate directly to the bank. The contents of the popup are loaded and the cardholder enters the PIN.
8. The information is submitted to the bank and authenticated. A response is then returned to the client browser.
9. The client browser receives the response from the bank, and forwards it to the merchant.
10. The merchant receives the response information from the cardholder browser, and passes an ACS request type to the Moneris MPI API.
11. Moneris MPI receives the ACS request and authenticates the information. The Moneris MPI then provides a CAVV value (getCavv()) and a crypt type (getMpiEciO) to the merchant.

    If the getSuccess() of the response is “true”, the merchant may proceed with the cavv purchase or cavv preauth.

    If the getSuccess() of the response is “false” **and** the getMessage() is “N”, the transaction must be cancelled because the cardholder failed to authenticate.

    If the getSuccess() of the response is “false” **and** the getMessage is “U”, the transaction can be processed as a normal purchase or PreAuth; however in this case the merchant assumes liability of a chargeback.

    If the response times out, the transaction can be processed as a normal purchase or PreAuth; however in this case the merchant assumes liability of a chargeback.
12. The merchant retrieves the CAVV value, and formats a cavv purchase or a cavv preauth request using the method that is normally used. As part of this transaction method, the merchant must pass the CAVV value and the crypt type.

8.6 MPI Transactions

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 18.5 Processing a Transaction.

TXN
Sends the initial transaction data to the Moneris MPI to verify whether the card is enrolled.

The browser returns a PARes as well as a success field.

ACS
Passes the PARes (received in the response to the TXN transaction) to the Moneris MPI API.

Cavv Purchase
After receiving confirmation from the ACS transaction, this verifies funds on the customer’s card, removes the funds and prepares them for deposit into the merchant’s account.

Cavv Pre-Authorization
After receiving confirmation from the ACS transaction, this verifies and locks funds on the customer’s credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant’s account, a basic Completion transaction (page 23) must be performed. A PreAuthorization transaction may only be "completed" once.

NOTE: Cavv Purchase and Cavv Pre-Authorization transactions are also used to process Apple Pay and Android Pay transactions. For further details on how to process these wallet transactions, please refer to 11 Apple Pay In-App and on the Web Integration.

8.6.1 VbV, MCSC and SafeKey Responses

For each transaction, a crypt type is sent to identify whether it is a VbV-, MCSC- or SafeKey-authenticated transaction. Below are the tables defining the possible crypt types as well as the possible VARes and PARes responses.

Table 30: Crypt type definitions

Crypt type	Visa definition	MasterCard definition	American Express Definition
5	• Fully authenticated • There is a liability shift, and the merchant is protected from chargebacks	• Fully authenticated • There is a liability shift, and the merchant is protected from chargebacks.	• Fully authenticated • There is a liability shift, and the merchant is protected from chargebacks.
6	• VbV has been attempted • There is a liability shift, and the merchant is protected from certain chargebacks on fraudulent transactions	• MCSC has been attempted • There is a liability shift, and the merchant is protected from certain chargebacks on fraudulent transactions	• SafeKey has been attempted • There is a liability shift, and the merchant is protected from certain chargebacks on fraudulent transactions
7	• Non-VbV transaction • No liability shift • Merchant is not protected from chargebacks	• Non-MCSC transaction • No liability shift • Merchant is not protected from chargebacks	• Non-SafeKey transaction • No liability shift • Merchant is not protected from chargebacks

Table 31: VERes response definitions

VERes Response	Response Definition
N	The card/issuer is not enrolled. Sent as a normal Purchase/Pre-Authorization transaction with a crypt type of 6, except for American Express SafeKey, which will use crypt type of 7.
U	The card type is not participating in VbV/MCSC/SafeKey. It could be corporate card or another card plan that Visa/MasterCard/Amex excludes. Proceed with a regular transaction with a crypt type of 7 or cancel the transaction.
Y	The card is enrolled. Proceed to create the VbV/MCSC/SafeKey inline window for cardholder authentication. Proceed to PARes for crypt type.

Table 32: PARes response definitions

PARes response	Response definition
A	Attempted to verify PIN, and will receive a CAVV. Send as a cavv_purchase/cavv_preAuth, which returns a crypt type of 6.
Y	Fully authenticated, and will receive a CAVV. Send as a cavv_purchase/cavv_preAuth which will return a crypt type of 5.
N	Failed to authenticate. No CAVV is returned. Cancel transaction. Merchant may proceed with a crypt type of 7 although this is strongly discouraged.

Table 33: 3-D Secure/CAVV transaction handling

Step 1: VERes Cardholder/issuer enrolled?	Step 2: PARes 3-D Secure InLine window response	Step 3: Transaction Are you protected?
Y	Y	Send a CAVV transaction
Y	N	Cancel transaction. Authentication failed or high-risk transaction.
Y	A	Send a CAVV transaction
U	n/a	Send a regular transaction with a crypt type of 7

Table 33: 3-D Secure/CAVV transaction handling (continued)

Step 1: VERes Cardholder/issuer enrolled?	Step 2: PARes 3-D Secure InLine window response	Step 3: Transaction Are you protected?
N	n/a	Send a regular transaction with a crypt type of 6, or for American ExpressSafeKey, send with crypt type of 7

8.6.2 MpiTxn Request Transaction

Sends the initial transaction data to the Moneris MPI to verify whether the card is enrolled. The browser returns a PARes as well as a success field.

MpiTxn transaction object definition

```
MpiTxn mpiTxn = new MpiTxn();
```

HttpsPostRequest object for MpiTxn transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mpiTxn);
```

MpiTxn transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 34: MpiTxn transaction object mandatory values

Value	Type	Limits	Set method
XID	String	20-character alpha-numeric	`mpiTxn.setXid(xid);`
Credit card number	String	20-character numeric	`mpiTxn.setPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`mpiTxn.setExpDate(expiry_date);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point	`mpiTxn.setAmount(amount);`

Table 34: MpiTxn transaction object mandatory values (continued)

Value	Type	Limits	Set method
		EXAMPLE: 1234567.89	
MD	String	1024-character alpha-numeric	`mpiTxn.setMD(MD);`
Merchant URL	String	N/A	`mpiTxn.setMerchantUrl (merchantUrl);`
Accept	String	N/A	`mpiTxn.setAccept(accept);`
User Agent	String	N/A	`mpiTxn.setUserAgent (userAgent);`

Sample MpiTxn Request

```
package Canada;
import JavaAPI.*;
public class TestCanadaMpiTxn
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String amount = "1.00";
String xid = "12345678910111214037";
String MD = xid + "mycardinfo" + amount;
String merchantUrl = "www.mystoreurl.com";
String accept = "true";
String userAgent = "Mozilla";
String processing_country_code = "CA";
String pan = "4242424242424242";
String expdate = "1905";
boolean status_check = false;
MpiTxn mpiTxn = new MpiTxn();
mpiTxn.setXid(xid);
mpiTxn.setPan(pan);
mpiTxn.setExpDate(expdate);
mpiTxn.setAmount(amount);
mpiTxn.setMD(MD);
mpiTxn.setMerchantUrl(merchantUrl);
mpiTxn.setHttpAccept(accept);
mpiTxn.setHttpUserAgent(userAgent);
//************************OPTIONAL VARIABLES***************************
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production
transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mpiTxn);
```

Sample MpiTxn Request

```
mpgReq.setStatusCheck(status_check);
mpgReq.send();
/********************** REQUEST ************************/
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("MpiMessage = " + receipt.getMpiMessage());
System.out.println("MpiSuccess = " + receipt.getMpiSuccess());
if (receipt.getMpiSuccess().equals("true"))
{
System.out.println(receipt.getMpiInLineForm());
}
else
{
System.out.println(receipt.getMessage());
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
} // end TestResMpiTxn
```

8.6.2.1 TXN Response and Creating the Popup

The TXN request returns a response with one of several possible values. The get Message method of the response object returns “Y”, “U”, or “N”.

N

Purchase or Pre-Authorization can be sent as a crypt type of 6 (attempted authentication) or 7 (for American Express SafeKey).

Y

A call to the API to create the VBV form is made.

U

(Returned for non-participating cards such as corporate cards)

Merchant can send the transaction with crypt_type 7. However, the merchant is liable for chargebacks.

Below is the TXN response code. This code can be found from the store.java sample included in the download.

```
MpiResponse mpiRes = mpiReq.getResponse();
String crypt_type;

if (mpiRes.getMessage().equals("Y") )
{
    out.print(mpiRes.getInLineForm());
}
else {
    if (mpiRes.getMessage().equals("N") )
        {
        //send transaction using the mpg API
        // use crypt_type="6";
        }
    else // corporate cards, unable to authenticate or times out (eg. MPI is down)
```

```
            {
            //optional to send transaction using the mpg API in this case merchant
            //assumes liability, use crypt_type="7";
            }
        }
```

8.6.3 Vault MPI Transaction – ResMpiTxn

Vault MPI Transaction transaction object definition

```
ResMpiTxn resMpiTxn = new ResMpiTxn();
```

HttpsPostRequest object for Vault MPI Transaction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resMpiTxn);
```

Vault MPI Transaction transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 35: Vault MPI Transaction transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`resMpiTxn.setData(data_key);`
XID	String	20-character alpha-numeric	`resMpiTxn.setXid(xid);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resMpiTxn.setAmount(amount);`
MD	String	1024-character alpha-numeric	`resMpiTxn.setMD(MD);`
Merchant URL	String	n/a	`resMpiTxn.setMerchantUrl (merchantUrl);`

Table 35: Vault MPI Transaction transaction object mandatory values (continued)

Value	Type	Limits	Set method
Accept	String	n/a	`resMpiTxn.setAccept(accept);`
User Agent	String	n/a	`resMpiTxn.setUserAgent (userAgent);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`resMpiTxn.setExpDate(expiry_ date);`

Sample Vault MPI Transaction

```
package Canada;
import java.util.HashMap;
import java.util.Map;
import JavaAPI.*;
public class TestCanadaResMpiTxn
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String data_key = "ot-7hkuLdmybbHdUD0y2gCXQQx6J";
String amount = "1.00";
java.util.Date createDate = new java.util.Date();
String xid = "TEMPXID"+ createDate.getTime();
String MD = xid + "mycardinfo" + amount;
String merchantUrl = "www.mystoreurl.com";
String accept = "true";
String userAgent = "Mozilla";
String processing_country_code = "CA";
String expdate = "1712";
boolean status_check = false;
ResMpiTxn resMpiTxn = new ResMpiTxn();
resMpiTxn.setData(data_key);
resMpiTxn.setXid(xid);
resMpiTxn.setAmount(amount);
resMpiTxn.setMD(MD);
resMpiTxn.setMerchantUrl(merchantUrl);
resMpiTxn.setAccept(accept);
resMpiTxn.setUserAgent(userAgent);
resMpiTxn.setExpDate(expdate);
//************************OPTIONAL VARIABLES***************************
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resMpiTxn);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
/********************** REQUEST ************************/
```

Sample Vault MPI Transaction

```
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("MpiMessage = " + receipt.getMpiMessage());
System.out.println("MpiSuccess = " + receipt.getMpiSuccess());
if (receipt.getMpiSuccess().equals("true"))
{
System.out.println(receipt.getMpiInLineForm());
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
} // end TestResMpiTxn
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Definitions of Response Fields (page 495).

8.6.4 MPI ACS Request

Passes the PARes (received in the response to the MPI TXN transaction) to the Moneris MPI API.

MPI ACS Request transaction object definition

```
MpiAcs mpiAcs = new MpiAcs();
```

HttpsPostRequest object for MPI ACS Request transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mpiAcs);
```

MPI ACS Request transaction values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 36: MPI ACS Request transaction object mandatory values

Value	Type	Limits	Set method
XID	String	20-character alpha-numeric	**NOTE:** Is the concatenated 20-character prefix that forms part of the variable `MD`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`mpiAcs.setAmount(amount);`
MD	String	1024-character alpha-numeric	`mpiAcs.setMD(MD);`
PARes	String	n/a	`mpiAcs.setPaRes(PaRes);`

Sample MPI ACS Request

```
package Canada;
import JavaAPI.*;
public class TestCanadaMpiAcs
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
String amount = "1.00";
String xid = "12345678910111214011";
String MD = xid + "mycardinfo" + amount;
String PaRes = "PaRes string";
String processing_country_code = "CA";
boolean status_check = false;
MpiAcs mpiAcs = new MpiAcs();
mpiAcs.setPaRes(PaRes);
mpiAcs.setMD(MD);
//***********************OPTIONAL VARIABLES*************************
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mpiAcs);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
/********************** REQUEST ************************/
```

Sample MPI ACS Request

```
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("MpiMessage = " + receipt.getMpiMessage());
    System.out.println("MpiSuccess = " + receipt.getMpiSuccess());
    if (receipt.getMpiSuccess().equals("true"))
    {
    System.out.println("CAVV = " + receipt.getMpiCavv());
    System.out.println("Crypt Type = " + receipt.getMpiEci());
    }
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    } // end TestResMpiTxn
```

8.6.4.1 ACS Response and Forming a Transaction

The ACS response contains the CAVV value and the e-commerce indicator. These values are to be passed to the transaction engine using the Cavv Purchase or Cavv Pre-Authorization request. Please see the documentation provided by your payment solution.

Outlined below is how to send a transaction to Moneris Gateway.

```
if ( mpiRes.getSuccess().equals("true") )
    {
    //Send transaction to host using CAVV purchase or CAVV preauth, refer to sample
    //code for Moneris Gateway. Call mpiRes.getCavv() to obtain the CAVV value.
    //If you are using preauth/capture model, be sure to call getMessage() so the
    //value can be stored and used in the capture transaction after on to protect
    //your chargeback liability. (e.g. getMPIMessage()= A = crypt type of 6 for
    //follow on transaction and getMPIMessage() = Y = crypt type of 5 for follow on
    //transaction.
    }
else
    {
        if (mpiRes.getMessage().equals("N"))
        {
        //Do not send transaction as the cardholder failed authentication.
        }
        else
        {
        //Optional to send transaction using the mpg API. In this case merchant
        //assumes liability.
        }
    }
```

8.6.5 Purchase with 3-D Secure – cavvPurchase

The Purchase with 3-D Secure transaction follows a 3-D Secure MPI authentication. After receiving confirmation from the MPI ACS transaction, this Purchase verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

To perform the 3-D Secure authentication, the Moneris MPI or any 3rd party MPI may be used.

This transaction can also be used to process an Apple Pay transaction. This transaction is applicable only if choosing to integrate directly to Apple Wallet (if not using the Moneris Apple Pay SDK). Please refer to 11 Apple Pay In-App and on the Web Integration for more details on your integration options.

Refer to Apple's developer portal for details on integrating directly to the wallet to retrieve the payload data.

Purchase with 3-D Secure transaction object definition

```
CavvPurchase cavvPurchase = new CavvPurchase();
```

HttpsPostRequest object for Purchase with 3-D Secure transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(cavvPurchase);
```

Purchase with 3-D Secure transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`cavvPurchase.setOrderId (order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`cavvPurchase.setAmount (amount);`
credit card number	*String* max 20-character alpha-numeric	`cavvPurchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`cavvPurchase.setExpDate (expiry_date);`
Cardholder Authentication	*String*	`cavvPurchase.setCavv(cavv);`

Variable Name	Type and Limits	Set Method
Verification Value (CAVV) **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	50-character alphanumeric	
electronic commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	*String* 1-character alphanumeric	`cavvPurchase.setCryptType (crypt);`

Following fields are required for Apple Pay and Google Pay only:

Variable Name	Type and Limits	Set Method
network **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	*String* alphabetic	`cavvPurchase.SetNetwork (network);`
data type **NOTE:** This request variable is mandatory for INTERAC®	*String* 3-character alphanumeric	`cavvPurchase.SetDataType (data_type);`

Variable Name	Type and Limits	Set Method
e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.		

Purchase with 3-D Secure transaction request fields – Optional

Value	Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
customer ID	*String* 30-character alphanumeric	`cavvPurchase.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`cavvPurchase.setDynamicDescriptor(dynamic_descriptor);`
card match ID **NOTE:** Applies to Off-linx™ only; must be unique value for each transaction	*String* 50-character alphanumeric	`cavvPurchase.setCmId(transaction_id);`
Customer Information	*Object* N/A	`cavvPurchase.setCustInfo(customer);`
AVS Information	*Object* N/A	`cavvPurchase.setAvsInfo(avsCheck);`
CVD Information	*Object* N/A	`cavvPurchase.setCvdInfo(cvdCheck);`

Value	Limits	Set Method
NOTE: When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.		

Convenience Fee Information
NOTE: Not applicable when processing Apple Pay or Google Pay transactions.
	Object
	N/A
		`ConvFeeInfo convFeeInfo = new ConvFeeInfo();`
		`cavvPurchase.setConvenienceFee (convFeeInfo);`

Recurring Billing `recur` **NOTE:** For sample code for a Purchase with 3-D Secure including the Recurring Billing Info Object, see 8.6.5.1 Purchase with 3-D Secure and Recurring Billing.	*Object* N/A	`cavvPurchase.setRecurInfo (recurInfo);`
wallet indicator **NOTE:** For Cavv Purchase and Cavv Pre-Authorization, wallet indicator applies to Apple Pay or Android Pay only. For more, see Appendix A Definitions of Request Fields	*String* 3-character alphanumeric	`cavvPurchase.setWalletIndicator (wallet_indicator);`
Credential on File Info `cof`	*Object* N/A	`cavvPurchase.setCofInfo(cof);`

Value	Limits	Set Method
NOTE: This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.		

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	`cof.setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Recurring Billing Info Object Request Fields

Variable Name	Type and Limits	Description
number of recurs `num_recurs`	*String* numeric 1-99	The number of times that the transaction must recur
period `period`	*String* numeric 1-999	Number of recur unit intervals that must pass between recurring billings
start date `start_date`	*String* YYMMDD format	Date of the first future recurring billing transaction; this must be a date in the future If an additional charge will be made immediately, the start now variable must be set to true
start now `start_now`	*String* true/false	Set to true if a charge will be made against the card immediately; otherwise set to false When set to false, use Card Verification prior to sending the Purchase with Recurring Billing and Credential on File objects **NOTE:** Amount to be billed immediately can differ from the subsequent recurring amounts
recurring amount `recur_amount`	*String* 10-character decimal, minimum three digits Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	Dollar amount of the recurring transaction This amount will be billed on the start date, and then billed repeatedly based on the interval defined by period and recur unit
recur unit	*String*	Unit to be used as a basis for the inter-

Variable Name	Type and Limits	Description
`recur_unit`	day, week, month or eom	val Works in conjunction with the period variable to define the billing frequency

Sample Purchase with 3-D Secure – cavvPurchase

```
package Canada;
import JavaAPI.*;
public class TestCanadaCavvPurchase
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
CavvPurchase cavvPurchase = new CavvPurchase();
cavvPurchase.setOrderId(order_id);
cavvPurchase.setCustId(cust_id);
cavvPurchase.setAmount(amount);
cavvPurchase.setPan(pan);
cavvPurchase.setExpdate(expdate);
cavvPurchase.setCavv(cavv);
cavvPurchase.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPurchase.setDynamicDescriptor(dynamic_descriptor);
//cavvPurchase.setWalletIndicator("APP"); //set only for wallet transactions. e.g APPLE PAY
//cavvPurchase.setNetwork("Interac"); //set only for Interac e-commerce
//cavvPurchase.setDataType("3DSecure"); //set only for Interac e-commerce
cavvPurchase.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
alphanumeric characters transaction id generated by merchant

//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cavvPurchase.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPurchase);
mpgReq.setStatusCheck(status_check);
```

Sample Purchase with 3-D Secure – cavvPurchase

```
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
System.out.println("IssuerId = " + receipt.getIssuerId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

8.6.5.1 Purchase with 3-D Secure and Recurring Billing

The example below illustrates the Purchase with 3-D Secure when also sending the Recurring Billing Info object in the transaction.

Purchase with 3-D Secure and Recurring Billing

```
package Canada;
import JavaAPI.*;
public class TestCanadaCavvPurchaseRecur
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
```

Purchase with 3-D Secure and Recurring Billing

```
/************************* Recur Variables **********************************/
String recur_unit = "month"; //eom = end of month
String start_now = "true";
String start_date = "2018/02/09";
String num_recurs = "12";
String period = "1";
String recur_amount = "5.00";
/************************* Recur Object Option1 ******************************/
Recur recurring_cycle = new Recur(recur_unit, start_now, start_date,
num_recurs, period, recur_amount);
CavvPurchase cavvPurchase = new CavvPurchase();
cavvPurchase.setOrderId(order_id);
cavvPurchase.setCustId(cust_id);
cavvPurchase.setAmount(amount);
cavvPurchase.setPan(pan);
cavvPurchase.setExpdate(expdate);
cavvPurchase.setCavv(cavv);
cavvPurchase.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPurchase.setDynamicDescriptor(dynamic_descriptor);
cavvPurchase.setRecur(recurring_cycle);
//cavvPurchase.setWalletIndicator("APP"); //set only for wallet transactions. e.g APPLE PAY
//cavvPurchase.setNetwork("Interac"); //set only for Interac e-commerce
//cavvPurchase.setDataType("3DSecure"); //set only for Interac e-commerce
//Mandatory on Recurs - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("R");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cavvPurchase.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPurchase);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
```

Purchase with 3-D Secure and Recurring Billing

```
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

8.6.6 Pre-Authorization with 3-D Secure – cavvPreauth

The Pre-Authorization with 3-D Secure transaction follows a 3-D Secure MPI authentication. After receiving confirmation from the MPI ACS transaction, this Pre-Authorization verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

To perform the 3-D Secure authentication, the Moneris MPI or any 3rd party MPI may be used.

This transaction can also be used to process an Apple Pay transaction. This transaction is applicable only if choosing to integrate directly to Apple Wallet (if not using the Moneris Apple Pay SDK). Please refer to 11 Apple Pay In-App and on the Web Integration for more details on your integration options.

Refer to Apple's developer portal for details on integrating directly to the wallet to retrieve the payload data.

Pre-Authorization with 3-D Secure transaction object definition

```
CavvPreAuth cavvPreauth = new CavvPreAuth();
```

HttpsPostRequest object for Pre-Authorization with 3-D Secure transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(cavvPreauth);
```

Pre-Authorization with 3-D Secure transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`cavvPreauth.setOrderId (order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) +	`cavvPreauth.setAmount (amount);`

Variable Name	Type and Limits	Set Method
	decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	
credit card number	*String* max 20-character alpha-numeric	`cavvPreauth.setPan(pan);`
Cardholder Authentication Verification Value (CAVV) **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	*String* 50-character alphanumeric	`cavvPreauth.setCavv(cavv);`
expiry date	*String* 4-character alphanumeric YYMM	`cavvPreauth.setExpDate (expiry_date);`
electronic commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	*String* 1-character alphanumeric	`cavvPreauth.setCryptType (crypt);`

Following fields are required for Apple Pay and Google Pay only:

Variable Name	Type and Limits	Set Method
network **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	*String* alphabetic	`cavvPurchase.SetNetwork (network);`
data type **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	*String* 3-character alphanumeric	`cavvPurchase.SetDataType (data_type);`

Pre-Authorization with 3-D Secure transaction request fields – Optional

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck (status_check);`
customer ID	*String* 30-character alphanumeric	`cavvPreauth.setCustId(cust_ id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`cavvPreauth .setDynamicDescriptor (dynamic_descriptor);`
card match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String* 50-character alphanumeric	`cavvPreauth.setCmId (transaction_id);`

Variable Name	Type and Limits	Set Method
AVS Information	*Object* N/A	`cavvPreauth.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`cavvPreauth.setCvdInfo (cvdCheck);`
wallet indicator **NOTE:** For Cavv Purchase and Cavv Pre-Authorization, wallet indicator applies to Apple Pay or Android Pay only. For more, see Appendix A Definitions of Request Fields	*String* 3-character alphanumeric	`cavvPreauth .setWalletIndicator(wallet_ indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`cavvPreauth.setCofInfo(cof);`

Credential on File Info object request fields

Variable Name	Type and Limits	Set Method
issuer ID **NOTE:** This variable is	*String* 15-character alphanumeric	`cof.setIssuerId("VALUE_FOR_ ISSUER_ID");`

Variable Name	Type and Limits	Set Method
required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	variable length	**NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment indicator	*String* 1-character alphabetic	`cof.setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File
payment information	*String* 1-character numeric	`cof.setPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definition of Request Fields – Credential on File

Sample Pre-Authorization with 3-D Secure – cavvPreauth

```
package Canada;
import JavaAPI.*;
public class TestCanadaCavvPreauth
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1911"; //YYMM format
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
CavvPreAuth cavvPreauth = new CavvPreAuth();
cavvPreauth.setOrderId(order_id);
cavvPreauth.setCustId(cust_id);
cavvPreauth.setAmount(amount);
```

Sample Pre-Authorization with 3-D Secure – cavvPreauth

```
cavvPreauth.setPan(pan);
cavvPreauth.setExpdate(expdate);
cavvPreauth.setCavv(cavv);
cavvPreauth.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPreauth.setDynamicDescriptor(dynamic_descriptor);
//cavvPreauth.setWalletIndicator("APP"); //set only for wallet transactions. e.g APPLE PAY
cavvPreauth.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
alphanumeric characters transaction id generated by merchant
//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cavvPreauth.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPreauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
System.out.println("IssuerId = " + receipt.getIssuerId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

8.6.7 Cavv Result Codes for Verified by Visa

Code	Message	Significance
0	CAVV authentication results invalid	For this transaction, you may not receive protection from chargebacks as a result of using VbV because the CAVV was considered invalid at the time the financial transaction was processed. Check that you are following the VbV process correctly and passing the correct data in our transactions.
1	CAVV failed validation; authentication	Provided that you have implemented the VbV process correctly, the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
2	CAVV passed validation; authentication	The CAVV was confirmed as part of the financial transaction. This transaction is a fully authenticated VbV transaction (ECI 5)
3	CAVV passed validation; attempt	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VbV transaction (ECI 6)
4	CAVV failed validation; attempt	Provided that you have implemented the VbV process correctly the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
7	CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VbV process correctly and passing the correct data in your transactions. Provided that you have implemented the VbV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
8	CAVV passed validation; attempt (US issued cards only	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VbV transaction (ECI 6)
9	CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VbV process correctly and passing the correct data in our transactions.

Code	Message	Significance
		Provided that you have implemented the VbV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
A	CAVV passed validation; attempt (US issued cards only)	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VbV transaction (ECI 6)
B	CAVV passed validation; information only, no liability shift	The CAVV was confirmed as part of the financial transaction. However, this transaction does not qualify for the liability shift. Treat this transaction the same as an ECI 7.

8.6.8 Vault Cavv Purchase

Vault Cavv Purchase transaction object definition

```
ResCavvPurchaseCC resCavvPurchaseCC = new ResCavvPurchaseCC();
```

HttpsPostRequest object for Vault Cavv Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(resCavvPurchaseCC);
```

Vault Cavv Purchase transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resCavvPurchaseCC.setData (data_key);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resCavvPurchaseCC.setOrderId (order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal	`resCavvPurchaseCC.setAmount (amount);`

Variable Name	Type and Limits	Set Method
	point **EXAMPLE:** 1234567.89	
Cardholder Authentication Verification Value (CAVV)	*String* 50-character alphanumeric	`resCavvPurchaseCC.setCavv (cavv);`
electronic commerce indicator	*String* 1-character alphanumeric	`resCavvPurchaseCC .setCryptType(crypt);`

Vault Cavv Purchase transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`resCavvPurchaseCC.setCustId (cust_id);`
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
expiry date	*String* 4-character alphanumeric YYMM	`resCavvPurchaseCC.setExpDate (expiry_date);`

Sample Vault Cavv Purchase

```
package Canada;
import JavaAPI.*;
public class TestCanadaResCavvPurchaseCC
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "4INQR1A8ocxD0oafSz50LADXy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
String processing_country_code = "CA";
String exp_date = "1901";
```

Sample Vault Cavv Purchase

```
    boolean status_check = false;
    ResCavvPurchaseCC resCavvPurchaseCC = new ResCavvPurchaseCC();
    resCavvPurchaseCC.setOrderId(order_id);
    resCavvPurchaseCC.setData(data_key);
    resCavvPurchaseCC.setCustId(cust_id);
    resCavvPurchaseCC.setAmount(amount);
    resCavvPurchaseCC.setCavv(cavv);
    resCavvPurchaseCC.setExpDate(exp_date);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(resCavvPurchaseCC);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("DataKey = " + receipt.getDataKey());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("ResSuccess = " + receipt.getResSuccess());
    System.out.println("PaymentType = " + receipt.getPaymentType());
    System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
    //ResolveData
    System.out.println("Cust ID = " + receipt.getResCustId());
    System.out.println("Phone = " + receipt.getResPhone());
    System.out.println("Email = " + receipt.getResEmail());
    System.out.println("Note = " + receipt.getResNote());
    System.out.println("Masked Pan = " + receipt.getResMaskedPan());
    System.out.println("Exp Date = " + receipt.getResExpdate());
    System.out.println("Crypt Type = " + receipt.getResCryptType());
    System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
    System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
    System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

8.6.9 Vault Cavv Pre-Authorization

Vault Cavv Pre-Authorization transaction object definition

```
ResCavvPreAuthCC resCavvPreauthCC = new ResCavvPreAuthCC();
```

HttpsPostRequest object for Vault Cavv Pre-Authorization

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(resCavvPreauthCC);
```

Vault Cavv Pre-Authorization transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`resCavvPreauthCC.setData (data_key);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`resCavvPreauthCC.setOrderId (order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`resCavvPreauthCC.setAmount (amount);`
Cardholder Authentication Verification Value (CAVV)	*String* 50-character alphanumeric	`resCavvPreauthCC.setCavv (cavv);`
electronic commerce indicator	*String* 1-character alphanumeric	`resCavvPreauthCC .setCryptType(crypt);`

Vault Cavv Pre-Authorization transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String*	`resCavvPreauthCC.setCustId`

Variable Name	Type and Limits	Set Method
	30-character alphanumeric	`(cust_id);`
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
expiry date	*String* 4-character alphanumeric YYMM	`resCavvPreauthCC.setExpDate(expiry_date);`

Sample Vault Cavv Pre-Authorization

```
package Canada;
import JavaAPI.*;
public class TestCanadaResCavvPreauthCC
{
public static void main(String[] args)
{
String store_id = "store1";
String api_token = "yesguy";
String data_key = "4INQR1A8ocxD0oafSz50LADXy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
String processing_country_code = "CA";
String expdate = "1901";
boolean status_check = false;
ResCavvPreauthCC resCavvPreauthCC = new ResCavvPreauthCC();
resCavvPreauthCC.setOrderId(order_id);
resCavvPreauthCC.setData(data_key);
resCavvPreauthCC.setCustId(cust_id);
resCavvPreauthCC.setAmount(amount);
resCavvPreauthCC.setCavv(cavv);
resCavvPreauthCC.setExpDate(expdate);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(resCavvPreauthCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
```

Sample Vault Cavv Pre-Authorization

```
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
//ResolveData
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Masked Pan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

9 Multi-Currency Pricing (MCP)

9.1 About Multi-Currency Pricing (MCP)

Multi-currency pricing (MCP) is a financial service which allows businesses to price goods and services in a variety of foreign currencies, while continuing to receive settlement and reporting in Canadian dollars. MCP allows cardholders to shop, view prices and pay in the currency of their choice.

MCP is only available when processing Visa and Mastercard transactions.

NOTE: Use MCP only when processing transactions that involve foreign currency exchange; for transactions strictly in Canadian dollars, use the basic financial transaction requests

9.2 Methods of Processing MCP Transactions

There are two methods of processing multi-currency pricing transactions via the Moneris Gateway:

1. **Using the MCP Get Rate transaction** – this method is used to obtain a foreign exchange rate and locks that specific rate in for a limited time, and is applied in a subsequent transaction
2. **Without using MCP Get Rate** – this method sends a MCP transaction without performing the Get Rate request, and the foreign exchange rate is obtained at processing time

9.3 MCP Get Rate

Performs a foreign currency exchange rate look-up, and secures that exchange rate for use in a subsequent MCP financial transaction.

The exchange rate retrieved by this transaction request is represented in the response as the **RateToken**, and the underlying exchange rate is locked in for a limited time period.

MCP Get Rate transaction object definition

```
MCPGetRate getRate = new MCPGetRate();
```

HttpsPostRequest object for MCP Get Rate transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(getRate);
```

MCP Get Rate transaction request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`getRate.setMCPVersion("MCP_VERSION_NUM");`
rate transaction type	*String* 1-character alphabetic	`getRate.setRateTxnType("TRANSACTION_TYPE_VALUE");`
MCP Rate Info	*Object* N/A	`getRate.setMCPRateInfo(rate);`

MCP Rate Info object request fields

At least one of the following variables must be sent:

Variable Name	Type and Limits	Set Method
add cardholder amount	*String array* 12-character numeric, 3-character numeric (smallest discrete unit of foreign currency, currency code)	`rate.addCardholderAmount("FOREIGN_AMT", "FOREIGN_CURRENCY_CODE");`
add merchant settlement amount	*String array* 12-character numeric, 3-	`rate.addMerchantSettlementAmount`

Variable Name	Type and Limits	Set Method
	character numeric (amount in CAD pennies, currency code)	`("CAD_AMOUNT", "FOREIGN_CURRENCY_CODE");`

Sample MCP Get Rate

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPGetRate
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String processing_country_code = "CA";


MCPGetRate getRate = new MCPGetRate();
getRate.setMCPVersion("1.0"); //MCP Version number. Should always be 1.0
getRate.setRateTxnType("P"); //P or R are valid values (Purchase or Refund)

MCPRate rate = new MCPRate();
rate.addCardholderAmount("500", "840"); //penny value amount 1.25 = 125. Foreign amount and SO-4217
country currency number
//rate.addMerchantSettlementAmount("200", "826"); //penny value amount 1.25 = 125. Domestic(CAD)
amount and SO-4217 country currency number
//rate.addMerchantSettlementAmount("300", "036"); //penny value amount 1.25 = 125. Domestic(CAD)
amount and SO-4217 country currency number

getRate.setMCPRateInfo(rate);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(getRate);
mpgReq.send();

try
{
Receipt receipt = mpgReq.getReceipt();

System.out.println("RateTxnType = " + receipt.getRateTxnType());
System.out.println("MCPRateToken = " + receipt.getMCPRateToken());

System.out.println("RateInqStartTime = " + receipt.getRateInqStartTime()); //The time (unix UTC) of
when the rate is requested
System.out.println("RateInqEndTime = " + receipt.getRateInqEndTime()); //The time (unix UTC) of when
the rate is returned
System.out.println("RateValidityStartTime = " + receipt.getRateValidityStartTime()); //The time
(unix UTC) of when the rate is valid from
System.out.println("RateValidityEndTime = " + receipt.getRateValidityEndTime()); //The time (unix
UTC) of when the rate is valid until
System.out.println("RateValidityPeriod = " + receipt.getRateValidityPeriod()); //The time in minutes
this rate is valid for
```

```
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("TimedOut = " + receipt.getTimedOut());

    //RateData
    for (int index = 0; index < receipt.getRatesCount(); index++)
    {
    System.out.println("MCPRate = " + receipt.getMCPRate(index));
    System.out.println("MerchantSettlementCurrency = " + receipt.getMerchantSettlementCurrency(index));
    System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount(index));
    //Domestic(CAD) amount
    System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode(index));
    System.out.println("CardholderAmount = " + receipt.getCardholderAmount(index)); //Foreign amount

    System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode(index));
    System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage(index));
    }

    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.4 MCP Purchase

Verifies funds on the customer’s card, removes the funds and prepares them for deposit into the merchant’s account.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Purchase transaction object definition

```
MCPPurchase mcpPurchase = new MCPPurchase();
```

HttpsPostRequest object for MCP Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpPurchase);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Purchase transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`mcpPurchase.setOrderId (order_id);`
credit card number	*String* max 20-character alpha-numeric	`mcpPurchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`mcpPurchase.setExpDate (expiry_date);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpPurchase.setCryptType (crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpPurchase.setMCPVersion ("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of for-eign currency	`mcpPurchase .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpPurchase .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Purchase transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String*	`mcpPurchase.setCustId(cust_`

Variable Name	Type and Limits	Set Method
	30-character alphanumeric	`id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`mcpPurchase .setDynamicDescriptor (dynamic_descriptor);`
wallet indicator	*String* 3-character alphanumeric	`mcpPurchase .setWalletIndicator(wallet_ indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`mcpPurchase.setCofInfo(cof);`
AVS Information	*Object* N/A	`mcpPurchase.setAvsInfo (avsCheck);`
CVD Information	*Object* N/A	`mcpPurchase.setCvdInfo (cvdCheck);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpPurchase.setMCPRateToken ("MCP_RATE_TOKEN");`

Sample MCP Purchase

```
    package Canada;
    import JavaAPI.*;
    public class TestCanadaMCPPurchase
```

```
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String amount = "5.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM format
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
MCPPurchase mcpPurchase = new MCPPurchase();
mcpPurchase.setOrderId(order_id);
mcpPurchase.setAmount(amount);
mcpPurchase.setPan(pan);
mcpPurchase.setExpdate(expdate);
mcpPurchase.setCryptType(crypt);
mcpPurchase.setDynamicDescriptor("123456");
//mcpPurchase.setWalletIndicator(""); //Refer documentation for possible values
//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

//mcpPurchase.setCofInfo(cof);

//MCP Fields
mcpPurchase.setMCPVersion("1.0");
mcpPurchase.setCardholderAmount("500");
mcpPurchase.setCardholderCurrencyCode("840");
mcpPurchase.setMCPRateToken("P1538681661706745");

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpPurchase);
mpgReq.setStatusCheck(status_check);

//Optional - Proxy
mpgReq.setProxy(false); //true to use proxy
mpgReq.setProxyHost("proxyURL");
mpgReq.setProxyPort("proxyPort");
mpgReq.setProxyUser("proxyUser"); //optional - domainName\User
mpgReq.setProxyPassword("proxyPassword"); //optional
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
```

```
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    System.out.println("HostId = " + receipt.getHostId());
    System.out.println("IssuerId = " + receipt.getIssuerId());

    System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
    System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
    System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
    System.out.println("MCPRate = " + receipt.getMCPRate());
    System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
    System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
    System.out.println("HostId = " + receipt.getHostId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.5 MCP Pre-Authorization

Verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time based on the card issuer.To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant's account, a Pre-Authorization Completion transaction must be performed. A Pre-Authorization transaction may only be "completed" once.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Pre-Authorization transaction object definition

`MCPPreAuth mcpPreauth = new MCPPreAuth();`

HttpsPostRequest object for MCP Pre-Authorization transaction

`HttpsPostRequest mpgReq = new HttpsPostRequest();`

`mpgReq.setTransaction(mcpPreauth);`

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Pre-Authorization transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`mcpPreauth.setOrderId(order_id);`
credit card number	*String* max 20-character alpha-numeric	`mcpPreauth.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`mcpPreauth.setExpDate(expiry_date);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpPreauth.setCryptType(crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpPreauth.setMCPVersion("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of for-eign currency	`mcpPreauth.setCardholderAmount("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpPreauth.setCardholderCurrencyCode("CARDHOLDER_CURRENCY_CODE");`

MCP Pre-Authorization transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String*	`mcpPreauth.setCustId(cust_`

Variable Name	Type and Limits	Set Method
	30-character alphanumeric	`id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`mcpPreauth .setDynamicDescriptor (dynamic_descriptor);`
wallet indicator	*String* 3-character alphanumeric	`mcpPreauth .setWalletIndicator(wallet_ indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`mcpPreauth.setCofInfo(cof);`
AVS Information	*Object* N/A	`mcpPreauth.setAvsInfo (avsCheck);`
CVD Information	*Object* N/A	`mcpPreauth.setCvdInfo (cvdCheck);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpPreauth.setMCPRateToken ("MCP_RATE_TOKEN");`

Sample MCP Pre-Authorization

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPPreauth
```

```
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "5.00";
String pan = "4242424242424242";
String expdate = "1902";
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
MCPPreAuth mcpPreauth = new MCPPreAuth();
mcpPreauth.setOrderId(order_id);
mcpPreauth.setAmount(amount);
mcpPreauth.setPan(pan);
mcpPreauth.setExpdate(expdate);
mcpPreauth.setCryptType(crypt);
//mcpPreauth.setWalletIndicator(""); //Refer documentation for possible values
//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

//mcpPreauth.setCofInfo(cof);

//MCP Fields
mcpPreauth.setMCPVersion("1.0");
mcpPreauth.setCardholderAmount("500");
mcpPreauth.setCardholderCurrencyCode("840");
mcpPreauth.setMCPRateToken("P1538681661706745");

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpPreauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
//System.out.println("StatusCode = " + receipt.getStatusCode());
//System.out.println("StatusMessage = " + receipt.getStatusMessage());
System.out.println("IssuerId = " + receipt.getIssuerId());
System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
System.out.println("MCPRate = " + receipt.getMCPRate());
System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
System.out.println("HostId = " + receipt.getHostId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

9.6 MCP Pre-Authorization Completion

Retrieves funds that have been locked by an MCP Pre-Authorization transaction, and prepares them for settlement into the merchant's account.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Pre-Authorization Completion transaction object definition

```
MCPCompletion mcpCompletion = new MCPCompletion();
```

HttpsPostRequest object for MCP Pre-Authorization Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpCompletion);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Pre-Authorization Completion transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`mcpCompletion.setOrderId (order_id);`

Variable Name	Type and Limits	Set Method
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores variable length	`mcpCompletion.setTxnNumber (txn_number);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpCompletion.setCryptType (crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpCompletion.setMCPVersion ("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of for-eign currency	`mcpCompletion .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpCompletion .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Pre-Authorization Completion transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`mcpCompletion.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`mcpCompletion .setDynamicDescriptor (dynamic_descriptor);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpCompletion .setMCPRateToken("MCP_RATE_ TOKEN");`

Sample MCP Pre-Authorization Completion

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPCompletion
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test1538681966167";
String txn_number = "696294-0_11";
String crypt = "7";
String cust_id = "my customer id";
String dynamic_descriptor = "my descriptor";
String ship_indicator = "F" ;
String processing_country_code = "CA";
boolean status_check = false;
MCPCompletion mcpCompletion = new MCPCompletion();
mcpCompletion.setOrderId(order_id);
mcpCompletion.setTxnNumber(txn_number);
mcpCompletion.setCryptType(crypt);
mcpCompletion.setCustId(cust_id);
mcpCompletion.setDynamicDescriptor(dynamic_descriptor);
//mcpCompletion.setShipIndicator(ship_indicator); //optional
//MCP Fields
mcpCompletion.setMCPVersion("1.0");
mcpCompletion.setCardholderAmount("500");
mcpCompletion.setCardholderCurrencyCode("840");
//mcpCompletion.setMCPRateToken("P1538681661706745"); //optional

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpCompletion);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
```

```
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());

    System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
    System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
    System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
    System.out.println("MCPRate = " + receipt.getMCPRate());
    System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
    System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
    System.out.println("HostId = " + receipt.getHostId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.7 MCP Purchase Correction

Restores the full amount of a previous MCP Purchase or MCP Pre-Authorization Completion transaction to the cardholder's card, and removes any record of it from the cardholder's statement.

This transaction can be used against a Purchase or Pre-Authorization Completion transaction that occurred same day provided that the batch containing the original transaction remains open.

MCP processing uses the automated closing feature, and Batch Close occurs daily between 10 and 11 pm Eastern Time.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Purchase Correction transaction object definition

```
MCPPurchaseCorrection mcpPurchasecorrection = new MCPPurchaseCorrection();
```

HttpsPostRequest object for MCP Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpPurchasecorrection);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_ token);`

MCP Purchase Correction transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpPurchasecorrection .setOrderId(order_id);`
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores variable length	`mcpPurchasecorrection .setTxnNumber(txn_number);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpPurchasecorrection .setCryptType(crypt);`

MCP Purchase Correction transaction request fields – Optional

Variable Name	Type and Limits	Description
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters includ-ing your merchant name and separator	`mcpPurchasecorrection .setDynamicDescriptor (dynamic_descriptor);`
customer ID	*String* 30-character alphanumeric	`mcpPurchasecorrection .setCustId(cust_id);`

Sample MCP Purchase Correction

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPPurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test1538682314339";
String txn_number = "696314-0_11";
String crypt = "7";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
boolean status_check = false;
```

```
    MCPPurchaseCorrection mcpPurchasecorrection = new MCPPurchaseCorrection();
    mcpPurchasecorrection.setOrderId(order_id);
    mcpPurchasecorrection.setTxnNumber(txn_number);
    mcpPurchasecorrection.setCryptType(crypt);
    mcpPurchasecorrection.setDynamicDescriptor(dynamic_descriptor);
    mcpPurchasecorrection.setCustId("my customer id");
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(mcpPurchasecorrection);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.8 MCP Refund

Restores all or part of the funds from a MCP Purchase or MCP Pre-Authorization Completion transaction to the cardholder's card.

Unlike a MCP Purchase Correction, there is a record of both the initial charge and the refund on the cardholder's statement.

For processing refunds on a different card than the one used in the original transaction, the MCP Independent Refund transaction should be used instead.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Refund transaction object definition

```
MCPRefund mcpRefund = new MCPRefund();
```

HttpsPostRequest object for MCP Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpRefund);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Refund transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpRefund.setOrderId(order_id);`
transaction number	*String* 255-character, alpha-numeric, hyphens or under-scores variable length	`mcpRefund.setTxnNumber(txn_number);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpRefund.setCryptType(crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpRefund.setMCPVersion("MCP_VERSION_NUM");`
cardholder amount	*String*	`mcpRefund.setCardholderAmount("CARDHOLDER_AMOUNT");`

Variable Name	Type and Limits	Set Method
	12-character numeric smallest discrete unit of foreign currency	
cardholder currency code	*String* 3-character numeric	`mcpRefund .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Refund transaction request fields – Optional

Variable Name	Type and Limits	Description
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`mcpRefund .setDynamicDescriptor (dynamic_descriptor);`
customer ID	*String* 30-character alphanumeric	`mcpRefund.setCustId(cust_id);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpRefund.setMCPRateToken ("MCP_RATE_TOKEN");`

Sample MCP Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPRefund
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String amount = "2.00";
String crypt = "7";
String dynamic_descriptor = "123456";
String custid = "mycust9";
String order_id = "Test1534871380572";
String txn_number = "332654-0_11";
String processing_country_code = "CA";
boolean status_check = false;
MCPRefund mcpRefund = new MCPRefund();
```

```
    mcpRefund.setTxnNumber(txn_number);
    mcpRefund.setOrderId(order_id);
    mcpRefund.setCryptType(crypt);
    mcpRefund.setCustId(custid);
    mcpRefund.setDynamicDescriptor(dynamic_descriptor);

    //MCP Fields
    mcpRefund.setMCPVersion("1.0");
    mcpRefund.setCardholderAmount("200");
    mcpRefund.setCardholderCurrencyCode("840");
    mcpRefund.setMCPRateToken("P1534873994652426");
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(mcpRefund);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());

    System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
    System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
    System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
    System.out.println("MCPRate = " + receipt.getMCPRate());
    System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
    System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
    System.out.println("HostId = " + receipt.getHostId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.9 MCP Independent Refund

Credits a specified amount to the cardholder’s credit card. The credit card number and expiry date are mandatory.

It is not necessary for the transaction that you are refunding to have been processed via the Moneris Gateway.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

Things to Consider:

- Because of the potential for fraud, permission for this transaction is not granted to all accounts by default. If it is required for your business, it must be requested via your account manager.

MCP Independent Refund transaction object definition

```
MCPIndependentRefund mcpIndrefund = new MCPIndependentRefund();
```

HttpsPostRequest object for MCP Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpIndrefund);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Independent Refund transaction request fields – Required

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpIndrefund.setOrderId(order_id);`
credit card number	*String* max 20-character alpha-numeric	`mcpIndrefund.setPan(pan);`
expiry date	*String* 4-character alphanumeric	`mcpIndrefund.setExpDate(expiry_date);`

Variable Name	Type and Limits	Set Method
	YYMM	
electronic commerce indicator	*String* 1-character alphanumeric	`mcpIndrefund.setCryptType (crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpIndrefund.setMCPVersion ("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of foreign currency	`mcpIndrefund .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpIndrefund .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Independent Refund transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`mcpIndrefund.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`mcpIndrefund .setDynamicDescriptor (dynamic_descriptor);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpIndrefund.setMCPRateToken ("MCP_RATE_TOKEN");`

Sample MCP Independent Refund

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPIndependentRefund
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String cust_id = "my customer id";
String amount = "20.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
MCPIndependentRefund mcpIndrefund = new MCPIndependentRefund();
mcpIndrefund.setOrderId(order_id);
mcpIndrefund.setCustId(cust_id);
mcpIndrefund.setAmount(amount);
mcpIndrefund.setPan(pan);
mcpIndrefund.setExpdate(expdate);
mcpIndrefund.setCryptType(crypt);
mcpIndrefund.setDynamicDescriptor("123456");

//MCP Fields
mcpIndrefund.setMCPVersion("1.0");
mcpIndrefund.setCardholderAmount("500");
mcpIndrefund.setCardholderCurrencyCode("840");
mcpIndrefund.setMCPRateToken("R1538679861330690");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpIndrefund);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
```

```
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());

    System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
    System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
    System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
    System.out.println("MCPRate = " + receipt.getMCPRate());
    System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
    System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
    System.out.println("HostId = " + receipt.getHostId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

9.10 MCP Purchase with Vault

This transaction uses the data key to identify a previously registered credit card profile in Vault. The details saved within the profile are then submitted to perform a Purchase transaction.

The data key may be a temporary one generated used Hosted Tokenization, or may be a permanent one from the Vault.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Purchase with Vault transaction object definition

```
MCPResPurchaseCC mcpResPurchaseCC = new MCPResPurchaseCC();
```

HttpsPostRequest object for MCP Purchase with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(mcpResPurchaseCC);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_token);`

MCP Purchase with Vault transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`mcpResPurchaseCC.setData (data_key);`
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpResPurchaseCC.setOrderId (order_id);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpResPurchaseCC .setCryptType(crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpResPurchaseCC .setMCPVersion("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of for-eign currency	`mcpResPurchaseCC .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpResPurchaseCC .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Purchase with Vault transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`mcpResPurchaseCC.setCustId (cust_id);`
Credential on File Info `cof`	*Object* N/A	`mcpResPurchaseCC.setCofInfo (cof);`

Variable Name	Type and Limits	Set Method
NOTE: This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.		
AVS Information	*Object* N/A	`mcpResPurchaseCC.setAvsInfo (avsCheck);`
CVD Information	*Object* N/A	`mcpResPurchaseCC.setCvdInfo (cvdCheck);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpResPurchaseCC .setMCPRateToken("MCP_RATE_ TOKEN");`

Sample MCP Purchase with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPResPurchaseCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "8O0XGiwxgvfbZngigVFeld9d2";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String crypt_type = "1";
String descriptor = "my descriptor";
String processing_country_code = "CA";
String expdate = "1512"; //For Temp Token
boolean status_check = false;
MCPResPurchaseCC mcpResPurchaseCC = new MCPResPurchaseCC();
mcpResPurchaseCC.setDataKey(data_key);
mcpResPurchaseCC.setOrderId(order_id);
mcpResPurchaseCC.setCustId(cust_id);
mcpResPurchaseCC.setCryptType(crypt_type);
//resPurchaseCC.setDynamicDescriptor(descriptor);
//resPurchaseCC.setExpDate(expdate); //Temp Tokens only
//MCP Fields
mcpResPurchaseCC.setMCPVersion("1.0");
mcpResPurchaseCC.setCardholderAmount("500");
mcpResPurchaseCC.setCardholderCurrencyCode("840");
mcpResPurchaseCC.setMCPRateToken("P1538679861174342");
//Mandatory - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");

mcpResPurchaseCC.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpResPurchaseCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Masked Pan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
System.out.println("IssuerId = " + receipt.getIssuerId());

System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
System.out.println("MCPRate = " + receipt.getMCPRate());
System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
System.out.println("HostId = " + receipt.getHostId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

9.11 MCP Pre-Authorization with Vault

This transaction uses the data key to identify a previously registered credit card profile in Vault. The details saved within the profile are then submitted to perform a Pre-Authorization transaction.

The data key may be a temporary one generated used Hosted Tokenization, or may be a permanent one from the Vault.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

MCP Pre-Authorization with Vault transaction object definition

```
MCPResPreauthCC mcpResPreauthCC = new MCPResPreauthCC();
```

HttpsPostRequest object for MCP Pre-Authorization with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpResPreauthCC);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_ token);`

MCP Pre-Authorization with Vault transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`mcpResPreauthCC.setData (data_key);`
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpResPreauthCC.setOrderId (order_id);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpResPreauthCC.setCryptType (crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpResPreauthCC.setMCPVersion ("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of foreign currency	`mcpResPreauthCC .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpResPreauthCC .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Pre-Authorization with Vault transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`mcpResPreauthCC.setCustId (cust_id);`
dynamic descriptor	*String* max 20-character alphanumeric total of 22 characters including your merchant name and separator	`mcpResPreauthCC .setDynamicDescriptor (dynamic_descriptor);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. For information about fields in the Credential on FIle Info object, see Credential on File Info Object and Variables.	*Object* N/A	`mcpResPreauthCC.setCofInfo (cof);`

Variable Name	Type and Limits	Set Method
AVS Information	*Object* N/A	`mcpResPreauthCC.setAvsInfo (avsCheck);`
CVD Information	*Object* N/A	`mcpResPreauthCC.setCvdInfo (cvdCheck);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpResPreauthCC .setMCPRateToken("MCP_RATE_ TOKEN");`

Sample MCP Pre-Authorization with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPResPreauthCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "rS7DbroQHJmJxdBfXFXiauQc4";
String amount = "1.00";
String cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be
used
String crypt_type = "1";
String dynamic_descriptor = "my descriptor";
String processing_country_code = "CA";
String expdate = "1712"; //For Temp Token
boolean status_check = false;
MCPResPreauthCC mcpResPreauthCC = new MCPResPreauthCC();
mcpResPreauthCC.setDataKey(data_key);
mcpResPreauthCC.setOrderId(order_id);
mcpResPreauthCC.setCustId(cust_id);
mcpResPreauthCC.setAmount(amount);
mcpResPreauthCC.setCryptType(crypt_type);
mcpResPreauthCC.setDynamicDescriptor(dynamic_descriptor);
//mcpResPreauthCC.setExpDate(expdate); //Temp Tokens only

//MCP Fields
mcpResPreauthCC.setMCPVersion("1.0");
mcpResPreauthCC.setCardholderAmount("500");
mcpResPreauthCC.setCardholderCurrencyCode("840");
mcpResPreauthCC.setMCPRateToken("P1538681661706745");

//Mandatory - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");
mcpResPreauthCC.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpResPreauthCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
System.out.println("IsCorporate = " + receipt.getCorporateCard());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Masked Pan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());
System.out.println("IssuerId = " + receipt.getIssuerId());

System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
System.out.println("MCPRate = " + receipt.getMCPRate());
System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
System.out.println("HostId = " + receipt.getHostId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

9.12 MCP Independent Refund with Vault

This transaction uses the data key to identify a previously registered credit card profile in Vault. The details saved within the profile are then submitted to perform an Independent Refund transaction.

This transaction request is the multi-currency pricing (MCP) enabled version of the equivalent financial transaction.

Things to Consider:

- Because of the potential for fraud, permission for this transaction is not granted to all accounts by default. If it is required for your business, it must be requested via your account manager.

MCP Independent Refund with Vault transaction object definition

```
MCPResIndRefundCC mcpResIndRefundCC = new MCPResIndRefundCC();
```

HttpsPostRequest object for MCP Independent Refund with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(mcpResIndRefundCC);
```

Core connection object fields (all API transactions)

Variable Name	Type and Limits	Set Method
store ID	*String* N/A	`mpgReq.setStoreId(store_id);`
API token	*String* N/A	`mpgReq.setApiToken(api_ token);`

MCP Independent Refund with Vault transaction request fields – Required

Variable Name	Type and Limits	Set Method
data key	*String* 25-character alphanumeric	`mcpResIndRefundCC.setData (data_key);`
order ID	*String* 50-character alpha-numerica-Z A-Z 0-9 _ - : . @ spaces	`mcpResIndRefundCC.setOrderId (order_id);`
electronic commerce indic-ator	*String* 1-character alphanumeric	`mcpIndrefund.setCryptType (crypt);`

MCP-specific request fields – Required

Variable Name	Type and Limits	Set Method
MCP version number	*String* numeric current version is 1.0	`mcpResIndRefundCC .setMCPVersion("MCP_VERSION_NUM");`
cardholder amount	*String* 12-character numeric smallest discrete unit of foreign currency	`mcpResIndRefundCC .setCardholderAmount ("CARDHOLDER_AMOUNT");`
cardholder currency code	*String* 3-character numeric	`mcpResIndRefundCC .setCardholderCurrencyCode ("CARDHOLDER_CURRENCY_CODE");`

MCP Independent Refund with Vault transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`mcpResIndRefundCC.setCustId (cust_id);`

MCP-specific request fields – Optional

Variable Name	Type and Limits	Set Method
MCP rate token	*String* N/A	`mcpResIndRefundCC .setMCPRateToken("MCP_RATE_TOKEN");`

Sample MCP Independent Refund with Vault

```
package Canada;
import JavaAPI.*;
public class TestCanadaMCPResIndRefundCC
{
public static void main(String[] args)
{
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String store_id = "store5";
String api_token = "yesguy";
String data_key = "rS7DbroQHJmJxdBfXFXiauQc4";
String amount = "1.00";
String cust_id = "customer1";
String crypt_type = "1";
String processing_country_code = "CA";
boolean status_check = false;
MCPResIndRefundCC mcpResIndRefundCC = new MCPResIndRefundCC();
mcpResIndRefundCC.setDataKey(data_key);
mcpResIndRefundCC.setOrderId(order_id);
mcpResIndRefundCC.setCustId(cust_id);
mcpResIndRefundCC.setAmount(amount);
mcpResIndRefundCC.setCryptType(crypt_type);

//MCP Fields
mcpResIndRefundCC.setMCPVersion("1.0");
mcpResIndRefundCC.setCardholderAmount("500");
mcpResIndRefundCC.setCardholderCurrencyCode("840");
mcpResIndRefundCC.setMCPRateToken("R1538679861330690");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(mcpResIndRefundCC);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("DataKey = " + receipt.getDataKey());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("ResSuccess = " + receipt.getResSuccess());
System.out.println("PaymentType = " + receipt.getPaymentType());
System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
System.out.println("Cust ID = " + receipt.getResCustId());
System.out.println("Phone = " + receipt.getResPhone());
System.out.println("Email = " + receipt.getResEmail());
System.out.println("Note = " + receipt.getResNote());
System.out.println("Masked Pan = " + receipt.getResMaskedPan());
System.out.println("Exp Date = " + receipt.getResExpdate());
System.out.println("Crypt Type = " + receipt.getResCryptType());
System.out.println("Avs Street Number = " + receipt.getResAvsStreetNumber());
System.out.println("Avs Street Name = " + receipt.getResAvsStreetName());
System.out.println("Avs Zipcode = " + receipt.getResAvsZipcode());

System.out.println("MerchantSettlementAmount = " + receipt.getMerchantSettlementAmount());
System.out.println("CardholderAmount = " + receipt.getCardholderAmount());
System.out.println("CardholderCurrencyCode = " + receipt.getCardholderCurrencyCode());
System.out.println("MCPRate = " + receipt.getMCPRate());
System.out.println("MCPErrorStatusCode = " + receipt.getMCPErrorStatusCode());
System.out.println("MCPErrorMessage = " + receipt.getMCPErrorMessage());
System.out.println("HostId = " + receipt.getHostId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

9.13 MCP Currency Codes

Numeric Currency Code (ISO)	Currency Name/Acronym
032	Argentine Peso (ARS)
036	Australian Dollar (AUD)
048	Bahraini Dinar (BHD)
052	Barbados Dollar (BBD)
060	Bermudian Dollar (BMD)
084	Belize Dollar (BZD)
096	Brunei Dollar (BND)
136	Cayman Islands Dollar (KYD)
144	Sri Lanka Rupee (LKR)
152	Chilean Peso (CLP)
156	Chinese Yuan (CNY)
170	Colombian Peso (COP)
188	Costa Rican Colon (CRC)
191	Croatian Kuna (HRK)
203	Czech Koruna (CZK)
208	Danish Krone (DKK)
214	Dominican Republic Peso
242	Fiji Dollar (FJD)
320	Guatemala Quetzal (GTQ)
344	Hong Kong Dollar (HKD)
348	Hungarian Forint (HUF)
352	Iceland Krona (ISK)

Numeric Currency Code (ISO)	Currency Name/Acronym
356	Indian Rupee (INR)
360	Indonesian Rupiah (IDR)
376	Israeli Shekel (ILS)
388	Jamaican Dollar (JMD)
392	Japanese Yen (JPY)
400	Jordanian Dinar (JOD)
404	Kenyan Shilling (KES)
410	South Korean Won (KRW)
414	Kuwaiti Dinar (KWD)
458	Malaysian Ringgit (MYR)
480	Mauritius Rupee (MUR)
484	Mexican Peso (MXN)
504	Moroccan Dirham (MAD)
512	Omani Rial (OMR)
532	Netherlands Antillean Guilder (ANG)
533	Aruban Guilder (AWG)
548	Vanuatu Vatu (VUV)
554	New Zealand Dollar (NZD)
558	Nicaraguan Cordoba (NIO)
578	Norwegian Krone (NOK)
586	Pakistan Rupee (PKR)
598	Papua New Guinean Kina (PGK)
600	Paraguayan Guarani (PYG)

Numeric Currency Code (ISO)	Currency Name/Acronym
604	Peruvian Nuevo Sol (PEN)
608	Philippine Peso (PHP)
634	Qatari Rial (QAR)
643	Russian Ruble (RUB)
682	Saudi Riyal (SAR)
702	Singapore Dollar (SGD)
704	Vietnamese Dong (VND)
710	South African Rand (ZAR)
752	Swedish Krona (SEK)
756	Swiss Franc (CHF)
764	Thai Baht (THB)
780	Trinidad & Tobago Dollar (TTD)
784	UAE Dirham (AED)
788	Tunisian Dinar (TND)
818	Egyptian Pound (EGP)
826	UK Pound Sterling (GBP)
840	US Dollar (USD)
858	Uruguayan Peso (UYU)
901	New Taiwan Dollar (TWD)
946	Romanian New Leu (RON)
949	New Turkish Lira (TRY)
951	East Caribbean Dollar (XCD)
953	CFP Franc (XPF)

Numeric Currency Code (ISO)	Currency Name/Acronym
975	Bulgarian Lev (BGN)
978	Euro (EUR)
985	Polish New Zloty (PLN)
986	Brazilian Real (BRL)

9.14 MCP Error Codes

Error Code	Description
200	OK (there will be no value returned in the MCP error message)
500	Upstream error
1000	Invalid JSON format
1003	Invalid txnType detected: <invalid txnType> please enter PURCHASE or REFUND
1005	Invalid rateInquiryId-txnType combination.
1007	Warning: at least one of cardHolderCurrency or merchantSettlementCurrency must be non-zero.
1008	Card-holder amount must be non-zero.
1009	Negative amounts detected
1010	Unsupported cardholder currency detected: <unsupported currency>
1015	invalid rateInquiryId
1016	Unsupported merchant id

10 e-Fraud Tools

10.1 Address Verification Service

10.1.1 About Address Verification Service (AVS)

Address Verification Service (AVS) is an optional fraud-prevention tool offered by issuing banks whereby a cardholder's address is submitted as part of the transaction authorization. The AVS address is then compared to the address kept on file at the issuing bank. AVS checks whether the street number, street name and zip/postal code match. The issuing bank returns an AVS result code indicating whether the data was matched successfully. Regardless of the AVS result code returned, the credit card is authorized by the issuing bank.

The response that is received from AVS verification is intended to provide added security and fraud prevention, but the response itself does not affect the completion of a transaction. Upon receiving a response, the choice to proceed with a transaction is left entirely to the merchant. The responses is **not** a strict guideline of whether a transaction will be approved or declined.

The following transactions support AVS:

- Purchase (Basic and Mag Swipe)
- Pre-Authorization (Basic)
- Re-Authorization (Basic)
- ResAddCC (Vault)
- ResUpdateCC (Vault)

Things to Consider:

- AVS is supported by Visa, MasterCard, American Express, Discover and JCB.
- When testing AVS, you must **only** use the Visa test card numbers 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at the Moneris developer portal (https://developer-.moneris.com).
- Store ID “store5” is set up to support AVS testing.

10.1.2 AVS Info Object

AVSInfo object definition

```
AvsInfo avsCheck = new AvsInfo();
```

Transaction object set method

```
<transaction>.setAvsInfo(avsCheck);
```

Variable Name	Type and Limits	Set Method	Description
AVS street number	*String* 19-character alphanumeric **NOTE:** this character limit is a combined total allowed for AVS street number and AVS street name	`avsCheck.setAvsStreetNumber ("212");`	Cardholder street number
AVS street name	*String* 19-character alphanumeric **NOTE:** this character limit is the combined total allowed for AVS street number and AVS street name	`avsCheck.setAvsStreetName ("Payton Street");`	Cardholder street name
AVS zip/postal code	*String* 9-character alphanumeric	`avsCheck.setAvsZipCode ("M1M1M1");`	Cardholder zip/postal code

10.1.3 AVS Response Codes

Below is a full list of possible AVS response codes. These can be returned when you call the `receipt.getAvsResultCode()` method .

Value	Visa	Mastercard/Discover	American Express/ JCB
A	Street address matches, zip/postal code does not; acquirer rights not implied	Address matches, zip/ postal code does not	Billing address matches, zip/postal code does not
B	Street address matches; zip/postal code not verified due to incompatible formats (acquirer sent both street address and zip/postal code)	N/A	N/A
C	Street address not verified due to incompatible formats (acquirer sent both street address and zip/postal code)	N/A	N/A
D	Street address and zip/-postal code match	N/A	Customer name incorrect; zip/postal code matches
E	N/A	N/A	Customer name incorrect, billing address and zip/-postal code match
F	Applies to UK only: Street address and zip/postal code match	N/A	Customer name incorrect; billing address matches
G	Address information not verified for international transaction Any of following may be true: • Issuer is not an AVS participant, or • AVS data was present in the request but issuer did not return an AVS result, or	N/A	N/A

Value	Visa	Mastercard/Discover	American Express/ JCB
	• Visa performs AVS on behalf of the issuer and there was no address record on file for this account		
I	Address information not verified	N/A	N/A
K	N/A	N/A	Customer name matches
L	N/A	N/A	Customer name and postal code match
M	Street address and zip/-postal code match	N/A	Customer name, billing address, and zip/postal code match
N	No match; acquirer sent: • postal/ZIP code only, or • street address only, or • both postal code and street address Also used when acquirer requests AVS but sends no AVS data	Neither address nor zip/-postal code matches	Billing address and zip/-postal code do not match
O	N/A	N/A	Customer name and billing address match
P	Postal code match; acquirer sent both postal code and street address, but street address not veri-fied due to incompatible formats	N/A	N/A
R	Retry; system unavailable or timed out	Retry; system unable to process	System unavailable; retry

Value	Visa	Mastercard/Discover	American Express/ JCB
	Issuer ordinarily performs AVS but was unavailable. **NOTE:** Code R is used by Visa when issuers are unavailable; issuers should refrain from using this code.		
S	N/A	AVS currently not sup-ported	AVS currently not sup-ported
T	N/A	Nine-digit zip code matches; address does not match	N/A
U	Address not verified for domestic transaction, for any of the following reas-ons: • Issuer is not an AVS participant, or • AVS data was present in the request but issuer did not return an AVS result, or • Visa performs AVS on behalf of the issuer and there was no address record on file for this account	No data from issuer-/authorization system	Information is unavailable
W	Not applicable; if present, replaced with Z by Visa Available for U.S. issuers only	For U.S. addresses, nine-digit postal code matches, address does not For addresses outside the U.S., postal code matches, address does not	Customer name, billing address, and postal code are all correct matches
X	N/A	For U.S. addresses, nine-digit postal code and addresses matches	N/A

Value	Visa	Mastercard/Discover	American Express/ JCB
		For addresses outside the U.S., postal code and address match	
Y	Street address and postal code match	Billing address and postal code both match	Billing address and postal code both match
Z	Zip/postal code matches; street address does not match, or street address not included in request	For U.S. addresses, five-digit zip code matches, address does not match	Postal code matches, billing address does not

10.1.4 AVS Sample Code

This is a sample of Java code illustrating how AVS is implemented with a Purchase transaction. Purchase object information that is not relevant to AVS has been removed.

For more about Purchase transactions, see 2.1 Purchase.

Sample Purchase with AVS information

```
    AvsInfo avsCheck = new AvsInfo();
    avsCheck.setAvsStreetNumber("212");
    avsCheck.setAvsStreetName("Payton Street");
    avsCheck.setAvsZipCode("M1M1M1");
    avsCheck.setAvsEmail("test@host.com");
    avsCheck.setAvsHostname("hostname");
    avsCheck.setAvsBrowser("Mozilla");
    avsCheck.setAvsShiptoCountry("CAN");
    avsCheck.setAvsShipMethod("G");
    avsCheck.setAvsMerchProdSku("123456");
    avsCheck.setAvsCustIp("192.168.0.1");
    avsCheck.setAvsCustPhone("5556667777");

    Purchase purchase = new Purchase();
    purchase.setAvsInfo(avsCheck);
```

10.2 Card Validation Digits (CVD)

10.2.1 About Card Validation Digits (CVD)

The Card Validation Digits (CVD) value is an additional number printed on credit cards that is used as an additional check when verifying cardholder credentials during a transaction.

The response that is received from CVD verification is intended to provide added security and fraud prevention, but the response itself does not affect the completion of a transaction. Upon receiving a response, the choice whether to proceed with a transaction is left entirely to the merchant. The responses is **not** a strict guideline of which transaction will approve or decline.

The following transactions support CVD:

- Purchase (Basic, Vault and Mag Swipe)
- Pre-Authorization (Basic and Vault)
- Re-Authorization

Things to Consider:

- CVD is only supported by Visa, MasterCard, American Express, Discover, JCB and UnionPay.
- For UnionPay cards, the CVD response will not be returned; the issuer will approve or decline based on the CVD result.
- When testing CVD, you must **only** use the Visa test card numbers 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at the Moneris developer portal (https://developer-.moneris.com).
- Test store_id “store5” is set up to support CVD testing.

10.2.2 Transactions Where CVD Is Required

The Card Validation Digits (CVD) object is required in transaction requests in the following scenarios:

- Initial transactions when storing cardholder credentials in Credential on File scenarios; subsequent follow-on transactions do not use CVD
- Any Purchase, Pre-Authorization or Card Verification where you are not storing cardholder credentials

10.2.3 CVD Information Object

NOTE: The CVD value must only be passed to the Moneris Gateway. Under **no** circumstances may it be stored for subsequent uses or displayed as part of the receipt information.

CvdInfo object definition

```
CvdInfo cvdCheck = new CvdInfo();

$cvdTemplate = array(

'cvd_indicator' => $cvd_indicator,

'cvd_value' => $cvd_value

);

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
```

Transaction object set method

```
transaction.setCvdInfo(cvdCheck);

$mpgTxn->setCvdInfo($mpgCvdInfo);
```

Table 1 CVD Info Object – Required Fields

Variable Name	Type and Limits	Set Method	Description
CVD indicator	*String* 1-character numeric	`cvdCheck.setCvdIndicator ("1");`	Indicates presence of CVD Possible values: 0: CVD value is deliberately bypassed or is not provided by the merchant. 1: CVD value is present. 2: CVD value is on the card, but is illegible. 9: Cardholder states that the card has no CVD imprint.
CVD value	*String* 4-character	`cvdCheck.setCvdValue ("099");`	CVD value located on credit card

Variable Name	Type and Limits	Set Method	Description
	numeric		**NOTE:** The CVD value must only be passed to the Moneris Gateway. Under **no** circumstances may it be stored for subsequent uses or displayed as part of the receipt information.

10.2.4 CVD Result Codes

Value	Definition
M	Match
N	No match
P	Not processed
S	CVD should be on the card, but Merchant has indicated that CVD is not present
U	Issuer is not a CVD participant
Y	Match for Amex/JCB only
D	Invalid security code for Amex or JCB only
Other	Invalid response code

10.2.5 Sample Purchase with CVD Info Object

This is a sample of Java code illustrating how CVD is implemented with a Purchase transaction. Purchase object information that is not relevant to CVD has been removed.

Sample Purchase with CVD Information

```
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.setCvdIndicator("1");
cvdCheck.setCvdValue("099");
```

Sample Purchase with CVD Information

```
Purchase purchase = new Purchase();
purchase.setCvdInfo(cvdCheck);
```

10.3 Transaction Risk Management Tool

The Transaction Risk Management Tool (TRMT) is available to **Canadian integrations** only.

10.3.1 About the Transaction Risk Management Tool

The Transaction Risk Management Tool provides additional information to assist in identifying fraudulent transactions. To maximize the benefits from the Transaction Risk Management Tool, it is highly recommended that you:

- Carefully consider the business logic and processes that you need to implement surrounding the handling of response information the Transaction Risk Management Tool provides.
- Implement the other fraud tools available through Moneris Gateway (such as AVS, CVD, Verified by Visa, MasterCard SecureCode and American Express SafeKey).

10.3.2 Introduction to Queries

There are two types of transactions associated with the Transaction Risk Management Tool (TRMT):

- Session Query (page 345)
- Attribute Query (page 352)

The Session Query and Attribute Query are used at the time of the transaction to obtain the risk assessment.

Moneris recommends that you use the Session Query as much as possible for obtaining your risk assessment because it uses the device fingerprint as well as other transaction information when providing the risk scores.

To use the Session Query, you must implement two components:

- Tags on your website to collect the device fingerprinting information
- Session Query transaction.

If you are not able to collect the necessary information for the Session Query (such as the device fingerprint), then use the Attribute Query.

10.3.3 Session Query

Once a device profiling session has been initiated upon a client device, the Session Query API is used at the time of the transaction or even to obtain a device identifier or 'fingerprint', attribute list and risk assessment for the client device.

Session Query transaction object definition

```
SessionQuery sq = new SessionQuery();
```

HttpsPostRequest object for Session Query transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(sq);
```

Session Query transaction values

Table 37: Session Query transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Session ID	String	9-character decimal Permitted characters: [a-z], [A-Z], 0-9, _, -	`sq.setSessionId(session_id);`
	Web server session identifier generated when device profiling was initiated.		
Service type	String	9-character decimal	`sq.setServiceType(service_type);`
	Which output fields are returned. session -- returns IP and device related attributes.		
Event type	String	payment	`sq.setEventType(service_type);`
	Defines the type of transaction or event for reporting purposes. payment - Purchasing of goods/services.		
Credit card number (PAN)	String	20-character numeric No spaces or dashes	`sq.setPan(pan);`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		

Table 37: Session Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Account address street 1	String	32-character alphanumeric	`sq.setAccountAddressStreet1 ("3300 Bloor St W");`
	First portion of the street address component of the billing address.		
Account Address street 2	String	32-character alphanumeric	`sq.setAccountAddressStreet2("4th Flr West Tower");`
	Second portion of the street address component of the billing address.		
Account address city	String	50-character alphanumeric	`sq.setAccountAddressCity ("Toronto");`
	The city component of the billing address.		
Account address state/-province	String	64-character alphanumeric	`sq.setAccountAddressState ("Ontario");`
	The state/province component of the billing address.		
Account address coun-try	String	2-character alphanumeric	`sq.setAccountAddressCountry ("CA");`
	ISO2 country code of the billing addresses.		
Account address ZIP/-postal code	String	8-character alphanumeric	`sq.setAccountAddressZip ("M8X2X2");`
	ZIP/postal code of the billing address.		
Shipping address street 1	String	32-character alphanumeric	`sq.setShippingAddressStreet1 ("3300 Bloor St W");`
	First portion of the street address component of the shipping address.		
Shipping address street 2	String	32-character alphanumeric	`sq.setShippingAddressStreet2 ("4th Flr West Tower");`
	Second portion of the street address component of the shipping address.		
Shipping address city	String	50-character alphanumeric	`sq.setShippingAddressCity ("Toronto");`
	City component of the shipping address.		
Shipping address state/-province	String	64-character alphanumeric	`sq.setShippingAddressState ("Ontario");`
	The state/province component of the shipping address.		

Table 37: Session Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Shipping address country	String	2-character alphanumeric	`sq.setShippingAddressCountry ("CA");`
	ISO2 country code of the account address country.		
Shipping address ZIP	String	8-character alphanumeric	`sq.setAccountAddressZip ("M8X2X2");`
	The ZIP/postal code component of the shipping address.		
Local attribute 1-5	String	255-character alphanumeric	`sq.setLocalAttrib1("a");`
	These five attributes can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		
Transaction amount	String	255-character alphanumeric Must contain 2 decimal places	`sq.setTransactionAmount("1.00");`
	The numeric currency amount.		
Transaction currency	String	10-character numeric	`sq.setTransactionCurrency ("CAN");`
	The currency type that the transaction was denominated in. If TransactionAmount is passed, the TransactionCurrency is required. Values to be used are: • CAD – 124 • USD – 840		

Table 38: Session Query transaction object optional values

Value	Type	Limits	Set method
	Description		
Account login	String	255-character alphanumeric	`sq.setAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");`
	The Account Login name.		
Password hash	String	40-character alphanumeric	`sq.setPasswordHash ("489c830f10f7c601d30599a0deaf66e64d2aa50a");`
	The input must be a SHA-2 hash of the password in hexadecimal format. Used to check if it is on a watch list.		

Table 38: Session Query transaction object optional values (continued)

Value	Type	Limits	Set method
	Description		
Account number	String	255-character alphanumeric	`sq.setAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");`
	The account number for the account.		
Account name	String	255-character alphanumeric	`sq.setAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");`
	Account name (or concatenation of first and last name of account holder).		
Account email	String	100-character alphanumeric	`sq.setAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");`
	The email address entered into the form for this contact. Used to check if this is a high risk account email id.		
Account telephone	String	32-character alphanumeric	
	Contact telephone number including country and city codes. All whitespace is removed. Must be in format: 0..9,<space>,(,),[,] braces must be matched.		
Address street 1	String	32-character alphanumeric	
	The first portion of the street address component of the account address.		
Address street 2	String	32-character alphanumeric	
	The second portion of the street address component of the account address.		
Address city	String	50-character alphanumeric	
	The city component of the account address.		
Address state/province	String	64-character alphanumeric	
	The state/province component of the account address		
Address country	String	2-character alphanumeric	
	The 2 character ISO2 country code of the account address country		

Table 38: Session Query transaction object optional values (continued)

Value	Type	Limits	Set method
	Description		
Address ZIP	String	8-character alphanumeric	
	The ZIP/postal code of the account address.		
Ship Address Street 1	String	32-character alphanumeric	
	The first portion of the street address component of the shipping address		
Ship Address Street 2	String	32-character alphanumeric	
	The second portion of the street address component of the shipping address		
Ship Address City	String	50-character alphanumeric	
	The city component of the shipping address		
Ship Address State/Province	String	64-character alphanumeric	
	The state/province component of the shipping address		
Ship Address Country	String	2-character alphanumeric	
	The 2 character ISO2 country code of the shipping address country		
Ship Address ZIP	String	8-character alphanumeric	
	The ZIP/postal code of the shipping address		
CC Number Hash	String	255-character alphanumeric	
	This is a SHA-2 hash (in hexadecimal format) of the credit card number.		
Custom Attribute 1-8	String	255-character alphanumeric	
	These 8 attributes can be used to pass custom attribute data which can be used within the rules.		

Sample Session Query - CA

```
package Canada;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;
import JavaAPI.*;
public class TestCanadaRiskCheckSession
{
public static void main(String[] args)
{
String store_id = "moneris";
String api_token = "hurgle";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String session_id = "abc123";
String service_type = "session";
//String event_type = "LOGIN";
String processing_country_code = "CA";
boolean status_check = false;
SessionQuery sq = new SessionQuery();
sq.setOrderId(order_id);
sq.setSessionId(session_id);
sq.setServiceType(service_type);
sq.setEventType(service_type);
//sq.setPolicy("");
//sq.setDeviceId("4EC40DE5-0770-4fa0-BE53-981C067C598D");
sq.setAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");
sq.setPasswordHash("489c830f10f7c601d30599a0deaf66e64d2aa50a");
sq.setAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");
sq.setAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");
sq.setAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");

//sq.setAccountTelephone("5556667777");
sq.setPan("4242424242424242");
//sq.setAccountAddressStreet1("3300 Bloor St W");
//sq.setAccountAddressStreet2("4th Flr West Tower");
//sq.setAccountAddressCity("Toronto");
//sq.setAccountAddressState("Ontario");
//sq.setAccountAddressCountry("CA");
//sq.setAccountAddressZip("M8X2X2");
//sq.setShippingAddressStreet1("3300 Bloor St W");
//sq.setShippingAddressStreet2("4th Flr West Tower");
//sq.setShippingAddressCity("Toronto");
//sq.setShippingAddressState("Ontario");
//sq.setShippingAddressCountry("CA");
//sq.setShippingAddressZip("M8X2X2");
//sq.setLocalAttrib1("a");
//sq.setLocalAttrib2("b");
//sq.setLocalAttrib3("c");
//sq.setLocalAttrib4("d");
//sq.setLocalAttrib5("e");
//sq.setTransactionAmount("1.00");
//sq.setTransactionCurrency("840");
//set SessionAccountInfo
sq.setTransactionCurrency("CAN");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
```

Sample Session Query - CA

```
mpgReq.setTransaction(sq);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
String[] rules;
Hashtable<String, String> results = new Hashtable<String, String>();
Receipt receipt = mpgReq.getReceipt();
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("Message = " + receipt.getMessage());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
results = receipt.getRiskResult();
Iterator<Map.Entry<String, String>> response = results.entrySet().iterator();
while (response.hasNext())
{
Map.Entry<String, String> entry = response.next();
System.out.println(entry.getKey().toString() + " = " + entry.getValue().toString());
}
rules = receipt.getRiskRules();
for (int i = 0; i < rules.length; i++)
{
System.out.println("RuleName = " + rules[i]);
System.out.println("RuleCode = " + receipt.getRuleCode(rules[i]));
System.out.println("RuleMessageEn = " + receipt.getRuleMessageEn(rules[i]));
System.out.println("RuleMessageFr = " + receipt.getRuleMessageFr(rules[i]));
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

10.3.3.1 Session Query Transaction Flow

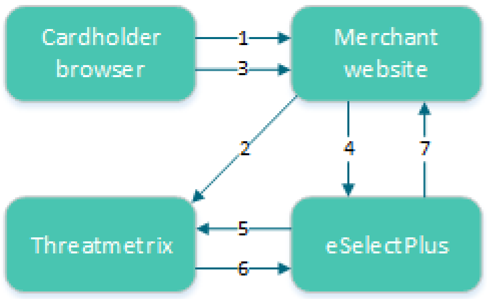


Figure 4: Session Query transaction flow

1. Cardholder logs onto the merchant website.
2. When the page has loaded in the cardholder's browser, special tags within the site allow information from the device to be gathered and sent to ThreatMetrix as the device fingerprint.

   The HTML tags should be placed where the cardholder is resident on the page for a couple of seconds to get the broadest data possible.

3. Customer submits a transaction.
4. Merchant's web application makes a Session Query transaction to the Moneris Gateway using the same session id that was included in the device fingerprint. This call must be made within 30 minutes of profiling (2).
5. Moneris Gateway submits the Session Query data to ThreatMetrix.
6. ThreatMetrix uses the Session Query data and the device fingerprint information to assess the transaction against the rules. A score is generated based on the rules.
7. The merchant uses the returned device information in its risk analysis to make a business decision. The merchant may wish to continue or cancel with the cardholder's payment trans-action.

10.3.4 Attribute Query

The Attribute Query is used to obtain a risk assessment of transaction-related identifiers such as the email address and the card number. Unlike the Session Query, the Attribute Query does not require the device fingerprinting information to be provided.

AttributeQuery transaction object definition

```
AttributeQuery aq = new AttributeQuery();
```

HttpsPostRequest object for AttributeQuery transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(aq);
```

Attribute Query transaction values

Table 39: Attribute Query transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Service type	String	N/A	`aq.setServiceType(service_type);`
	Which output fields are returned. session -- returns IP and device related attributes.		
Device ID	String	36-character alphanumeric	`aq.setDeviceId("");`
	Unique device identifier generated by a previous call to the ThreatMetrix session-query API.		
Credit card number	String	20-character numeric No spaces or dashes	`aq.setPan(pan);`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		

Table 39: Attribute Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
IP address	String	64-character alphanumeric	`aq.setIPAddress("192.168.0.1");`
	True IP address. Results will be returned as true_ip_geo, true_ip_score and so on.		
IP forwarded	String	64-character alphanumeric	`aq.setIPForwarded ("192.168.1.0");`
	The IP address of the proxy. If the IPAddress is supplied, results will be returned as proxy_ip_geo and proxy_ip_score. If the IP Address is not supplied, this IP address will be treated as the true IP address and results will be returned as true_ip_geo, true_ip_score and so on		
Account address street 1	String	32-character alphanumeric	`aq.setAccountAddressStreet1 ("3300 Bloor St W");`
	First portion of the street address component of the billing address.		
Account Address Street 2	String	32-character alphanumeric	`aq.setAccountAddressStreet2("4th Flr West Tower");`
	Second portion of the street address component of the billing address.		
Account address city	String	50-character alphanumeric	`aq.setAccountAddressCity ("Toronto");`
	The city component of the billing address.		
Account address state/-province	String	64-character alphanumeric	`aq.setAccountAddressState ("Ontario");`
	The state component of the billing address.		
Account address coun-try	String	2-character alphanumeric	`aq.setAccountAddressCountry ("CA");`
	ISO2 country code of the billing addresses.		
Account address zip/-postal code	String	8-character alphanumeric	`aq.setAccountAddressZip ("M8X2X2");`
	Zip/postal code of the billing address.		
Shipping address street 1	String	32-character alphanumeric	`aq.setShippingAddressStreet1 ("3300 Bloor St W");`
	Account address country		
Shipping Address Street 2	String	32-character alphanumeric	`aq.setShippingAddressStreet2 ("4th Flr West Tower");`
	Second portion of the street address component of the shipping address.		

Table 39: Attribute Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Shipping Address City	String	50-character alphanumeric	`aq.setShippingAddressCity ("Toronto");`
	City component of the shipping address.		
Shipping Address State/Province	String	64-character alphanumeric	`aq.setShippingAddressState ("Ontario");`
	State/Province component of the shipping address.		
Shipping Address Country	String	2-character alphanumeric	`aq.setShippingAddressCountry ("CA");`
	ISO2 country code of the account address country.		
Shipping Address zip/-postal code	String	8-character alphanumeric	`aq.setAccountAddressZip ("M8X2X2");`
	The zip/postal code component of the shipping address.		

Sample Attribute Query

```
    String store_id = "moneris";
    String api_token = "hurgle";
    java.util.Date createDate = new java.util.Date();
    String order_id = "Test"+createDate.getTime();
    String service_type = "session";
    String processing_country_code = "CA";
    boolean status_check = false;

    AttributeQuery aq = new AttributeQuery();
    aq.setOrderId(order_id);
    aq.setServiceType(service_type);
    aq.setDeviceId("");
    aq.setAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");
    aq.setPasswordHash("489c830f10f7c601d30599a0deaf66e64d2aa50a");
    aq.setAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");
    aq.setAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");
    aq.setAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");
    //aq.setCCNumberHash("4242424242424242");
    //aq.setIPAddress("192.168.0.1");
    //aq.setIPForwarded("192.168.1.0");
    aq.setAccountAddressStreet1("3300 Bloor St W");
    aq.setAccountAddressStreet2("4th Flr West Tower");
    aq.setAccountAddressCity("Toronto");
    aq.setAccountAddressState("Ontario");
    aq.setAccountAddressCountry("CA");
    aq.setAccountAddressZip("M8X2X2");
    aq.setShippingAddressStreet1("3300 Bloor St W");
    aq.setShippingAddressStreet2("4th Flr West Tower");
    aq.setShippingAddressCity("Toronto");
    aq.setShippingAddressState("Ontario");
    aq.setShippingAddressCountry("CA");
```

Sample Attribute Query

```
    aq.setShippingAddressZip("M8X2X2");

    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setTransaction(aq);
    mpgReq.send();

    try
    {
            String[] rules;
            Hashtable<String, String> results = new Hashtable<String, String>();
            Receipt receipt = mpgReq.getReceipt();
            System.out.println("ResponseCode = " + receipt.getResponseCode());
            System.out.println("Message = " + receipt.getMessage());
            System.out.println("TxnNumber = " + receipt.getTxnNumber());

            results = receipt.getRiskResult();
            Iterator<Map.Entry<String, String>> response = results.entrySet().iterator();
            while (response.hasNext())
            {
                Map.Entry<String, String> entry = response.next();
                System.out.println(entry.getKey().toString() + " = " + entry.getValue().toString());
            }
            rules = receipt.getRiskRules();
            for (int i = 0; i < rules.length; i++)
            {
                System.out.println("RuleName = " + rules[i]);
                System.out.println("RuleCode = " + receipt.getRuleCode(rules[i]));
                System.out.println("RuleMessageEn = " + receipt.getRuleMessageEn(rules[i]));
                System.out.println("RuleMessageFr = " + receipt.getRuleMessageFr(rules[i]));
            }
    }
```

10.3.4.1 Attribute Query Transaction Flow

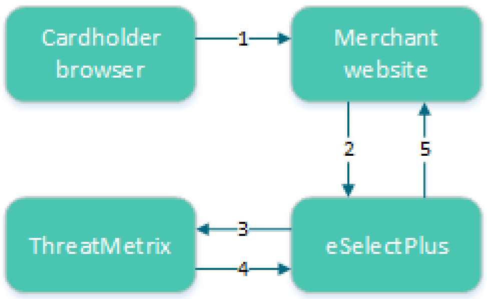


Figure 5: Attribute query transaction flow

1. Cardholder logs onto merchant website and submits a transaction.
2. The merchant’s web application makes an Attribute Query transaction that includes the session ID to the Moneris Gateway.
3. Moneris Gateway submits Attribute Query data to ThreatMetrix.
4. ThreatMetrix uses the Attribute Query data to assess the transaction against the rules. A score is generated based on the rules.
5. The merchant uses the returned device information in its risk analysis to make a business decision. The merchant may wish to continue or cancel with the cardholder's payment transaction.

10.3.5 Handling Response Information

When reviewing the response information and determining how to handle the transaction, it is recommended that you (either manually or through automated logic on your site) use the following pieces of information:

- Risk score
- Rules triggered (such as Rule Codes, Rule Names, Rule Messages)
- Results obtained from Verified by Visa, MasterCard Secure Code, AVS, CVD and the financial transaction authorization
- Response codes for the Transaction Risk Management Transaction that are included by automated processes.

10.3.5.1 TRMT Response Fields

Table 40: Receipt object response values for TRMT

Value	Type	Limits	Get method
	Definition		
Response Code	String	3-character alpha-numeric	`receipt.getResponseCode();`
	001 – Success 981 – Data error 982 – Duplicate Order ID 983 – Invalid Transaction 984 – Previously asserted 985 – Invalid activity description 986- Invalid impact description 987 – Invalid Confidence description 988 - Cannot find Previous		
Message	String	N/A	`receipt.getMessage();`
	Response message		
Event type	String	N/A	
	Type of transaction or event returned in the response.		

Table 40: Receipt object response values for TRMT (continued)

Value	Type	Limits	Get method
	Definition		
Org ID	String	N/A	
	ThreatMetrix-defined unique transaction identifier		
Policy	String	N/A	
	Policy used for the Session Query will be returned with the return request. If the Policy was not included, then the Policy name default is returned.		
Policy score	String	N/A	
	The sum of all the risks weights from triggered rules within the selected policy in the range [-100...100]		
Request duration	String	N/A	
	Length of time it takes for the transaction to be processed.		
Request ID	String	N/A	
	Unique number and will always be returned with the return request.		
Request result	String	N/A	`receipt.getRiskResult();`
	See 10.3.5.1 (page 356).		
Review status	String	N/A	
	The transaction status based on the assessments and risk scores.		
Risk rating	String	N/A	
	The rating based on the assessments and risk scores.		
Service type	String	N/A	
	The service type will be returned in the attribute query response.		
Session ID	String	N/A	
	Temporary identifier unique to the visitor will be returned in the return request.		
Summary risk score	String	N/A	
	Based on all of the returned values in the range [-100 ... 100]		
Transaction ID	String	N/A	
	This is the transaction identifier and will always be returned in the response when supplied as input.		
Unknown session	String	N/A	
	If present, the value is "yes". It indicates the session ID that was passed was not found.		

Table 41: Response code descriptions

Value	Definition
001	Success
981	Data error
982	Duplicate order ID
983	Invalid transaction
984	Previously asserted
985	Invalid activity description
986	Invalid impact description
987	Invalid confidence description
988	Cannot find previous

Table 42: Request result values and descriptions

Value	Definition
fail_duplicate_entities_of_same_type	More than one entity of the same was specified, e.g. password_hash was specified twice.
fail_incomplete	ThreatMetrix was unable to process the request due to incomplete or incorrect input data
fail_invalid_account_number	The format of the supplied account number was invalid
fail_invalid_characters	Invalid characters submitted
fail_invalid_charset	The value of character set was invalid
fail_invalid_currency_code	The format of the currency_code was invalid
fail_invalid_currency_format	The format of the currency_format was invalid
fail_invalid_telephone_number	Format of the supplied telephone number was invalid
fail_access	ThreatMetrix was unable to process the request because of API verification failing
fail_internal_error	ThreatMetrix encountered an error while processing the request

Value	Definition
fail_invalid_device_id	Format of the supplied device_id was invalid
fail_invalid_email_address	Format of the supplied email address was invalid
fail_invalid_fuzzy_device_id	The format of fuzzy_device_id was invalid
fail_invalid_ip_address_parameter	Format of a supplied ip_address parameter was invalid
fail_invalid_parameter	The format of the parameter was invalid, or the value is out of boundary
fail_invalid_sha_hash	The format of a parameter specified as a sha hash was invalid, sha hash included sha1/2/3 hash
fail_invalid_submitter_id	The format of the submitter id was invalid or the value is out of boundary
fail_no_policy_configured	No policy was configured against the org_id
fail_not_enough_params	Not enough device attributes were collected during profiling to perform a fingerprint match
fail_parameter_overlength	The value of the parameter was overlength
fail_temporarily_unavailable	Request failed because the service is temporarily unavailable
fail_too_many_instances_of_same_parameter	Multiple values for some parameters which only allow one instance
fail_verification	API query limit reached
success	ThreatMetrix was able to process the request successfully

10.3.5.2 Understanding the Risk Score

For each Session Query or Attribute Query, a score with a value between -100 and +100 is returned based on the rules that were triggered for the transaction.

Table 43 defines the risk scores ranges.

Table 43: Session Query and Attribute Query risk score definitions

Risk score	Visa definition
-100 to -1	A lower score indicates a higher probability that the transaction is fraudulent.
0	Neutral transaction
1 to 100	A higher score indicates a lower probability that the transaction is fraudulent. **Note**: All e-commerce transactions have some level of risk associated with them. Therefore, it is rare to see risk score in the high positive values.

When evaluating the risk of a transaction, the risk score gives an initial indicator of the potential risk that the transaction is fraudulent. Because some of the rules that are evaluated on each transaction may not be relevant to your business scenario, review the rules that were triggered for the transaction before determining how to handle the transaction.

10.3.5.3 Understanding the Rule Codes, Rule Names and Rule Messages

The rule codes, rule names and rule messages provide details about what rules were triggered during the assessment of the information provided in the Session or Attribute Query. Each rule code has a rule name and rule message. The rule name and rule message are typically similar. Table 44 provides additional information on each rule.

When evaluating the risk of a transaction, it is recommended that you review the rules that were triggered for the transaction and assess the relevance to your business. (That is, how does it relate to the typical buying habits of your customer base?)

If you are automating some or all of the decision-making processes related to handling the responses, you may want to use the rule codes. If you are documenting manual processes, you may want to refer to the more user-friendly rule name or rule message.

Table 44: Rule names, numbers and messages

Rule name	Rule number	Rule message
	Rule explanation	
White lists		
DeviceWhitelisted	WL001	Device White Listed
	Device is on the white list. This indicates that the device has been flagged as always "ok". **Note**: This rule is currently not in use.	
IPWhitelisted	WL002	IP White Listed
	IP address is on the white list. This indicates the device has been flagged as always "ok". **Note**: This rule is currently not in use.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
EmailWhitelisted	WL003	Email White Listed
	Email address is on the white list. This indicates that the device has been flagged as always "ok". **Note**: This rule is currently not in use.	
Event velocity		
2DevicePayment	EV003	2 Device Payment Velocity
	Multiple payments were detected from this device in the past 24 hours.	
2IPPaymentVelocity	EV006	2 IP Payment Velocity
	Multiple payments were detected from this IP within the past 24 hours.	
2ProxyPaymentVelocity	EV008	2 Proxy Payment Velocity
	The device has used 3 or more different proxies during a 24 hour period. This could be a risk or it could be someone using a legitimate corporate proxy.	
Email		
3EmailPerDeviceDay	EM001	3 Emails for the Device ID in 1 Day
	This device has presented 3 different email IDs within the past 24 hours.	
3EmailPerDeviceWeek	EM002	3 emails for the Device ID in 1 week
	This device has presented 3 different email IDs within the past week.	
3DevciePerEmailDay	EM003	3 Device Ids for email address in 1 day
	This email has been presented from three different devices in the past 24 hours.	
3DevciePerEmailWeek	EM004	3 Device Ids for email address in 1 week
	This email has been presented from three different devices in the past week.	
EmailDistanceTravelled	EM005	Email Distance Travelled
	This email address has been associated with different physical locations in a short period of time.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
3EmailPerSmartIDHour	EM006	3 Emails for SmartID in 1 Hour
	The SmartID for this device has been associated with 3 different email addresses in 1 hour.	
GlobalEMailOverOneMonth	EM007	Global Email over 1 month
	The e-mail address involved in the transaction over 30 days ago. This generally indicates that the transaction is less risky. **Note**: This rule is set so that it does not impact the policy score or risk rating.	
ComputerGeneratedEmailAddress	EM008	Computer Generated Email Address
	This transaction used a computer-generated email address.	
Account Number		
3AccountNumberPerDeviceDay	AN001	3 Account Numbers for device in 1 day
	This device has presented 3 different user accounts within the past 24 hours.	
3AccountNumberPerDeviceWeek	AN002	3 Account Numbers for device in 1 week
	This device has presented 3 different user accounts within the past week.	
3DevciePerAccountNumberDay	AN003	3 Device IDs for account number in 1 day
	This user account been used from three different devices in the past 24 hours.	
3DevciePerAccountNumberWeek	AN004	3 Device IDs for account number in 1 week
	This card number has been used from three different devices in the past week.	
AccountNumberDistanceTravelled	AN005	Account Number distance travelled
	This card number has been used from a number of physically different locations in a short period of time.	
Credit card/payments		
3CreditCardPerDeviceDay	CP001	3 credit cards for device in 1 day
	This device has used three credit cards within 24 hours.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
3CreditCardPerDeviceWeek	CP002	3 credit cards for device in 1 week
	This device has used three credit cards within 1 week.	
3DevicePerCreditCardDay	CP003	3 device ids for credit card in 1 day
	This credit card has been used on three different devices in 24 hours.	
3DevciePerCreditCardWeek	CP004	3 device ids for credit card in 1 week
	This credit card has been used on three different devices in 1 week.	
CredtCardDistanceTravelled	CP005	Credit Card has travelled
	The credit card has been used at a number of physically different locations in a short period of time.	
CreditCardShipAddressGeoMismatch	CP006	Credit Card and Ship Address do not match
	The credit card was issued in a region different from the Ship To Address information provided.	
CreditCardBillAddressGeoMismatch	CP007	Credit Card and Billing Address do not match
	The credit card was issued in a region different from the Billing Address information provided.	
CreditCardDeviceGeoMismatch	CP008	Credit Card and device location do not match
	The device is located in a region different from where the card was issued.	
CreditCardBINShipAddressGeoMismatch	CP009	Credit Card issuing location and Shipping address do not match
	The credit card was issued in a region different from the Ship To Address information provided.	
CreditCardBINBillAddressGeoMismatch	CP010	Credit Card issuing location and Billing address do not match
	The credit card was issued in a region different from the Billing Address information provided.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
CreditCardBINDeviceGeoMismatch	CP011	Credit Card issuing location and location of the device do not match
	The device is located in a region different from where the card was issued.	
TransactionValueDay	CP012	Daily Transaction Value Threshold
	The transaction value exceeds the daily threshold.	
TransactionValueWeek	CP013	Weekly Transaction Value Threshold
	The transaction value exceeds the weekly threshold.	
Proxy rules		
3ProxyPerDeviceDay	PX001	3 Proxy Ips in 1 day
	This device has used three different proxy servers in the past 24 hours.	
AnonymousProxy	PX002	Anonymous Proxy IP
	This device is using an anonymous proxy	
UnusualProxyAttributes	PX003	Unusual Proxy Attributes
	This transaction is coming from a source with unusual proxy attributes.	
AnonymousProxy	PX004	Anonymous Proxy
	This device is connecting through an anonymous proxy connection.	
HiddenProxy	PX005	Hidden Proxy
	This device is connecting via a hidden proxy server.	
OpenProxy	PX006	Open Proxy
	This transaction is coming from a source that is using an open proxy.	
TransparentProxy	PX007	Transparent Proxy
	This transaction is coming from a source that is using a transparent proxy.	
DeviceProxyGeoMismatch	PX008	Proxy and True GEO Match
	This device is connecting through a proxy server that didn’t match the devices geo-location.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
ProxyTrueISPMismatch	PX009	Proxy and True ISP Match
	This device is connecting through a proxy server that doesn’t match the true IP address of the device.	
ProxyTrueOrganizationMismatch	PX010	Proxy and True Org Match
	The Proxy information and True ISP information for this source do not match.	
DeviceProxyRegionMismatch	PX011	Proxy and True Region Match
	The proxy and device region location information do not match.	
ProxyNegativeReputation	PX012	Proxy IP Flagged Risky in Reputation Network
	This device is connecting from a proxy server with a known negative reputation.	
SatelliteProxyISP	PX013	Satellite Proxy
	This transaction is coming from a source that is using a satel-lite proxy.	
GEO		
DeviceCountriesNotAllowed	GE001	True GEO in Countries Not Allowed blacklist
	This device is connecting from a high-risk geographic loc-ation.	
DeviceCountriesNotAllowed	GE002	True GEO in Countries Not Allowed (negative whitelist)
	The device is from a region that is not on the whitelist of regions that are accepted.	
DeviceProxyGeoMismatch	GE003	True GEO different from Proxy GEO
	The true geographical location of this device is different from the proxy geographical location.	
DeviceAccountGeoMismatch	GE004	Account Address different from True GEO
	This device has presented an account billing address that doesn't match the devices geolocation.	
DeviceShipGeoMismatch	GE005	Device and Ship Geo mismatch
	The location of the device and the shipping address do not match.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
DeviceShipGeoMismatch	GE006	Device and Ship Geo mismatch
	The location of the device and the shipping address do not match.	
Device		
SatelliteISP	DV001	Satellite ISP
	This transaction is from a source that is using a satellite ISP.	
MidsessionChange	DV002	Session Changed Mid-session
	This device changed session details and identifiers in the middle of a session.	
LanguageMismatch	DV003	Language Mismatch
	The language of the user does not match the primary language spoken in the location where the True IP is registered.	
NoDeviceID	DV004	No Device ID
	No device ID was available for this transaction.	
Dial-upConnection	DV005	Dial-up connection
	This device uses a less identifiable dial-up connection.	
DeviceNegativeReputation	DV006	Device Blacklisted in Reputational Network
	This device has a known negative reputation as reported to the fraud network.	
DeviceGlobalBlacklist	DV007	Device on the Global Black List
	This device has been flagged on the global blacklist of known problem devices.	
DeviceCompromisedDay	DV008	Device compromised in last day
	This device has been reported as compromised in the last 24 hours.	
DeviceCompromisedHour	DV009	Device compromised in last hour
	This device has been reported as compromised in the last hour.	
FlashImagesCookiesDisabled	DV010	Flash Images Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	

Table 44: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
FlashCookiesDisabled	DV011	Flash Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	
FlashDisabled	DV012	Flash Disabled
	Key browser functions/identifiers have been disabled on this device.	
ImagesDisabled	DV013	Images Disabled
	Key browser functions/identifiers have been disabled on this device.	
CookiesDisabled	DV014	Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	
DeviceDistanceTravelled	DV015	Device Distance Travelled
	The device has been used from multiple physical locations in a short period of time.	
PossibleCookieWiping	DV016	Cookie Wiping
	This device appears to be deleting cookies after each session.	
PossibleCookieCopying	DV017	Possible Cookie Copying
	This device appears to be copying cookies.	
PossibleVPNConnection	DV018	Possibly using a VPN Connection
	This device may be using a VPN connection	

10.3.5.4 Examples of Risk Response

Session Query

Sample Risk Response - Session Query

```
<?xml version="1.0"?>
<response>
<receipt>
<ResponseCode>001</ResponseCode>
<Message>Success</Message>
<Result>
<session_id>abc123</session_id>
```

Sample Risk Response - Session Query

```
<unknown_session>yes</unknown_session>
<event_type>payment</event_type>
<service_type>session</service_type>
<policy_score>-25</policy_score>
<transaction_id>riskcheck42</transaction_id>
<org_id>11kue096</org_id>
<request_id>91C1879B-33D4-4D72-8FCB-B60A172B3CAC</request_id>
<risk_rating>medium</risk_rating>
<request_result>success</request_result>
<summary_risk_score>-25</summary_risk_score>
<Policy>default</policy>
<review_status>review</review_status>
</Result>
<Rule>
<RuleName>ComputerGeneratedEMail</RuleName>
<RuleCode>UN001</RuleCode>
<RuleMessageEn>Unknown Rule</RuleMessageEn>
<RuleMessageFr>Regle Inconnus</RuleMessageFr>
</Rule>
<Rule>
<RuleName>NoDeviceID</RuleName>
<RuleCode>DV004</RuleCode>
<RuleMessageEn>No Device ID</RuleMessageEn>
<RuleMessageFr>null</RuleMessageFr>
</Rule>
</receipt>
</response>
```

Attribute Query

Sample Risk Response - Attribute Query

```
<?xml version="1.0"?>
<response>
<receipt>
<ResponseCode001</ReponseCode>
<Message = Success</Message>
<Result>
<org_id>11kue096</org_id>
<request_id>443D7FB5-CC5C-4917-A57E-27EAC824069C</request_id>
<service_type>session</service_type>
<risk_rating>medium</risk_rating>
<summary_risk_score>-25</summary_risk_score>
<request_result>success</request_result>
<policy>default</policy>
<policy_score>-25</policy_score>
<transaction_id>riskcheck19</transaction_id>
<review_status>review</review_status>
</Result>
<Rule>
<RuleName>ComputerGeneratedEMail</RuleName>
<RuleCode>UN001</RuleCode>
<RuleMessageEn>Unknown Rule</RuleMessageEn>
<RuleMessageFr>Regle Inconnus</RuleMessageFr>
</Rule>
<Rule>
<RuleName>NoDeviceID</RuleName>
```

Sample Risk Response - Attribute Query

```
    <RuleCode>DV004</RuleCode>
    <RuleMessageEn>No Device ID</RuleMessageEn>
    <RuleMessageFr>null</RuleMessageFr>
    </Rule>
    </receipt>
    </response>
```

10.3.6 Inserting the Profiling Tags Into Your Website

Place the profiling tags on an HTML page served by your web application such that ThreatMetrix can collect device information from the customer's web browser. The tags must be placed on a page that a visitor would display in a browser window for 3-5 seconds (such as a page that requires a user to input data). After the device is profiled, a Session Query may be used to obtain the detail device information for risk assessment before submitting a financial payment transaction.

There are two profiling tags that require two variables. Those tags are `org_id` and `session_id`. `session_id` must match the session ID value that is to be passed in the Session Query transaction. The valid `org_id` values are:

11kue096
QA testing environment.

lbhqgx47
Production environment.

Below is an HTML sample of the profiling tags.

NOTE: Your site must replace `<my_session_id>` in the sample code with a unique alphanumeric value each time you fingerprint a new customer.

```
<p style="background:url(https://h.online-metrix.net/fp/clear.png?org_id=11kue096&session_id=<my_
    session_id>&m=1)">
</p>

<img src="https://h.onlinemetrix.net/fp/clear.png?org_id=11kue096&session_id=<my_session_id>&m=2" alt=""
    >

<script src="https://h.onlinemetrix.net/fp/check.js?org_id=11kue096&session_id=<my_session_id>"
type="text/javascript">
</script>

<object type="application/x-shockwave-flash"

data="https://h.onlinemetrix.net/fp/fp.swf?org_id=11kue096&session_id=<my_session_id>"
width="1" height="1" id="obj_id">
<param name="movie"
value="https://h.onlinemetrix.net/fp/fp.swf?org_id=11kue096&session_id=<my_session_id>" />
<div></div>
</object>
```

10.4 Encorporating All Available Fraud Tools

To minimize fraudulent activity in online transactions, Moneris recommends that you implement all of the fraud tools available through the Moneris Gateway. These are explained below:

Address Verification Service (AVS)
Verifies the cardholder's billing address information.

Verified by Visa, MasterCard Secure Code and Amex SafeKey (VbV/MCSC/SafeKey)
Authenticates the cardholder at the time of an online transaction.

Card Validation Digit (CVD)
Validates that cardholder is in possession of a genuine credit card during the transaction.

Note that all responses that are returned from these verification methods are intended to provide added security and fraud prevention. The response itself does not affect the completion of a transaction. Upon receiving a response, the choice to proceed with a transaction is left entirely to the merchant.

10.4.1 Implementation Options for TRMT

Option A

Process a Transaction Risk Management Tool query and obtain the response. You can then decide whether to continue with the transaction, abort the transaction, or use additional efraud features.

If you want to use additional efraud features, perform one or both of the following to help make your decision about whether to continue with the transaction or abort it:

- Process a VbV/MCSC/SafeKey transaction and obtain the response. The merchant then makes the decision whether to continue with the transaction or to abort it.
- Process a financial transaction including AVS/CVD details and obtain the response. The merchant then makes a decision whether to continue with the transaction or to abort it.

Option B

1. Process a Transaction Risk Management Tool query and obtain the response.
2. Process a VbV/MCSC/SafeKey transaction and obtain the response.
3. Process a financial transaction including AVS/CVD details and obtain the response.
4. Merchant then makes a one-time decision based on the responses received from the eFraud tools.

10.4.2 Implementation Checklist

The following checklists provide high-level tasks that are required as part of your implementation of the Transaction Risk Management Tool. Because each organization has certain project requirements for implementing system and process changes, this list is only a guideline, and does not cover all aspects of your project.

Download and review all of the applicable APIs and Integration Guides

Please review the sections outlined within this document that refers to the following feature

Table 45: API documentation

Document/API	Use the document if you are....
Transaction Risk Management Tool Integration Guide (Section #)	Implementing or updating your integration for the Transaction Risk Management Tool
Moneris MPI – Verified by Visa/MasterCard SecureCode/American Express SafeKey – Java API Integration Guide	Implementing or updating Verified by Visa, MasterCard SecureCode or American Express SafeKey
Basic transaction with VS and CVD (Section#)	Implementing or updating transaction processing, AVS or CVD

Design your transaction flow and business processes

When designing your transaction flow, think about which scenarios you would like to have automated, and which scenarios you would like to have handled manually by your employees.

The “Understand Transaction Risk Management Transaction Flow” and Handling Response Information (page 356) sections can help you work through the design of your transaction and process flows.

Things to consider when designing your process flows:

- Processes for notifying people within your organization when there is scheduled maintenance for Moneris Gateway.
- Handling refunds, canceled orders and so on.
- Communicating with customers when you will not be shipping the goods because of suspected fraud, back-ordered goods and so on.

Complete your development and testing

- The Moneris Gateway API - Integration Guide provides the technical details required for the development and testing. Ensure that you follow the testing instructions and data provided.

If you are an integrator

- Ensure that your solution meets the requirements for PCI-DSS/PA-DSS as applicable.
- Send an email to eproducts@moneris.com with the subject line “Certification Request”.
- Develop material to set up your customers as quickly as possible with your solution and a Moneris account. Include information such as:
    - Steps they must take to enter their store ID or API token information into your solution.

- Any optional services that you support via Moneris Gateway (such as TRMT, AVS, CVD, VBV/MCSC/SafeKey and so on) so that customers can request these features.

10.4.3 Making a Decision

Depending on your business policies and processes, the information obtained from the fraud tools (such as AVS, CVD, VbV/MCSC/SafeKey and TRMT) can help you make an informed decision about whether to accept a transaction or deny it because it is potentially fraudulent.

If you do not want to continue with a likely fraudulent transaction, you must inform the customer that you are not proceeding with their transaction.

If you are attempting to do further authentication by using the available fraud tools, but you have received an approval response instead, cancel the financial transaction by doing one of the following:

- If the original transaction is a Purchase, use a Purchase Correction or Refund transaction. You will need the original order ID and transaction number.
- If the original transaction is a Pre-Authorization, use a Completion transaction for $0.00.

11 Apple Pay In-App and on the Web Integration

11.1 About Apple Pay In-App and on the Web Integration

The Moneris Gateway enables merchants to process in-app or on the web payment methods in mobile applications and the Safari web browser on Apple devices via Apple Pay.

Moneris Solutions offers two processing and integration methods for Apple Pay. Merchants can choose to use one of two methods:

- Software Development Kit (SDK), or
- API

While both methods provide the same basic payment features, there are differences in their implementations.

This guide only deals with the API method; for detailed information about the SDK method of integration, see the Moneris Developer Portal at https://developer.moneris.com.

11.2 About API Integration of Apple Pay

An API integration works to provide a communication link between the merchants' server and Moneris' server. APIs are required to complete any transaction, and therefore the APIs for Apple Pay are also included within an SDK integration.

If the merchant chooses to use only an API integration, the merchant must decrypt payload information themselves before sending the decrypted information to the Moneris Gateway to be processed. Because this process is complicated, Apple recommend only businesses with expertise and a previously integrated payment processing system use APIs instead of SDKs.

11.2.1 Transaction Types That Use Apple Pay

In the Moneris Gateway API, there are two transaction types that allow you to process decrypted transaction payload information with Apple Pay:

- 11.4 Cavv Purchase – Apple Pay 11.4 Cavv Purchase – Apple Pay
- 11.5 Cavv Pre-Authorization – Apple Pay

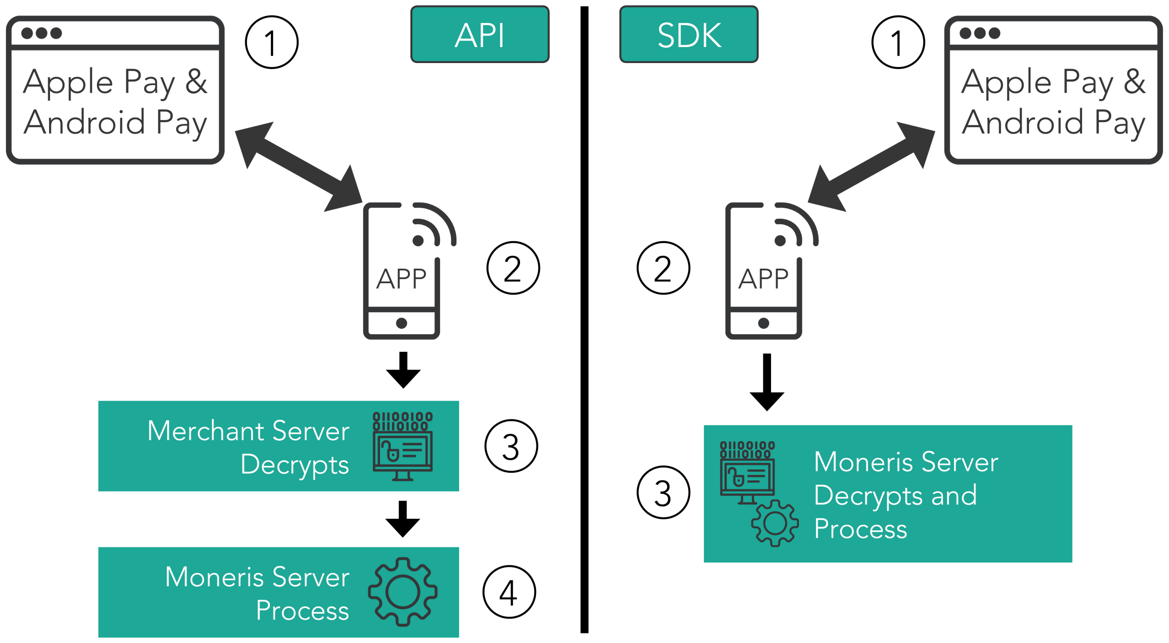
NOTE: INTERAC® e-Commerce functionality is currently available using the Cavv Purchase transaction type only.

Once you have processed the initial transaction using Cavv Purchase or Cavv Pre-Authorization, if required you can then process any of the following transactions:

- Refund (page 36)
- Pre-Authorization Completion (page 23)
- Purchase Correction (page 33)

11.3 Apple Pay In-App Process Flows

For both API and SDK methods of mobile in-app integration, the merchant's iOS app uses Apple's PassKit Framework to request and receive encrypted payment details from Apple. When payment details are returned in their encrypted form, they can be decrypted and processed by the Moneris Gateway in one of two ways: SDK or API.



Steps in the Apple Pay In-App and on the Web payment process

API

1. Merchant's mobile application or web page requests and receives the encrypted payload.
2. Encrypted payload is sent to the merchant's server, where it is decrypted.

3. Moneris Gateway receives the decrypted payload from the merchant's server, and processes the Cavv Purchase – Apple Pay (page 375)Cavv Purchase – Apple Pay (page 375) or Cavv Pre-Authorization – Apple Pay (page 379)Cavv Pre-Authorization – Apple Pay (page 379) transaction.
   a. Please ensure the wallet indicator is properly populated with the correct value (APP for Apple Pay In-App or APW for Apple Pay on the Web).

SDK

1. Merchant's mobile application or web page requests and receives the encrypted payload.
2. Encrypted payload is sent from the merchant's server to the Moneris Gateway, and the payload is decrypted and processed.

This guide only deals with the API method; for detailed information about the SDK method of integration, see the Moneris Developer Portal at https://developer.moneris.com.

11.4 Cavv Purchase – Apple Pay

The Cavv Purchase for Apple Pay transaction follows a 3-D Secure model but it does not require an MPI. Once the Apple Pay payload has been decrypted, this Purchase verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

For Apple Pay processing, this transaction is only applicable if choosing to integrate directly to the Apple Wallet (if not using the Moneris Apple Pay SDK). Please refer to 11 Apple Pay In-App and on the Web Integration for more details on your integration options.

Refer to Apple's Developer Portal for details on integrating directly to Apple Wallet to retrieve the payload data.

CavvPurchase transaction object definition

```
CavvPurchase cavvPurchase = new CavvPurchase();
```

HttpsPostRequest object for Cavv Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(cavvPurchase);
```

Cavv Purchase transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`cavvPurchase.setOrderId(order_id);`
amount	*String*	`cavvPurchase.setAmount(amount);`

Variable Name	Type and Limits	Set Method
	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	
credit card number	*String* max 20-character alpha-numeric	`cavvPurchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`cavvPurchase.setExpDate (expiry_date);`
Cardholder Authentication Verification Value (CAVV) **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	*String* 100-character alphanumeric	`cavvPurchase.setCavv(cavv);`
electronic commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	*String* 1-character alphanumeric	`cavvPurchase.setCryptType (crypt);`

Following fields are required for Apple Pay and Google Pay only:

Variable Name	Type and Limits	Set Method
network **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	*String* alphabetic	`cavvPurchase.SetNetwork (network);`
data type **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	*String* 3-character alphanumeric	`cavvPurchase.SetDataType (data_type);`

Cavv Purchase – Apple Pay transaction request fields – Optional

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck (status_check);`
customer ID	*String* 30-character alphanumeric	`cavvPurchase.setCustId(cust_ id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`cavvPurchase .setDynamicDescriptor (dynamic_descriptor);`
card match ID **NOTE:** Applies to Offlinx™ only; must be unique value	*String* 50-character alphanumeric	`cavvPurchase.setCmId (transaction_id);`

Variable Name	Type and Limits	Set Method
for each transaction		
Customer Information	*Object* N/A	`cavvPurchase.setCustInfo (customer);`

Sample Cavv Purchase for Apple Pay

```
package Canada;
import JavaAPI.*;
public class TestCanadaCavvPurchase
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
CavvPurchase cavvPurchase = new CavvPurchase();
cavvPurchase.setOrderId(order_id);
cavvPurchase.setCustId(cust_id);
cavvPurchase.setAmount(amount);
cavvPurchase.setPan(pan);
cavvPurchase.setExpdate(expdate);
cavvPurchase.setCavv(cavv);
cavvPurchase.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPurchase.setDynamicDescriptor(dynamic_descriptor);
//cavvPurchase.setWalletIndicator("APP"); //set only for wallet transactions. e.g APPLE PAY
//cavvPurchase.setNetwork("Interac"); //set only for Interac e-commerce
//cavvPurchase.setDataType("3DSecure"); //set only for Interac e-commerce
cavvPurchase.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
alphanumeric characters transaction id generated by merchant

//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cavvPurchase.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPurchase);
```

Sample Cavv Purchase for Apple Pay

```
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
System.out.println("IssuerId = " + receipt.getIssuerId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

11.5 Cavv Pre-Authorization – Apple Pay

The Cavv Pre-Authorization for Apple Pay transaction follows a 3-D Secure model but it does not require an MPI. Once the Apple Pay payload has been decrypted, this Pre-Authorization verifies funds on the customer's card, and holds the funds. To prepare the funds for deposit into the merchant's account please process a Pre-Authorization Completion transaction.

For Apple Pay processing, this transaction is only applicable if choosing to integrate directly to the Apple Wallet (if not using the Moneris Apple Pay SDK). Please refer to 11 Apple Pay In-App and on the Web Integration for more details on your integration options.

Refer to Apple's Developer Portal for details on integrating directly to Apple Wallet to retrieve the payload data.

NOTE: INTERAC® e-Commerce functionality is currently available using the Cavv Purchase transaction type only.

Cavv Pre-Authorization transaction object definition

```
CavvPreAuth cavvPreauth = new CavvPreAuth();
```

HttpsPostRequest object for Cavv Pre-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.setTransaction(cavvPreauth);
```

Cavv Pre-Authorization transaction values

Table 46: Cavv Pre-Authorization object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`cavvPreauth.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`cavvPreauth.setAmount(amount);`
Credit card number	String	20-character numeric	`cavvPreauth.setPan(pan);`

Value	Type	Limits	Set method
Cardholder Authentication Verification Value (CAVV) **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	String	50-character alphanumeric	`cavvPreauth.setCavv(cavv);`
Expiry date	String	4-character numeric	`cavvPreauth.setExpDate(expiry_date);`
E-commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	String	1-character alphanumeric	`cavvPreauth.setCryptType(crypt);`

Table 1 Cavv Pre-Authorization object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
Customer ID	String	50-character alphanumeric	`cavvPreauth.setCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-	`cavvPreauth`

Value	Type	Limits	Set method
		numeric	`.setDynamicDescriptor (dynamic_descriptor);`
Card Match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	String	50-character alpha-numeric	`cavvPreauth.setCmId (transaction_id);`
Network **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	String	alphabetical	`cavvPurchase.SetNetwork (network);`
Data Type **NOTE:** This request variable is mandatory for INTERAC® e-Commerce transactions conducted via Apple Pay, and is not for use with credit card transactions.	String	3-character alpha-numeric	`cavvPurchase.SetDataType (data_type);`

Sample Cavv Pre-Authorization for Apple Pay

```
package Canada;
import JavaAPI.*;
public class TestCanadaCavvPreauth
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1911"; //YYMM format
```

Sample Cavv Pre-Authorization for Apple Pay

```
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
CavvPreAuth cavvPreauth = new CavvPreAuth();
cavvPreauth.setOrderId(order_id);
cavvPreauth.setCustId(cust_id);
cavvPreauth.setAmount(amount);
cavvPreauth.setPan(pan);
cavvPreauth.setExpdate(expdate);
cavvPreauth.setCavv(cavv);
cavvPreauth.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPreauth.setDynamicDescriptor(dynamic_descriptor);
//cavvPreauth.setWalletIndicator("APP"); //set only for wallet transactions. e.g APPLE PAY
cavvPreauth.setCmId("8nAK8712sGaAkls56"); //set only for usage with Offlinx - Unique max 50
alphanumeric characters transaction id generated by merchant
//optional - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("U");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

cavvPreauth.setCofInfo(cof);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPreauth);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());
System.out.println("IssuerId = " + receipt.getIssuerId());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

12 Offlinx™

12.1 What Is a Pixel Tag?

A pixel tag is a piece of code that goes on a web page and requests an image file (a tiny transparent image or pixel) when loaded, which, while not visible to the user, allows Offlinx™ to gather relevant information about the user.

The data collected by our pixel tag is:

- Anonymous (not personally identifiable) and compliant with privacy standards
- Secure — utilizes SSL communication to transmit the data securely
- Not shared with anyone

12.2 Offlinx™ and API Transactions

The Offlinx™ Card Match pixel tag feature can be implemented via the Unified API with the Card Match ID variable, which corresponds to the Transaction ID in Offlinx™. The Card Match ID must be a unique value for each transaction.

For more information about the Offlinx™ solution, consult the Offlinx™ Pixel Tag Setup Guide available from your account/service manager.

API transactions where this applies:

- Purchase
- Pre-Authorization
- Purchase with 3-D Secure – cavvPurchase
- Pre-Authorization with 3-D Secure – cavvPreauth
- Cavv Purchase – Apple Pay
- Cavv Pre-Authorization – Apple Pay

13 Convenience Fee

13.1 About Convenience Fee

The Convenience Fee program was designed to allow merchants to offer the convenience of an alternative payment channel to the cardholder at a charge. This applies only when providing a true "convenience" in the form of an alternative payment channel outside the merchant's customary face-to-face payment channels. The convenience fee will be a separate charge on top of what the consumer is paying for the goods and/or services they were given, and this charge will appear as a separate line item on the consumer’s statement.

NOTE: The Convenience Fee program is only offered to certain supported Merchant Category Codes (MCCs). Please speak to your account manager for further details.

13.2 Purchase with Convenience Fee

NOTE: Convenience Fee Purchase with Customer Information is also supported.

Convenience Fee Purchase transaction object definition

```
Purchase purchase = new Purchase();
```

HttpsPostRequest object for Convenience Fee Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(purchase);
```

Convenience Fee Purchase transaction object values

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Table 1 Convenience Fee Purchase transaction object mandatory values

Value	Type	Limits	Set Method
Convenience Fee	Object	n/a	`ConvFeeInfo convFeeInfo = new ConvFeeInfo();` `purchase.setConvenienceFee (convFeeInfo);`
Order ID	String	50-character alpha-numeric	`purchase.setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`purchase.setAmount(amount);`
Credit card number	String	20-character numeric	`purchase.setPan(pan);`
Expiry date	String	4-character numeric YYMM format	`purchase.setExpDate(expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`purchase.setCryptType(crypt);`
Convenience fee amount	String	9-character decimal	`convFeeInfo.setConvenienceFee (convfee_amount);`

Table 2 Convenience Fee Purchase transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`purchase.setCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`purchase.setDynamicDescriptor (dynamic_descriptor);`
AVS information	Object		`AvsInfo avsCheck = new AvsInfo();` `purchase.setAvsInfo`

Table 2 Convenience Fee Purchase transaction object optional values (continued)

Value	Type	Limits	Set Method
			`(avsCheck);`
CVD information	Object		`CvdInfo cvdCheck = new CvdInfo();` `purchase.setCvdInfo (cvdCheck);`

Sample Purchase with Convenience Fee

```
package Canada;
import JavaAPI.*;
public class TestCanadaConvFeePurchase
{
public static void main(String args[])
{
String store_id = "monca00392";
String api_token = "qYdISUhHiOdfTr1CLNpN";
String processing_country_code = "CA";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "10.00";
String pan = "4242424242424242";
String expdate = "1911";
String crypt = "7";

ConvFeeInfo convFeeInfo = new ConvFeeInfo();
convFeeInfo.setConvenienceFee("1.00");

Purchase purchase = new Purchase();
purchase.setOrderId(order_id);
purchase.setAmount(amount);
purchase.setPan(pan);
purchase.setExpdate(expdate);
purchase.setCryptType(crypt);
purchase.setConvFeeInfo(convFeeInfo);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(purchase);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
```

Sample Purchase with Convenience Fee

```
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CfSuccess = " + receipt.getCfSuccess());
System.out.println("CfStatus = " + receipt.getCfStatus());
System.out.println("FeeAmount = " + receipt.getFeeAmount());
System.out.println("FeeRate = " + receipt.getFeeRate());
System.out.println("FeeType = " + receipt.getFeeType());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

13.3 Convenience Fee Purchase w/ Customer Information

Convenience Fee Purchase with Customer information transaction object definition

`Purchase purchase = new Purchase();`

HttpsPostRequest object for Convenience Fee Purchase with Customer Info transaction

`HttpsPostRequest mpgReq = new HttpsPostRequest();`

`mpgReq.setTransaction(purchase);`

Convenience Fee Purchase w/ Customer Information transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method

Convenience Fee Inform-ation	*Object*
	N/A
		`ConvFeeInfo convFeeInfo = new ConvFeeInfo();`
		`purchase.setConvenienceFee (convFeeInfo);`

order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`purchase.setOrderId(order_id);`
amount	*String*	`purchase.setAmount(amount);`

Variable Name	Type and Limits	Set Method
	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	
credit card number	*String* max 20-character alpha-numeric	`purchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`purchase.setExpDate(expiry_date);`
electronic commerce indicator	*String* 1-character alphanumeric	`purchase.setCryptType(crypt);`
convenience fee amount	*String* 9-character decimal	`purchase convFeeInfo.setConvenienceFee (convfee_amount);`

Convenience Fee Purchase w/ Customer Information transaction request fields – Optional

Variable Name	Type and Limits	Set Method
customer ID	*String* 30-character alphanumeric	`purchase.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`purchase .setDynamicDescriptor (dynamic_descriptor);`
Customer Information	*Object* N/A	`purchase.setCustInfo (customer);`

Variable Name	Type and Limits	Set Method
AVS Information	*Object* N/A	`purchase.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`purchase.setCvdInfo (cvdCheck);`

Sample Convenience Fee Purchase with Customer Information

```
package Canada;

import java.util.*;
import JavaAPI.*;
public class TestCanadaConvFeePurchaseCustInfo
{
public static void main(String[] args)
{
String store_id = "monca00392";
String api_token = "qYdISUhHiOdfTr1CLNpN";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "10.00";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM format
String crypt = "7";
String processing_country_code = "CA";
boolean status_check = false;
/********************* Billing/Shipping Variables ****************************/
String first_name = "Bob";
String last_name = "Smith";
String company_name = "ProLine Inc.";
String address = "623 Bears Ave";
String city = "Chicago";
String province = "Illinois";
String postal_code = "M1M2M1";
String country = "Canada";
String phone = "777-999-7777";
String fax = "777-999-7778";
String tax1 = "10.00";
String tax2 = "5.78";
String tax3 = "4.56";
String shipping_cost = "10.00";
/********************* Order Line Item Variables *****************************/
```

Sample Convenience Fee Purchase with Customer Information

```
String[] item_description = new String[] { "Chicago Bears Helmet", "Soldier Field Poster" };
String[] item_quantity = new String[] { "1", "1" };
String[] item_product_code = new String[] { "CB3450", "SF998S" };
String[] item_extended_amount = new String[] { "150.00", "19.79" };
/********* ************************************* *******************************/
/* */
/* Customer Information Option 1 */
/* */
/********** ********************************** ******************************/
/********************** Customer Information Object *************************/
CustInfo customer = new CustInfo();
/********************** Set Customer Billing Information *********************/
customer.setBilling(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/******************** Set Customer Shipping Information **********************/
customer.setShipping(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/***************************** Order Line Items ******************************/
customer.setItem(item_description[0], item_quantity[0],
item_product_code[0], item_extended_amount[0]);
customer.setItem(item_description[1], item_quantity[1],
item_product_code[1], item_extended_amount[1]);
/********* ************************************** *****************************/
/* */
/* Customer Information Option 2 */
/* */
/********* *************************************** ****************************/
/********************** Customer Information Object *************************/
CustInfo customer2 = new CustInfo();
/******************************* Billing Hashtable *************************/
Hashtable<String, String> b = new Hashtable<String, String>(); //billing hashtable
b.put("first_name", first_name);
b.put("last_name", last_name);
b.put("company_name", company_name);
b.put("address", address);
b.put("city", city);
b.put("province", province);
b.put("postal_code", postal_code);
b.put("country", country);
b.put("phone", phone);
b.put("fax", fax);
b.put("tax1", tax1); //federal tax
b.put("tax2", tax2); //prov tax
b.put("tax3", tax3); //luxury tax
b.put("shipping_cost", shipping_cost); //shipping cost
customer2.setBilling(b);
/****************************** Shipping Hashtable *************************/
Hashtable<String, String> s = new Hashtable<String, String>(); //shipping hashtable
s.put("first_name", first_name);
s.put("last_name", last_name);
s.put("company_name", company_name);
s.put("address", address);
s.put("city", city);
s.put("province", province);
s.put("postal_code", postal_code);
s.put("country", country);
s.put("phone", phone);
s.put("fax", fax);
```

Sample Convenience Fee Purchase with Customer Information

```
s.put("tax1", tax1); //federal tax
s.put("tax2", tax2); //prov tax
s.put("tax3", tax3); //luxury tax
s.put("shipping_cost", shipping_cost); //shipping cost
customer2.setShipping(s);
/************************* Order Line Item1 Hashtable ************************/
Hashtable<String, String> i1 = new Hashtable<String, String>(); //item hashtable #1
i1.put("name", item_description[0]);
i1.put("quantity", item_quantity[0]);
i1.put("product_code", item_product_code[0]);
i1.put("extended_amount", item_extended_amount[0]);
customer2.setItem(i1);
/************************* Order Line Item2 Hashtable **************************/
Hashtable<String, String> i2 = new Hashtable<String, String>(); //item hashtable #2
i2.put("name", "item2's name");
i2.put("quantity", "7");
i2.put("product_code", "item2's product code");
i2.put("extended_amount", "5.01");
customer2.setItem(i2);
/*************** Miscellaneous Customer Information Methods *******************/
customer.setEmail("nick@widget.com");
customer.setInstructions("Make it fast!");

/*************** Convenience Fee *******************/
ConvFeeInfo convFeeInfo = new ConvFeeInfo();
convFeeInfo.setConvenienceFee("1.00");
/********************** Transactional Request Object **************************/
Purchase purchase = new Purchase();
purchase.setOrderId(order_id);
purchase.setAmount(amount);
purchase.setPan(pan);
purchase.setExpdate(expdate);
purchase.setCryptType(crypt);
purchase.setCustInfo(customer);
purchase.setConvFeeInfo(convFeeInfo);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(purchase);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
```

Sample Convenience Fee Purchase with Customer Information

```
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());

    System.out.println("CfSuccess = " + receipt.getCfSuccess());
    System.out.println("CfStatus = " + receipt.getCfStatus());
    System.out.println("FeeAmount = " + receipt.getFeeAmount());
    System.out.println("FeeRate = " + receipt.getFeeRate());
    System.out.println("FeeType = " + receipt.getFeeType());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

13.4 Convenience Fee Purchase with VbV, MCSC and Amex SafeKey

Convenience Fee Purchase with VbV/MCSC/SafeKey transaction object definition

```
CavvPurchase cavvPurchase = new CavvPurchase();
```

HttpsPostRequest object for Convenience Fee Purchase w/ VbV/MCSC/SafeKey transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

Convenience Fee Purchase with VbV/MCSC/SafeKey transaction request fields – Required

For a full description of mandatory and optional values, see Appendix A Definitions of Request Fields

Variable Name	Type and Limits	Set Method
Convenience Fee Information	*Object* N/A	`cavvPurchase .setConvenienceFee (convFeeInfo);`
order ID	*String* 50-character alphanumeric a-Z A-Z 0-9 _ - : . @ spaces	`cavvPurchase.setOrderId (order_id);`
amount	*String* 10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point	`cavvPurchase.setAmount (amount);`

Variable Name	Type and Limits	Set Method
	EXAMPLE: 1234567.89	
credit card number	*String* max 20-character alpha-numeric	`cavvPurchase.setPan(pan);`
expiry date	*String* 4-character alphanumeric YYMM	`cavvPurchase.setExpDate(expiry_date);`
electronic commerce indicator	*String* 1-character alphanumeric	`cavvPurchase.setCryptType(crypt);`
Cardholder Authentication Verification Value (CAVV)	*String* 50-character alphanumeric	`cavvPurchase.setCavv(cavv);`
convenience fee amount	*String* 9-character decimal	`convFeeInfo.setConvenienceFee(convfee_amount);`

Convenience Fee Purchase with VbV/MCSC/SafeKey transaction request fields – Optional

Variable Name	Type and Limits	Set Method
status check	*Boolean* true/false	`mpgReq.setStatusCheck(status_check);`
customer ID	*String* 30-character alphanumeric	`cavvPurchase.setCustId(cust_id);`
dynamic descriptor	*String* max 20-character alpha-numeric total of 22 characters including your merchant name and separator	`cavvPurchase.setDynamicDescriptor(dynamic_descriptor);`
electronic commerce indic-	*String*	`cavvPurchase.setCryptType`

Variable Name	Type and Limits	Set Method
ator	1-character alphanumeric	`(crypt);`
Customer Information	*Object* N/A	`cavvPurchase.setCustInfo (customer);`
AVS Information	*Object* N/A	`cavvPurchase.setAvsInfo (avsCheck);`
CVD Information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	*Object* N/A	`cavvPurchase.setCvdInfo (cvdCheck);`

Sample Convenience Fee Purchase with VbV/MCSC/SafeKey

```
package Canada;
import JavaAPI.*;
public class TestCanadaConvFeeCavvPurchase
{
public static void main(String[] args)
{
String store_id = "monca00392";
String api_token = "qYdISUhHiOdfTr1CLNpN";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String cust_id = "CUS887H67";
String amount = "10.42";
String pan = "4242424242424242";
String expdate = "1901"; //YYMM
String cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
String dynamic_descriptor = "123456";
String processing_country_code = "CA";
String crypt_type = "5";
boolean status_check = false;
/*************** Convenience Fee *******************/
ConvFeeInfo convFeeInfo = new ConvFeeInfo();
convFeeInfo.setConvenienceFee("1.00");

CavvPurchase cavvPurchase = new CavvPurchase();
cavvPurchase.setOrderId(order_id);
```

Sample Convenience Fee Purchase with VbV/MCSC/SafeKey

```
cavvPurchase.setCustId(cust_id);
cavvPurchase.setAmount(amount);
cavvPurchase.setPan(pan);
cavvPurchase.setExpdate(expdate);
cavvPurchase.setCavv(cavv);
cavvPurchase.setCryptType(crypt_type); //Mandatory for AMEX only
cavvPurchase.setDynamicDescriptor(dynamic_descriptor);
cavvPurchase.setConvFeeInfo(convFeeInfo);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(cavvPurchase);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("CavvResultCode = " + receipt.getCavvResultCode());

System.out.println("CfSuccess = " + receipt.getCfSuccess());
System.out.println("CfStatus = " + receipt.getCfStatus());
System.out.println("FeeAmount = " + receipt.getFeeAmount());
System.out.println("FeeRate = " + receipt.getFeeRate());
System.out.println("FeeType = " + receipt.getFeeType());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

14 Recurring Billing

14.1 About Recurring Billing

Recurring Billing allows you to set up payments whereby Moneris automatically processes the transactions and bills customers on your behalf based on the billing cycle information you provide.

Recurring Billing series are created by sending the Recurring Billing object in these transactions:

- Purchase
- Purchase with Vault
- Purchase with 3-D Secure (cavvPurchase)

You can modify a Recurring Billing series after it has been created by sending the Recurring Billing Update administrative transaction.

> **NOTE:** Alternatively, if you prefer to manage recurring series on your own merchant system, you can send the periodic payments as basic Purchase transactions with the e-commerce indicator (`crypt_type`) value = 2 and with the Credential on File info object included.

14.2 Purchase with Recurring Billing

Recurring Billing Info Object Definition

```
Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_
recurs, period, recur_amount);
```

Transaction object set method

```
<transaction>.setRecur(recurring_cycle);
```

Recurring Billing Info Object Request Fields

Variable Name	Type and Limits	Description
number of recurs `num_recurs`	*String* numeric 1-99	The number of times that the transaction must recur

Variable Name	Type and Limits	Description
period `period`	*String* numeric 1-999	Number of recur unit intervals that must pass between recurring billings
start date `start_date`	*String* YYMMDD format	Date of the first future recurring billing transaction; this must be a date in the future If an additional charge will be made immediately, the start now variable must be set to true
start now `start_now`	*String* true/false	Set to true if a charge will be made against the card immediately; otherwise set to false When set to false, use Card Verification prior to sending the Purchase with Recurring Billing and Credential on File objects **NOTE:** Amount to be billed immediately can differ from the subsequent recurring amounts
recurring amount `recur_amount`	*String* 10-character decimal, minimum three digits Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	Dollar amount of the recurring transaction This amount will be billed on the start date, and then billed repeatedly based on the interval defined by period and recur unit
recur unit `recur_unit`	*String* day, week, month or eom	Unit to be used as a basis for the interval Works in conjunction with the period variable to define the billing frequency

Sample Purchase with Recurring Billing

```
package Canada;

import java.util.*;
import JavaAPI.*;
public class TestCanadaPurchaseRecur
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "10.00";
String pan = "4242424242424242";
String expiry_date = "1901"; //YYMM format
String crypt = "7";
/************************* Recur Variables **********************************/
String recur_unit = "month"; //eom = end of month
String start_now = "true";
String start_date = "2018/04/01";
String num_recurs = "12";
String period = "1";
String recur_amount = "30.00";
String processing_country_code = "CA";
boolean status_check = false;
/************************* Recur Object Option1 ******************************/
Recur recurring_cycle = new Recur(recur_unit, start_now, start_date,
num_recurs, period, recur_amount);
/************************* Recur Object Option2 ******************************/
Hashtable<String, String> recur_hash = new Hashtable<String, String>();
recur_hash.put("recur_unit", recur_unit);
recur_hash.put("start_now", start_now);
recur_hash.put("start_date", start_date);
recur_hash.put("num_recurs", num_recurs);
recur_hash.put("period", period);
recur_hash.put("recur_amount", recur_amount);
/************************ Transactional Object *******************************/
Purchase purchase = new Purchase();
purchase.setOrderId(order_id);
purchase.setAmount(amount);
purchase.setPan(pan);
purchase.setExpdate(expiry_date);
purchase.setCryptType(crypt);
/******************************* Set Recur ***********************************/
purchase.setRecur(recurring_cycle);
//Mandatory on Recurs - Credential on File details
CofInfo cof = new CofInfo();
cof.setPaymentIndicator("R");
cof.setPaymentInformation("2");
cof.setIssuerId("139X3130ASCXAS9");

purchase.setCofInfo(cof);

/**************************** Https Post Request ***************************/
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
```

Sample Purchase with Recurring Billing

```
    mpgReq.setTransaction(purchase);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    /******************************* Receipt ***********************************/
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("Recur Success = " + receipt.getRecurSuccess());
    System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
    System.out.println("IssuerId = " + receipt.getIssuerId());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

14.3 Recurring Billing Update

After you have set up a Recurring Billing transaction series, you can change some of the details of the series as long as it has not yet completed the preset recurring duration (i.e., it hasn't terminated yet).

Before sending a Recurring Billing Update transaction that updates the credit card number, you must send a Card Verification request. This requirement does not apply if you are only updating the schedule or amount.

Things to Consider:

- When completing the update recurring billing portion please keep in mind that the recur bill dates cannot be changed to have an end date greater than 10 years from today and cannot be changed to have an end date end today or earlier.

Recurring Billing Update transaction object definition

```
RecurUpdate recurUpdate = new RecurUpdate();
```

HttpsPostRequest object for Recurring Billing Update transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(recurUpdate);
```

Recurring Billing Update transaction values

Table 1 Recurring Billing Update – Basic Required Fields

Variable Name	Type and Limits	Set Method
Order ID `order_id`	*String* 50-character alphanumeric	`recurUpdate.setOrderId (order_id);`

Table 2 Recurring Billing Update – Basic Optional Fields

Variable Name	Type and Limits	Set Method
Customer ID `cust_id`	*String* 50-character alphanumeric	`recurUpdate.setCustId(cust_ id);`
Credit card number `pan`	*String* 20-character alphanumeric	`recurUpdate.setPan(pan);`
Expiry date `expiry_date`	*String* YYMM	`recurUpdate.setExpdate (expiry_date);`

Table 3 Recurring Billing Update – Recurring Billing Required Fields

Variable Name	Type and Limits	Set Method	Description
Recurring amount `recur_ amount`	*String* 10-character decimal; Up to 7 digits (dollars) + decimal point + 2 digits (cents) after the decimal point **EXAMPL-**	`recurUpdate.setRecurAmount (recur_amount);`	Changes the amount that is billed recurrently The change takes effect on the next charge

Variable Name	Type and Limits	Set Method	Description
	E: 1234567-.89		
Add number of recurs `add_num`	*String* numeric, 1-999	`recurUpdate.setAddNumRecurs (add_num);`	**Adds** to the given number of recurring transactions to the current (remaining) number This can be used if a customer decides to extend a membership or subscription Cannot be used to decrease the current number of recurring transactions; use Change number of recurs instead
Change number of recurs `total_num`	*String* numeric, 1-999	`recurUpdate.setTotalNumRecurs (total_num);`	**Replaces** the current (remaining) number of recurring transactions
Hold recurring billing `hold`	*String* true/false	`recurUpdate.setHold(hold);`	Temporarily pauses recurring billing While a transaction is on hold, it is not billed for the recurring amount; however, the number of remaining recurs continues to be decremented during that time
Terminate recurring transaction `terminate`	*String* true/false	`recurUpdate.setTerminate (terminate);`	Terminates recurring billing **NOTE:** After it has been terminated, a recurring time

Variable Name	Type and Limits	Set Method	Description
			transaction **cannot** be reactivated; a new purchase transaction with recurring billing must be submitted.

Sample Recurring Billing Update

```
package Canada;
import JavaAPI.*;
public class TestCanadaRecurUpdate
{
public static void main(String[] args)
{
String store_id = "store5";
String api_token = "yesguy";
String order_id = "Test155409282";
String cust_id = "antonio";
String recur_amount = "1.50";
String pan = "4242424242424242";
String expiry_date = "1902";
//String add_num = "";
//String total_num = "";
//String hold = "";
//String terminate = "";
String processing_country_code = "CA";
boolean status_check = false;
//Credential on File details
CofInfo cof = new CofInfo();
cof.setIssuerId("139X3130ASCXAS9");

RecurUpdate recurUpdate = new RecurUpdate();
recurUpdate.setOrderId(order_id);
recurUpdate.setCustId(cust_id);
recurUpdate.setRecurAmount(recur_amount);
recurUpdate.setPan(pan);
recurUpdate.setExpdate(expiry_date);
//recurUpdate.setAddNumRecurs(add_num);
//recurUpdate.setTotalNumRecurs(total_num);
//recurUpdate.setHold(hold);
//recurUpdate.setTerminate(terminate);
recurUpdate.setCofInfo(cof);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(recurUpdate);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
```

Sample Recurring Billing Update

```
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("RecurUpdateSuccess = " + receipt.getRecurUpdateSuccess());
    System.out.println("NextRecurDate = " + receipt.getNextRecurDate());
    System.out.println("RecurEndDate = " + receipt.getRecurEndDate());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

14.4 Recurring Billing Response Fields and Codes

Table 47 outlines the response fields that are part of recurring billing. Some are available when you set up recurring billing (such as with a Purchase transaction), and some are available when you update an existing transaction with the Recurring Billing transaction.

Receipt object definition

`Receipt receipt = mpgReq.getReceipt();`

Table 47: Recurring Billing response fields

Value	Type	Limits	Get method
	Description		
Transaction object with Recurring Billing response fields			
Response code	String	3-character numeric	`receipt.getResponseCode();`
	See Table 48: for a description of possible response codes.		
Recur success	String	TBD	`receipt.getRecurSuccess();`
	Indicates whether the transaction successfully registered		
Recur update object response fields			
Recur update success	String	`true`/`false`	`receipt.getRecurUpdateSuccess();`
	Indicates whether the transaction successfully updated.		
Next recur date	String	yyyy-mm-dd format	`receipt.getNextRecurDate();`
	Indicates when the transaction will be billed again.		

Table 47: Recurring Billing response fields

Value	Type	Limits	Get method
	Description		
Recur end date	String	yyyy-mm-dd format	`receipt.getRecurEndDate();`
	Indicates when the Recurring Billing Transaction will end.		

The Recur Update response is a 3-digit numeric value. The following is a list of all possible responses after a Recur Update transaction has been sent.

Table 48: Recur update response codes

Request Value	Definition
001	Recurring transaction successfully updated (optional: terminated)
983	Cannot find the previous transaction
984	Data error: (optional: field name)
985	Invalid number of recurs
986	Incomplete: timed out
null	Error: Malformed XML

14.5 Credential on File and Recurring Billing

NOTE: The value of the **payment indicator** field must be **R** when sending Recurring Billing transactions.

For Recurring Billing transactions which are set to start **immediately**:

1. Send a Purchase transaction request with both the Recurring Billing and Credential on File info objects (with Recurring Billing object field **start now** = true)

For Recurring Billing transactions which are set to start on a **future** date:

1. Send Card Verification transaction request including the Credential on File info object to get the Issuer ID
2. Send Purchase transaction request with the Recur and Credential on File info objects included

For updating a Recurring Billing series where you are updating the card number (does not apply if you are only modifying the schedule or amount in a recurring series):

1. Send Card Verification request including the Credential on File info object to get the Issuer ID
2. Send a Recurring Billing Update transaction

For more information about the Recurring Billing object, see Definition of Request Fields – Recurring.

15 Customer Information

The Customer Information object offers a number of fields to be submitted as part of the financial transaction, and stored by Moneris. These details may be viewed in the future in the Merchant Resource Center.

The following transactions support the Customer Information object :

- Purchase (Basic, Interac Debit and Vault)
- Pre-Authorization (Basic and Vault)
- Re-Authorization (Basic)

The Customer Information object holds three types of information:

- Billing/Shipping information
- Miscellaneous customer information properties
- Item information

Things to Consider:

- If you send characters that are not included in the allowed list, these extra transaction details may not be stored.
- All fields are alphanumeric and allow the following characters: a-z A-Z 0-9 _ - : . @ $ = /
- All French accents should be encoded as HTML entities, such as &eacute.
- The data sent in Billing and Shipping Address fields will not be used for any address verification.

15.1 Using the Customer Information Object

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate a CustInfo object.

Any transaction that supports CustInfo has a setCustInfo method. This is used to write the customer information to the transaction object before writing the transaction object to the connection object.

CustInfo object definition

```
CustInfo customer = new CustInfo();
```

Transaction object set method

```
<transaction>.setCustInfo(customer);
```

15.1.1 Customer Info Object – Miscellaneous Properties

While most of the Customer Information data is organized into objects, there are some values that are properties of the CustInfo object itself. They are explained in the table below.

Table 49: CustInfo object miscellaneous properties

Value	Type	Limits	Set method
Email Address	String	60-character alphanumeric	`customer.setEmail(email);`
Instructions	String	100-character alphanumeric	`customer.setInstructions(note);`

15.1.2 Customer Info Object – Billing/Shipping Information

Billing and shipping information is stored as part of the Customer Information object. They can be written to the object in one of two ways:

- Using set methods
- Using hash tables

Whichever method you use, you will be writing the information found in the table below for both the billing information and the shipping information.

All values are alphanumeric strings. Their maximum lengths are given in the Limit column.

Table 50: Billing and shipping information values

Value	Limit	Hash table key
First name	30	"first_name"
Last name	30	"last_name"
Company name	50	"company_name"
Address	70	"address"
City	30	"city"

Table 50: Billing and shipping information values (continued)

Value	Limit	Hash table key
Province/State	30	"province"
Postal/Zip code	30	"postal_code"
Country	30	"country"
Phone number (voice)	30	"phone"
Fax number	30	"fax"
Federal tax	10	"tax1"
Provincial/State tax	10	"tax2"
County/Local/Specialty tax	10	"tax3"
Shipping cost	10	"shipping_cost"

15.1.2.1 Set Methods for Billing and Shipping Info

The billing information and the shipping information for a given CustInfo object are written by using the `customer.setBilling()` and `customer.setShipping()` methods respectively:

```
customer.setBilling(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2, tax3, shipping_cost);
```

```
customer.setShipping(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2, tax3, shipping_cost);
```

Both of these methods have the same set of mandatory arguments. They are described in the Billing and shipping information values table in 15.1.2.1 Set Methods for Billing and Shipping Info.

For sample code, see 15.2 Customer Information Sample Code.

15.1.2.2 Using Hash Tables for Billing and Shipping Info

Writing billing or shipping information using hash tables is done as follows:

1. Instantiate a CustInfo object.
2. Instantiate a hash table object. (The sample code uses a different hash table for billing and shipping for clarity purposes. However, the skillful developer can re-use the same one.)
3. Build the hash table using put methods with the hash table keys found in the Billing and shipping information values table in 15.1.2 Customer Info Object – Billing/Shipping Information.
4. Call the CustInfo object's setBilling/setShipping method to pass the hash table information to the CustInfo object

5. Call the transaction object's setCustInfo method to write the CustInfo object (with the billing/-shipping information to the transaction object.

For sample code, see 15.2 Customer Information Sample Code.

15.1.3 Customer Info Object – Item Information

The Customer Information object can hold information about multiple items. For each item, the values in the table below can be written.

All values are strings, but note the guidelines in the Limits column.

Table 51: Item information values

Value	Limits	Hash table key
Item name	45-character alphanumeric	"name"
Item quantity	5-character numeric	"quantity"
Item product code	20-character alphanumeric	"product_code"
Item extended amount	9-character decimal with at least 3 digits and 2 penny values. 0.01-999999.99	"extended_amount"

One way of representing multiple items is with four arrays. This is the method used in the sample code. However, there are two ways to write the item information to the CustInfo object:

- Set methods
- Hash tables

15.1.3.1 Set Methods for Item Information

All the item information found in the Item information values table in 15.1.3 Customer Info Object – Item Information is written to the CustInfo object in one instruction for a given item. Such as:

```
customer.setItem(item_description, item_quantity, item_product_code, item_
extended_amount);
```

For sample code (showing how to use arrays to write information about two items), see 15.2 Customer Information Sample Code.

15.1.3.2 Using Hash Tables for Item Information

Writing item information using hash tables is done as follows:

1. Instantiate a CustInfo object.

2. Instantiate a hash table object. (The sample code uses a different hash table for each item for clarity purposes. However, the skillful developer can re-use the same one.)
3. Build the hash table using put methods with the hash table keys in the Item information values table in 15.1.3 Customer Info Object – Item Information.
4. Call the CustInfo object's setItem method to pass the hash table information to the CustInfo object
5. Call the transaction object's setCustInfo method to write the CustInfo object (with the item information to the transaction object.

For sample code that shows how to use arrays to write information about two items, see 15.2 Customer Information Sample Code.

15.2 Customer Information Sample Code

Below are two examples of a Basic Purchase Transaction with Customer Information. Both samples start with the same declaration of variables, as shown.

Values that are not involved in the Customer Information feature are not shown.

Note that the two items ordered are represented by four arrays, and the billing and shipping details are the same.

Declaring the variables (common to both methods)

```
/********************* Billing/Shipping Variables ****************************/
String first_name = "Bob";
String last_name = "Smith";
String company_name = "ProLine Inc.";
String address = "623 Bears Ave";
String city = "Chicago";
String province = "Illinois";
String postal_code = "M1M2M1";
String country = "Canada";
String phone = "777-999-7777";
String fax = "777-999-7778";
String tax1 = "10.00";
String tax2 = "5.78";
String tax3 = "4.56";
String shipping_cost = "10.00";

/********************* Order Line Item Variables *****************************/
String[] item_description = new String[] { "Chicago Bears Helmet", "Soldier Field Poster" };
String[] item_quantity = new String[] { "1", "1" };
String[] item_product_code = new String[] { "CB3450", "SF998S" };
String[] item_extended_amount = new String[] { "150.00", "19.79" };
/*****************************************************************************/
```

Sample Purchase with Customer Information – Set method version

```
CustInfo customer = new CustInfo();

/*************** Miscellaneous Customer Information Methods *******************/
customer.setEmail("nick@widget.com");
customer.setInstructions("Make it fast!");
```

Sample Purchase with Customer Information – Set method version

```
/********************* Set Customer Billing Information *********************/
customer.setBilling(first_name, last_name, company_name, address, city, province, postal_code,
country, phone, fax, tax1, tax2, tax3, shipping_cost);

/******************* Set Customer Shipping Information **********************/
customer.setShipping(first_name, last_name, company_name, address, city, province, postal_code,
country, phone, fax, tax1, tax2, tax3, shipping_cost);

/*************************** Order Line Items ****************************/
customer.setItem(item_description[0], item_quantity[0], item_product_code[0], item_extended_amount
[0]);
customer.setItem(item_description[1], item_quantity[1], item_product_code[1], item_extended_amount
[1]);

Purchase purchase = new Purchase();
purchase.setCustInfo(customer);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(purchase);
mpgReq.send();
```

Sample Purchase with Customer Information – Hash table version

```
CustInfo customer2 = new CustInfo();
/*************** Miscellaneous Customer Information Methods *******************/
customer.setEmail("nick@widget.com");
customer.setInstructions("Make it fast!");
/****************************** Billing Hashtable **************************/
Hashtable<String, String> b = new Hashtable<String, String>(); //billing hashtable
b.put("first_name", first_name);
b.put("last_name", last_name);
b.put("company_name", company_name);
b.put("address", address);
b.put("city", city);
b.put("province", province);
b.put("postal_code", postal_code);
b.put("country", country);
b.put("phone", phone);
b.put("fax", fax);
b.put("tax1", tax1); //federal tax
b.put("tax2", tax2); //prov tax
b.put("tax3", tax3); //luxury tax
b.put("shipping_cost", shipping_cost); //shipping cost
customer2.setBilling(b);
/***************************** Shipping Hashtable **************************/
Hashtable<String, String> s = new Hashtable<String, String>(); //shipping hashtable
s.put("first_name", first_name);
s.put("last_name", last_name);
s.put("company_name", company_name);
s.put("address", address);
s.put("city", city);
s.put("province", province);
s.put("postal_code", postal_code);
s.put("country", country);
```

Sample Purchase with Customer Information – Hash table version

```
    s.put("phone", phone);
    s.put("fax", fax);
    s.put("tax1", tax1); //federal tax
    s.put("tax2", tax2); //prov tax
    s.put("tax3", tax3); //luxury tax
    s.put("shipping_cost", shipping_cost); //shipping cost
    customer2.setShipping(s);
    /************************* Order Line Item1 Hashtable ***********************/
    Hashtable<String, String> i1 = new Hashtable<String, String>(); //item hashtable #1
    i1.put("name", item_description[0]);
    i1.put("quantity", item_quantity[0]);
    i1.put("product_code", item_product_code[0]);
    i1.put("extended_amount", item_extended_amount[0]);
    customer2.setItem(i1);
    /************************* Order Line Item2 Hashtable *************************/
    Hashtable<String, String> i2 = new Hashtable<String, String>(); //item hashtable #2
    i2.put("name", "item2's name");
    i2.put("quantity", "7");
    i2.put("product_code", "item2's product code");
    i2.put("extended_amount", "5.01");
    customer2.setItem(i2);

    Purchase purchase = new Purchase();
    purchase.setCustInfo(customer);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setTransaction(purchase);
    mpgReq.send();
```

16 Status Check

16.1 About Status Check

Status Check is a connection object value that allows merchants to verify whether a previously sent transaction was processed successfully.

To submit a Status Check request, resend the original transaction with all the same parameter values, but set the status check value to either `true` or `false`.

Once set to “true”, the gateway will check the status of a transaction that has an order_id that matches the one passed.

- If the transaction is found, the gateway will respond with the specifics of that transaction.
- If the transaction is not found, the gateway will respond with a not found message.

Once it is set to “false”, the transaction will process as a new transaction.

For example, if you send a Purchase transaction with Status Check, include the same values as the original Purchase such as the order ID and the amount.

The feature must be enabled in your merchant profile. To have it enabled, contact Moneris.

Things to Consider:

- The Status Check request should only be used once and immediately (within 2 minutes) after the last transaction that had failed.
- The Status Check request should not be used to check openTotals & batchClose requests.
- Do not resend the Status Check request if it has timed out. Additional investigation is required.

16.2 Using Status Check Response Fields

After you have used the connection object to send a Status Check request, you can use the Receipt object to obtain the information you want regarding the success of the original transaction.

The status response fields related to the status check are Status Code and Status Message.

Possible Status Code response values:

- 0-49: successful transaction
- 50-999: unsuccessful transaction.

Possible Status Message response values:

- Found: Status code is 0-49
- Not found or Null: Status code is 50-999)

If the Status Message is `Found`, all other response fields are the same as those from the original transaction.

If the Status Message is `Not found`, all other response fields will be Null.

16.3 Sample Purchase with Status Check

Sample Purchase transaction with Status Check

```
package Canada;
import JavaAPI.*;
public class TestCanadaPurchase
{
    public static void main(String[] args)
    {
            Purchase purchase = new Purchase();
            purchase.setOrderId("order");
            purchase.setAmount("1.00");
            purchase.setPan("4242424242424242");
            purchase.setExpdate("2202");
            purchase.setCryptType("1");

            HttpsPostRequest mpgReq = new HttpsPostRequest();
            mpgReq.setProcCountryCode("CA");
            mpgReq.setTestMode(true); //false or comment out this line for production transactions
            mpgReq.setStoreId("store1");
            mpgReq.setApiToken("yesguy");
            mpgReq.setTransaction(purchase);
            boolean status_check = true;
            mpgReq.setStatusCheck(status_check);
            mpgReq.send();

            try
            {
                Receipt receipt = mpgReq.getReceipt();
                System.out.println("StatusCode = " + receipt.getStatusCode());
                System.out.println("StatusMessage = " + receipt.getStatusMessage());
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
    }
}
```

17 Visa Checkout

17.1 About Visa Checkout

Visa Checkout is a digital wallet service offered to customers using credit cards. Visa Checkout functionality can be integrated into the Moneris Gateway via the API.

17.2 Transaction Types - Visa Checkout

Below is a list of transactions supported by the Visa Checkout API, other terms used for the transaction type are indicated in brackets.

VdotMePurchase (sale)
Call to Moneris to obtain funds on the Visa Checkout `callId` and ready them for deposit into the merchant's account. It also updates the customer's Visa Checkout transaction history.

VdotMePreAuth (authorisation / pre-authorization)
Call to Moneris to verify funds on the Visa Checkout `callid` and reserve those funds for your merchant account. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from this call so that they may be settled in the merchant's account, a `VdotMeCompletion` must be performed. It also updates the customer's Visa Checkout transaction history.

VdotMeCompletion (Completion / Capture)
Call to Moneris to obtain funds reserved by `VdotMePreAuth` call. This transaction call retrieves the locked funds and readies them for settlement into the merchant's account. This call must be made typically within 72 hours of performing `VdotMePreAuth`. It also updates the customer's Visa Checkout transaction history.

VdotMePurchaseCorrection (Void / Purchase Correction)
Call to Moneris to void the VdotMePurchases and VdotMeCompletions the same day* that they occurred on. It also updates the customer's Visa Checkout transaction history.

VdotMeRefund (Credit)
Call to Moneris to refund against a `VdotMePurchase` or `VdotMeCompletion` to refund any part, or all of the transaction. It also updates the customer's Visa Checkout transaction history.

VdotMeInfo (Credit)
Call to Moneris to obtain cardholder details such as, name on card, partial card number, expiry date, shipping and billing information.

17.3 Integrating Visa Checkout Lightbox

1. Using the API Key you obtained when you configured your Visa Checkout store, create Visa Checkout Lightbox integration with JavaScript by following the Visa documentation, which is available on Visa Developer portal:

   Visa Checkout General Information (JavaScript SDK download)
   https://developer.visa.com/products/visa_checkout

   Getting Started With Visa checkout
   https://developer.visa.com/products/visa_checkout/guides#getting_started

   Adding Visa Checkout to Your Web Page
   https://developer.visa.com/products/visa_checkout/guides#adding_to_page

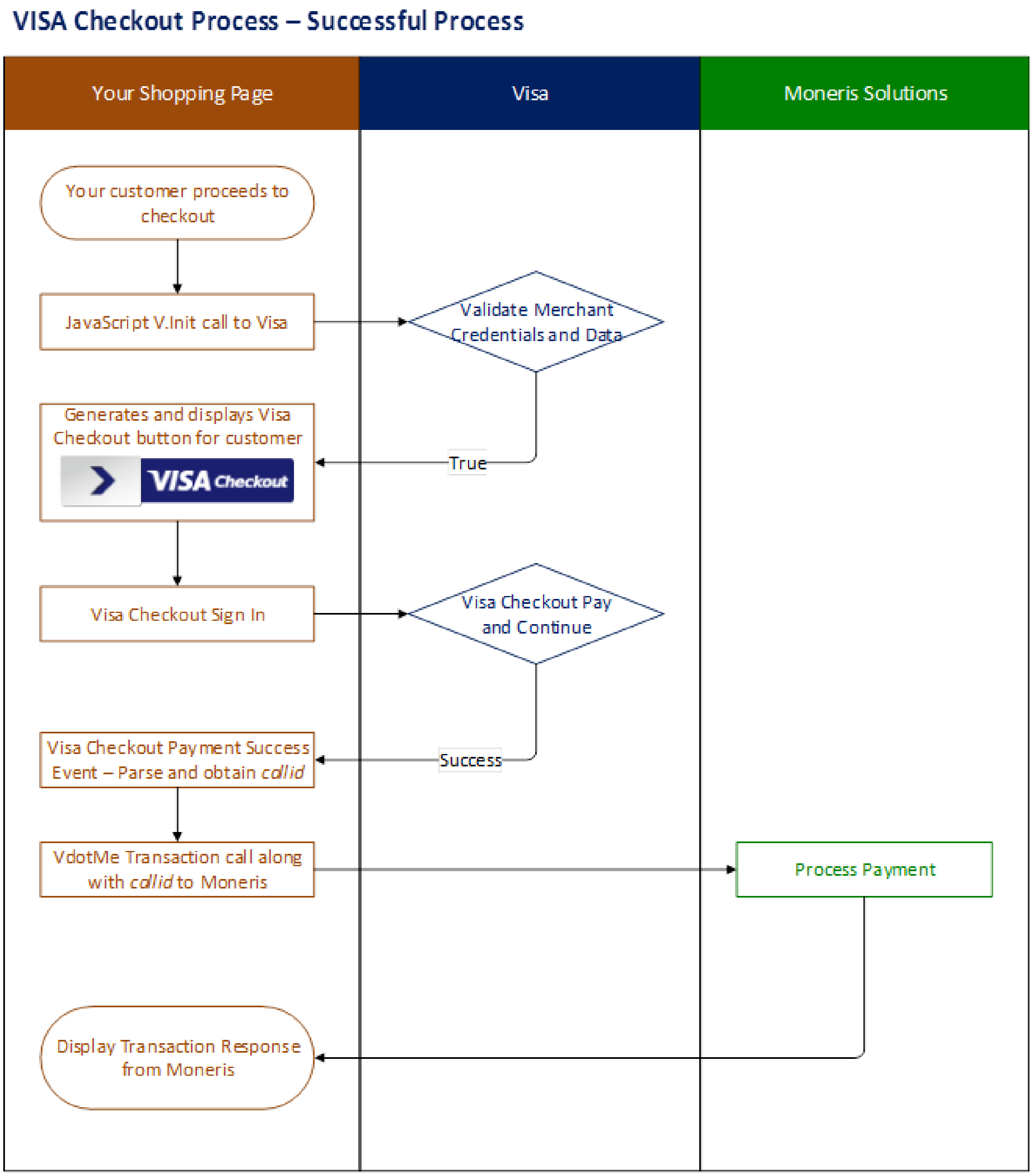
   Submitting the Consumer Payment Request
   https://developer.visa.com/products/visa_checkout/guides#submitting_csr

2. If you get a payment success event from the resulting Visa Lightbox JavaScript, you will have to parse and obtain the `callid` from their JSON response. The additional information is obtained using `VdotMeInfo`.

Once you have obtained the callid from Visa Lightbox, you can make appropriate Visa Checkout `VdotMe` transaction call to Moneris to process your transaction and obtain your funds.

> **NOTE:** During Visa Checkout testing in our QA test environment, please use the API key that you generated in the Visa Checkout configuration for the `V.Init` call in your JavaScript.

17.4 Transaction Flow for Visa Checkout


VISA Checkout Process – Successful Process
Your Shopping Page
Visa
Moneris Solutions
Your customer proceeds to checkout
JavaScript V.Init call to Visa
Validate Merchant Credentials and Data
True
Generates and displays Visa Checkout button for customer
VISA Checkout
Visa Checkout Sign In
Visa Checkout Pay and Continue
Success
Visa Checkout Payment Success Event – Parse and obtain *callid*
VdotMe Transaction call along with *callid* to Moneris
Process Payment
Display Transaction Response from Moneris

17.5 Visa Checkout Purchase

VdotMePurchase transaction object definition

```
VdotMePurchase vmepurchase = new VdotMePurchase();
```

HttpsPostRequest for VdotMePurchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMePurchase transaction object values

Table 1 VdotMePurchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vmepurchase.setOrderId(order_id);`
Call ID	String	20-character numeric	`vmepurchase.setCallId(call_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vmepurchase.setAmount(amount);`
E-commerce indicator	String	1-character alpha-numeric	`vmepurchase.setCryptType(crypt);`

Table 2 VdotMePurchase transaction object optional values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alpha-numeric	`vmepurchase.setDynamicDescriptor(dynamic_descriptor);`
Status check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`

Sample VdotMePurchase

```
    package Canada;
```

Sample VdotMePurchase

```
import JavaAPI.*;
public class TestCanadaVdotMePurchase
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
String cust_id = "Joe Doe";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String amount = "8.00";
String crypt_type = "7";
String call_id = "9104624497663077101";
String dynamic_descriptor = "inv123";
String processing_country_code = "CA";
boolean status_check = false;
VdotMePurchase vmepurchase = new VdotMePurchase();
vmepurchase.setOrderId(order_id);
vmepurchase.setCustId(cust_id);
vmepurchase.setAmount(amount);
vmepurchase.setCallId(call_id);
vmepurchase.setCryptType(crypt_type);
vmepurchase.setDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vmepurchase);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("StatusCode = " + receipt.getStatusCode());
System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

17.6 Visa Checkout Pre-Authorization

`VdotMePreAuth` is virtually identical to the `VdotMePurchase` with the exception of the transaction type name.

If the order could not be completed for some reason, such as an order is cancelled, made in error or not fulfillable, the `VdotMePreAuth` transaction must be reversed within 72 hours.

To reverse an authorization, perform a `VdotMeCompletion` transaction for $0.00 (zero dollars).

VdotMePreAuth transaction object definition

```
VdotMePreauth vMePreauthRequest = new VdotMePreauth();
```

HttpsPostRequest object for VdotMePreAuth transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMePreAuth transaction object values

Table 1 VdotMePreAuth transaction object mandatory values

Value	Type	Limits	Set Method
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vDotMeReauthRequest.setAmount(amount);`
Call ID	String	20-character numeric	`vDotMeReauthRequest.setCallId(call_id);`
Order ID	String	50-character alpha-numeric	`vDotMeReauthRequest.setOrderId(order_id);`
E-commerce indicator	String	1-character alpha-numeric	`vDotMeReauthRequest.setCryptType(crypt);`

Table 2 VdotMePreAuth transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vMePreauthRequest.setCustId (cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vDotMeReauthRequest .setDynamicDescriptor (dynamic_descriptor);`

Sample VdotMePreAuth

```
package Canada;
import JavaAPI.*;
public class TestCanadaVdotMePreauth
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
String amount = "5.00";
String crypt_type = "7";
java.util.Date createDate = new java.util.Date();
String order_id = "Test"+createDate.getTime();
String call_id = "9104624497663077101";
String cust_id = "my customer id";
String processing_country_code = "CA";
boolean status_check = false;
VdotMePreauth vMePreauthRequest = new VdotMePreauth();
vMePreauthRequest.setOrderId(order_id);
vMePreauthRequest.setAmount(amount);
vMePreauthRequest.setCallId(call_id);
vMePreauthRequest.setCustId(cust_id);
vMePreauthRequest.setCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vMePreauthRequest);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
```

Sample VdotMePreAuth

```
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("StatusCode = " + receipt.getStatusCode());
System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

17.7 Visa Checkout Completion

The `VdotMeCompletion` transaction is used to secure the funds locked by a `VdotMePreAuth` transaction.

You may also perform this transaction at $0.00 (zero dollars) to reverse a `VdotMePreauth` transaction that you are unable to fulfill.

VdotMeCompletion transaction object definition

```
VdotMeCompletion vmecompletion = new VdotMeCompletion();
```

HttpsPostRequest object for VdotMeCompletion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMeCompletion transaction object values

Table 1 VdotMeCompletion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vmecompletion.setOrderId (order_id);`
Transaction number	String	255-character alpha-numeric	`vmecompletion.setTxnNumber (txn_number);`
Completion amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point	`vmecompletion.setCompAmount (comp_amount);`

Value	Type	Limits	Set Method
		EXAMPLE: 1234567.89	
E-commerce indicator	String	1-character alpha-numeric	`vmecompletion.setCryptType (crypt);`

Table 2 VdotMeCompletion transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vmecompletion.setCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vmecompletion .setDynamicDescriptor (dynamic_descriptor);`

Sample VdotMeCompletion

```
package Canada;
import JavaAPI.*;
public class TestCanadaVdotMeCompletion
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
String order_id = "Test1432134710264";
String txn_number = "724379-0_10";
String comp_amount = "1.00";
String ship_indicator = "P";
String crypt_type = "7";
String cust_id = "mycustomerid";
String dynamic_descriptor = "inv 123";
String processing_country_code = "CA";
boolean status_check = false;
VdotMeCompletion vmecompletion = new VdotMeCompletion();
vmecompletion.setOrderId(order_id);
vmecompletion.setTxnNumber(txn_number);
vmecompletion.setAmount(comp_amount);
vmecompletion.setCryptType(crypt_type);
vmecompletion.setDynamicDescriptor(dynamic_descriptor);
vmecompletion.setCustId(cust_id);
vmecompletion.setShipIndicator(ship_indicator);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vmecompletion);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
```

Sample VdotMeCompletion

```
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("StatusCode = " + receipt.getStatusCode());
System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

17.8 Visa Checkout Purchase Correction

`VdotMePurchaseCorrection` is used to cancel a `VdotMeCompletion` or `VdotMePurchase` transaction that was performed in the current batch. No other transaction types can be corrected using this method.

No amount is required because it is always for 100% of the original transaction.

VdotMePurchaseCorrection transaction object definition

```
VdotMePurchaseCorrection vDotMePurchaseCorrection = new
VdotMePurchaseCorrection();
```

HttpsPostRequest object for VdotMePurchaseCorrection transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMePurchaseCorrection transaction object values

Table 1 VdotMePurchaseCorrection transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vDotMePurchaseCorrection .setOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`vDotMePurchaseCorrection .setTxnNumber(txn_number);`

Table 2 VdotMePurchaseCorrection transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vDotMePurchaseCorrection .setCustId(cust_id);`
Status check	Boolean	true/false	`mpgReq.setStatusCheck(status_ check);`

Sample VdotMePurchaseCorrection

```
package Canada;
import JavaAPI.*;
public class TestCanadaVdotMePurchaseCorrection
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
String order_id = "Test1432134533159";
String txn_number = "724377-0_10";
String crypt_type = "7";
String cust_id = "my customer id";
String processing_country_code = "CA";
boolean status_check = false;
VdotMePurchaseCorrection vDotMePurchaseCorrection = new VdotMePurchaseCorrection();
vDotMePurchaseCorrection.setOrderId(order_id);
vDotMePurchaseCorrection.setCustId(cust_id);
vDotMePurchaseCorrection.setTxnNumber(txn_number);
vDotMePurchaseCorrection.setCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vDotMePurchaseCorrection);
mpgReq.setStatusCheck(status_check);
mpgReq.send();
try
{
Receipt receipt = mpgReq.getReceipt();
System.out.println("CardType = " + receipt.getCardType());
```

Sample VdotMePurchaseCorrection

```
System.out.println("TransAmount = " + receipt.getTransAmount());
System.out.println("TxnNumber = " + receipt.getTxnNumber());
System.out.println("ReceiptId = " + receipt.getReceiptId());
System.out.println("TransType = " + receipt.getTransType());
System.out.println("ReferenceNum = " + receipt.getReferenceNum());
System.out.println("ResponseCode = " + receipt.getResponseCode());
System.out.println("ISO = " + receipt.getISO());
System.out.println("BankTotals = " + receipt.getBankTotals());
System.out.println("Message = " + receipt.getMessage());
System.out.println("AuthCode = " + receipt.getAuthCode());
System.out.println("Complete = " + receipt.getComplete());
System.out.println("TransDate = " + receipt.getTransDate());
System.out.println("TransTime = " + receipt.getTransTime());
System.out.println("Ticket = " + receipt.getTicket());
System.out.println("TimedOut = " + receipt.getTimedOut());
System.out.println("StatusCode = " + receipt.getStatusCode());
System.out.println("StatusMessage = " + receipt.getStatusMessage());
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

17.9 Visa Checkout Refund

`VdotMeRefund` will credit a specified amount to the cardholder's credit card and update their Visa Checkout transaction history. A refund can be sent up to the full value of the original `VdotMeCompletion` or `VdotMePurchase`.

VdotMeRefund transaction object definition

```
VdotMeRefund vDotMeRefundRequest = new VdotMeRefund();
```

HttpsPostRequest object for VdotMeRefund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMeRefund transaction object values

Table 1 VdotMeRefund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vDotMeRefundRequest .setOrderId(order_id);`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`vDotMeRefundRequest.setAmount (amount);`
Transaction number	String	255-character alpha-numeric	`vDotMeRefundRequest .setTxnNumber(txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`vDotMeRefundRequest .setCryptType(crypt);`

Table 2 VdotMeRefund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vDotMeRefundRequest.setCustId (cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vDotMeRefundRequest .setDynamicDescriptor (dynamic_descriptor);`
Status check	Boolean	true/false	`mpgReq.setStatusCheck(status_ check);`

Sample VdotMeRefund

```
package Canada;
import JavaAPI.*;
public class TestCanadaVdotMeRefund
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
```

Sample VdotMeRefund

```
    String order_id = "Test1432134710264";
    String txn_number = "724380-1_10";
    String amount = "1.00";
    String crypt_type = "7";
    String dynamic_descriptor = "inv 123";
    String cust_id = "my customer id";
    String processing_country_code = "CA";
    boolean status_check = false;
    VdotMeRefund vDotMeRefundRequest = new VdotMeRefund();
    vDotMeRefundRequest.setOrderId(order_id);
    vDotMeRefundRequest.setAmount(amount);
    vDotMeRefundRequest.setCustId(cust_id);
    vDotMeRefundRequest.setTxnNumber(txn_number);
    vDotMeRefundRequest.setCryptType(crypt_type);
    vDotMeRefundRequest.setDynamicDescriptor(dynamic_descriptor);
    HttpsPostRequest mpgReq = new HttpsPostRequest();
    mpgReq.setProcCountryCode(processing_country_code);
    mpgReq.setTestMode(true); //false or comment out this line for production transactions
    mpgReq.setStoreId(store_id);
    mpgReq.setApiToken(api_token);
    mpgReq.setTransaction(vDotMeRefundRequest);
    mpgReq.setStatusCheck(status_check);
    mpgReq.send();
    try
    {
    Receipt receipt = mpgReq.getReceipt();
    System.out.println("CardType = " + receipt.getCardType());
    System.out.println("TransAmount = " + receipt.getTransAmount());
    System.out.println("TxnNumber = " + receipt.getTxnNumber());
    System.out.println("ReceiptId = " + receipt.getReceiptId());
    System.out.println("TransType = " + receipt.getTransType());
    System.out.println("ReferenceNum = " + receipt.getReferenceNum());
    System.out.println("ResponseCode = " + receipt.getResponseCode());
    System.out.println("ISO = " + receipt.getISO());
    System.out.println("BankTotals = " + receipt.getBankTotals());
    System.out.println("Message = " + receipt.getMessage());
    System.out.println("AuthCode = " + receipt.getAuthCode());
    System.out.println("Complete = " + receipt.getComplete());
    System.out.println("TransDate = " + receipt.getTransDate());
    System.out.println("TransTime = " + receipt.getTransTime());
    System.out.println("Ticket = " + receipt.getTicket());
    System.out.println("TimedOut = " + receipt.getTimedOut());
    System.out.println("StatusCode = " + receipt.getStatusCode());
    System.out.println("StatusMessage = " + receipt.getStatusMessage());
    }
    catch (Exception e)
    {
    e.printStackTrace();
    }
    }
    }
```

17.10 Visa Checkout Information

`VdotMeInfo` will get customer information from their Visa Checkout wallet. The details returned are dependent on what the customer has stored in Visa Checkout.

VdotMeInfo transaction object definition

```
VdotMeInfo vmeinfo = new VdotMeInfo();
```

HttpsPostRequest object for VdotMeInfo transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

VdotMeInfo transaction object values

Table 1 VdotMeInfo transaction object mandatory values

Value	Type	Limits	Set Method
Call ID	String	20-character numeric	`vmeinfo.setCallId(call_id);`

Sample VdotMeInfo

```
package Canada;
import java.util.Hashtable;
import java.util.Set;
import JavaAPI.*;
public class TestCanadaVdotMeInfo
{
public static void main(String[] args)
{
String store_id = "store2";
String api_token = "yesguy";
String call_id = "8620484083629792701";
String processing_country_code = "CA";
boolean status_check = false;
VdotMeInfo vmeinfo = new VdotMeInfo();
vmeinfo.setCallId(call_id);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true); //false or comment out this line for production transactions
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(vmeinfo);
mpgReq.setStatusCheck(status_check);
mpgReq.send();


try
{
Receipt receipt = mpgReq.getReceipt();

System.out.println("dump of vmeDataHash variables:");
Hashtable<String, String>vmeDataHash = new Hashtable<String, String>();
vmeDataHash = receipt.getVmeDataHash();

Set<String> keys = vmeDataHash.keySet();
for(String key: keys){
System.out.println("Value of "+key+" is: "+vmeDataHash.get(key));
}
System.out.println("Response Code: " + receipt.getResponseCode());
System.out.println("Response Message: " + receipt.getMessage());
System.out.println("Currency Code: " + receipt.getCurrencyCode());
System.out.println("Payment Totals: " + receipt.getPaymentTotal());
```

Sample VdotMeInfo

```
System.out.println("User First Name: " + receipt.getUserFirstName());
System.out.println("User Last Name: " + receipt.getUserLastName());
System.out.println("Username: " + receipt.getUserName());
System.out.println("User Email: " + receipt.getUserEmail());
System.out.println("Encrypted User ID: " + receipt.getEncUserId());
System.out.println("Creation Time Stamp: " + receipt.getCreationTimeStamp());
System.out.println("Name on Card: " + receipt.getNameOnCard());
System.out.println("Expiration Month: " + receipt.getExpirationDateMonth());
System.out.println("Expiration Year: " + receipt.getExpirationDateYear());
System.out.println("Last 4 Digits: " + receipt.getLastFourDigits());
System.out.println("Bin Number (6 Digits): " + receipt.getBinSixDigits());
System.out.println("Card Brand: " + receipt.getCardBrand());
System.out.println("Card Type: " + receipt.getVdotMeCardType());
System.out.println("Billing Person Name: " + receipt.getPersonName());
System.out.println("Billing Address Line 1: " + receipt.getBillingAddressLine1());
System.out.println("Billing City: " + receipt.getBillingCity());
System.out.println("Billing State/Province Code: " + receipt.getBillingStateProvinceCode());
System.out.println("Billing Postal Code: " + receipt.getBillingPostalCode());
System.out.println("Billing Country Code: " + receipt.getBillingCountryCode());
System.out.println("Billing Phone: " + receipt.getBillingPhone());
System.out.println("Billing ID: " + receipt.getBillingId());
System.out.println("Billing Verification Status: " + receipt.getBillingVerificationStatus());
System.out.println("Partial Shipping Country Code: " + receipt.getPartialShippingCountryCode());
System.out.println("Partial Shipping Postal Code: " + receipt.getPartialShippingPostalCode());
System.out.println("Shipping Person Name: " + receipt.getShippingPersonName());
System.out.println("Shipping Address Line 1: " + receipt.getShipAddressLine1());
System.out.println("Shipping City: " + receipt.getShippingCity());
System.out.println("Shipping State/Province Code: " + receipt.getShippingStateProvinceCode());
System.out.println("Shipping Postal Code: " + receipt.getShippingPostalCode());
System.out.println("Shipping Country Code: " + receipt.getShippingCountryCode());
System.out.println("Shipping Phone: " + receipt.getShippingPhone());
System.out.println("Shipping Default: " + receipt.getShippingDefault());
System.out.println("Shipping ID: " + receipt.getShippingId());
System.out.println("Shipping Verification Status: " + receipt.getShippingVerificationStatus());
System.out.println("isExpired: " + receipt.getIsExpired());
System.out.println("Base Image File Name: " + receipt.getBaseImageFileName());
System.out.println("Height: " + receipt.getHeight());
System.out.println("Width: " + receipt.getWidth());
System.out.println("Issuer Bid: " + receipt.getIssuerBid());
System.out.println("Risk Advice: " + receipt.getRiskAdvice());
System.out.println("Risk Score: " + receipt.getRiskScore());
System.out.println("AVS Response Code: " + receipt.getAvsResponseCode());
System.out.println("CVV Response Code: " + receipt.getCvvResponseCode());
System.out.println("\r\nPress the enter key to exit");
}
catch (Exception e)
{
e.printStackTrace();
}
}
}
```

18 Testing a Solution

18.1 About the Merchant Resource Center

The Merchant Resource Center is the user interface for Moneris Gateway services. There is also a QA version of the Merchant Resource Center site specifically allocated for you and other developers to use to test your API integrations with the gateway.

You can access the Merchant Resource Center in the test environment at:

https://esqa.moneris.com/mpg (Canada)

The test environment is generally available 24/7, but 100% availability is not guaranteed. Also, please be aware that other merchants are using the test environment in the Merchant Resource Center. Therefore, you may see transactions and user IDs that you did not create. As a courtesy to others who are testing, we ask that you use only the transactions/users that you created. This applies to processing Refund transactions, changing passwords or trying other functions.

18.2 Logging In to the QA Merchant Resource Center

To log in to the QA Merchant Resource Center for testing purposes:

1. Go to the Merchant Resource Center QA website at https://esqa.moneris.com/mpg
2. Enter your username and password, which are the same email address and password you use to log in to the Developer Portal
3. Enter your Store ID, which you obtained from the Developer Portal's My Testing Credentials as described in Test Credentials for Merchant Resource Center (page 433)

18.3 Test Credentials for Merchant Resource Center

For testing purposes, you can either use the pre-existing test stores in the Merchant Resource Center, or you can create your own unique test store where you will only see your own transactions. If you want to use the pre-existing stores, use the test credentials provided in the following tables with the corresponding lines of code, as in the examples below.

Example of Corresponding Code For Canada:

```
String processing_country_code = "CA";

mpgReq.setTestMode(true);

String store_id = "store5";

String api_token = "yesguy";
```

Table 52: Test Server Credentials - Canada

store_id	api_token	Username	Password	Other Information
store1	yesguy	demouser	password	
store2	yesguy	demouser	password	
store3	yesguy	demouser	password	
store4	yesguy	demouser	password	
store5	yesguy	demouser	password	
monca00392	yesguy	demouser	password	Use this store to test Convenience Fee transactions
moncaqagt1	mgtokenguy1	demouser	password	Use this store to test Token Sharing
moncaqagt2	mgtokenguy2	demouser	password	Use this store to test Token Sharing
moncaqagt3	mgtokenguy3	demouser	password	Use this store to test Token Sharing
monca01428	mcmpguy	demouser	password	Use this store to test MasterCard MasterPass

Alternatively, you can create and use a unique test store where you will only see your own transactions. For more on this, see Getting a Unique Test Store ID and API Token (page 435)

18.4 Getting a Unique Test Store ID and API Token

Transactions requests via the API will require you to have a Store ID and a corresponding API token.For testing purposes, you can either use the pre-existing test stores in the Merchant Resource Center, or you can create your own unique test store where you will only see your own transactions.

To get your unique Store ID and API token:

1. Log in to the Developer Portal at https://developer.moneris.com
2. In the My Profile dialog, click the Full Profile FULL PROFILE button
3. Under My Testing Credentials, select Request Testing Credentials
4. Enter your Developer Portal password and select your country
5. Record the Store ID and API token that are given, as you will need them for logging in to the Merchant Resource Center (Store ID) and for API requests (API token).

Alternatively, you can use the pre-existing test stores already set up in the Merchant Resource Center as described in Test Credentials for Merchant Resource Center (page 433).

18.5 Processing a Transaction

18.5.1 Overview

There are some common steps for every transaction that is processed.

1. Instantiate the transaction object (e.g., Purchase), and update it with object definitions that refer to the individual transaction.
2. Instantiate the HttpsPostRequest connection object and update it with connection information, host information and the transaction object that you created in step 18.5

   Section 18.5 (page 437) provides the HttpsPostRequest connection object definition. This object and its variables apply to **every** transaction request.
3. Invoke the HttpsPostRequest object's `send()` method.
4. Instantiate the Receipt object, by invoking the HttpsPostRequest object's get Receipt method. Use this object to retrieve the applicable response details.

Some transactions may require steps in addition to the ones listed here. Below is a sample Purchase transaction with each major step outlined. For extensive code samples of other transaction types, refer to the Java API ZIP file.

NOTE: For illustrative purposes, the order in which lines of code appear below may differ slightly from the same sample code presented elsewhere in this document.

```
    import java.io.*;
    import java.util.*;
    import java.net.*;
    import JavaAPI.*;
```

Include all necessary classes.

```
    String order_id = "Test"+createDate.getTime();
    String amount = "5.00";
    String pan = "4242424242424242";
    String expdate = "1901"; //YYMM format
    String crypt = "7";
    String processing_country_code = "CA";
```

Define all mandatory values for the transaction object properties.

```
    String store_id = "store5";
    String api_token = "yesguy";
```

Define all mandatory values for the connection object properties.

```
  Purchase purchase = new Purchase();

  purchase.setOrderId(order_id);
  purchase.setAmount(amount);
  purchase.setPan(pan);
  purchase.setExpdate(expdate);
  purchase.setCryptType(crypt);
  purchase.setDynamicDescriptor("2134565");
```

Instantiate the transaction object and assign values to properties.

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setProcCountryCode(processing_country_code);
mpgReq.setTestMode(true);
mpgReq.setStoreId(store_id);
mpgReq.setApiToken(api_token);
mpgReq.setTransaction(purchase);
mpgReq.setStatusCheck(status_check);
```

Instantiate connection object and assign values to properties, including the transaction object you just created.

```
try
{
        Receipt receipt = mpgReq.getReceipt();

        System.out.println("CardType = " + receipt.getCardType());
        System.out.println("TransAmount = " + receipt.getTransAmount());
        System.out.println("TxnNumber = " + receipt.getTxnNumber());
        System.out.println("ReceiptId = " + receipt.getReceiptId());
        System.out.println("TransType = " + receipt.getTransType());
        System.out.println("ReferenceNum = " + receipt.getReferenceNum());
        System.out.println("ResponseCode = " + receipt.getResponseCode());
        System.out.println("ISO = " + receipt.getISO());
        System.out.println("BankTotals = " + receipt.getBankTotals());
        System.out.println("Message = " + receipt.getMessage());
        System.out.println("AuthCode = " + receipt.getAuthCode());
        System.out.println("Complete = " + receipt.getComplete());
        System.out.println("TransDate = " + receipt.getTransDate());
        System.out.println("TransTime = " + receipt.getTransTime());
        System.out.println("Ticket = " + receipt.getTicket());
        System.out.println("TimedOut = " + receipt.getTimedOut());
        System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
}
        catch (Exception e)
        {
            e.printStackTrace();
        }
}
}
```

Instantiate the Receipt object and use its get methods to retrieve the desired response data.

18.5.2 HttpsPostRequest Object

The transaction object that you instantiate becomes a property of this object when you call its set transaction method.

HttpsPostRequest Object Definition

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

After instantiating the HttpsPostRequest object, update its mandatory and optional values as outlined in the following values tables.

Table 53: HttpsPostRequest object mandatory values

Value	Type	Limits	Set method
	Description		
Processing country code	String	2-character alphabetic	`mpgReq.setProcCountryCode (processing_country_code);`
	`CA` for Canada, `US` for USA.		
Test mode	Boolean	true/false	`mpgReq.setTestMode(true);`
	Set to `true` when in test mode. Set to `false` (or comment out entire line) when in production mode.		
Store ID	String	10-character alphanumeric	`mpgReq.setStoreId(store_id);`
	Unique identifier provided by Moneris upon merchant account set up. See 18.1 About the Merchant Resource Center for test environment details.		
API Token	String	20-character alphanumeric	`mpgReq.setApiToken(api_token);`
	Unique alphanumeric string assigned upon merchant account activation. To locate your production API token, refer to the Merchant Resource Center Admin Store Settings. See 18.3 Test Credentials for Merchant Resource Center for test environment details.		
Transaction	Object	Not applicable	`mpgReq.setTransaction (transaction);`
	This argument is one of the numerous transaction types discussed in the rest of this manual. (Such as Purchase, Refund and so on.) This object is instantiated in step 1 above.		

Table 1 HttpsPostRequest object optional values

Value	Type	Limits	Set method
	Description		
Status Check	Boolean	true/false	`mpgReq.setStatusCheck(status_check);`
	See Appendix A Definitions of Request Fields. **NOTE:** while this value belongs to the HttpsPostRequest object, it is only supported by some transactions. Check the individual transaction definition to find out whether Status Check can be used.		

18.5.3 Receipt Object

After you send a transaction using the HttpsPostRequest object's send method, you can instantiate a receipt object.

Receipt Object Definition

```
Receipt receipt = mpgReq.getReceipt();
```

For an in-depth explanation of Receipt object methods and properties, see Appendix B Definitions of Response Fields.

18.6 Testing INTERAC® Online Payment Solutions

Acxsys has two websites where merchants can post transactions for testing the fund guarantee porting of INTERAC® Online Payment transactions. The test `IDEBIT_MERCHNUM` value is provided by Moneris after registering in the test environment.

After registering, the following two links become accessible:

- Merchant Test Tool
- Certification Test Tool

Merchant Test Tool

https://merchant-test.interacidebit.ca/gateway/merchant_test_processor.do

This URL is used to simulate the transaction response process, to validate response variables, and to properly integrate your checkout process.

When testing INTERAC® Online Payment transactions, you are forwarded to the INTERAC® Online Payment Merchant Testing Tool. A screen appears where certain fields need to be completed.

For an approved response, do not alter any of the fields except for the ones listed here.

IDEBIT_TRACK2
To form a track2 when testing with the Moneris Gateway, use one of these three numbers:

3728024906540591206=01121122334455000

5268051119993326=01121122334455000000

453781122255=011211223344550000000000

IDEBIT_ISSNAME
RBC

IDEBIT_ISSCONF
123456

For a declined response, provide any other value as the IDEBIT_TRACK2. Click **Post to Merchant**.

Whether the transaction is approved or declined, do **not** click **Validate Data**. This will return validation errors.

Certification Test Tool

https://merchant-test.interacidebit.ca/gateway/merchant_certification_processor.do

This URL is used to complete the required INTERAC® Online Payment Merchant Front-End Certification test cases, which are outlined in Appendix E (page 512) and Appendix F (page 516).

To confirm the fund that was guaranteed above, an INTERAC® Online Payment Purchase must be sent to the Moneris Gateway QAusing the following test store information:

Host: esqa.moneris.com

Store ID: store3

API Token: yesguy

You can always log into the Merchant Resource Center to check the results using the following information:

URL: https://esqa.moneris.com/mpg

Store ID: store3

Note that all response variables that are posted back from the IOP gateway in step 5.4 of 5.4 must be validated for length of field, permitted characters and invalid characters.

18.7 Testing MPI Solutions

When testing your implementation of the Moneris MPI, you can use the Visa/MasterCard/Amex PIT (production integration testing) environment. The testing process is slightly different than a production environment in that when the inline window is generated, it does not contain any input boxes. Instead, it contains a window of data and a **Submit** button. Clicking **Submit** loads the response in the testing window. The response will not be displayed in production.

NOTE: MasterCard SecureCode and Amex SafeKey may not be directly tested within our current test environment. However, the process and behavior tested with the Visa test cards will be the same for MCSC and SafeKey.

When testing you may use the following test card numbers with any future expiry date. Use the appropriate test card information from the tables below: Visa and MasterCard use the same test card information, while Amex uses unique information.

Table 54: MPI test card numbers (Visa and MasterCard only)

Card Number	VERes	PARes	Action
4012001037141112 4242424242424242	Y	true	TXN – Call function to create inLine window. ACS – Send CAVV to Moneris Gateway using either the Cavv Purchase or the Cavv Pre-Authorization transaction.
4012001038488884	U	NA	Send transaction to Moneris Gateway using either the basic Purchase or the basic Pre-Authorization transaction. Set crypt_type = 7.

Table 54: MPI test card numbers (Visa and MasterCard only) (continued)

Card Number	VERes	PARes	Action
4012001038443335	N	NA	Send transaction to Moneris Gateway using either the basic Purchase or the basic Pre-Authorization transaction. Set crypt_type = 6.
4012001037461114	Y	false	Card failed to authenticate. Merchant may chose to send transaction or decline transaction. If transaction is sent, use crypt type = 7.

Table 55: MPI test card numbers (Amex only)

Card Number	VERes	Password Required?	PARes	Action
375987000000062	U	Not required	N/A	TXN – Call function to create inLine window. ACS – Send CAVV to Moneris Gateway using either the Cavv Purchase or the Cavv Pre-Authorization transaction.Set crypt_type = 7.
375987000000021	Y	Yes: test13fail	false	Card failed to authenticate. Merchant may chose to send transaction or decline transaction. If transaction is sent, use crypt type = 7.
375987000000013	N	Not required	N/A	Send transaction to Moneris Gateway using either the basic Purchase or the basic Pre-Authorization transaction. Set crypt_type = 6.
374500261001009	Y	Yes: test09	true	Card failed to authenticate. Merchant may choose to send transaction or decline transaction. Set crypt_type = 5.

VERes
The result U, Y or N is obtained by using getMessage().

PARes
The result “true” or “false” is obtained by using getSuccess().

To access the Merchant Resource Center in the test environment go to https://esqa.moneris.com/mpg.

Transactions in the test environment should not exceed $11.00.

18.8 Testing Visa Checkout

In order to test Visa Checkout you need to:

1. Create a Visa Checkout configuration profile in the Merchant Resource Center QA environment at https://esqa.moneris.com/mpg. To learn more about this, see "Creating a Visa Checkout Configuration for Testing" below.
2. Obtain a Lightbox API key to be used for Lightbox integration. To learn more about this, see "Integrating Visa Checkout Lightbox" on page 418.
3. For test card numbers specifically for use when testing Visa Checkout, see "Test Cards for Visa Checkout" on the next page

18.8.1 Creating a Visa Checkout Configuration for Testing

Once you have a test store created, you need to activate Visa Checkout in the QA environment.

To activate Visa Checkout in QA:

1. Log in to the the QA environment at https://esqa.moneris.com/mpg
2. In the Admin menu, select Visa Checkout
3. Complete the applicable fields
4. Click Save.

18.9 Test Card Numbers

Because of security and compliance reasons, the use of live credit and debit card numbers for testing is strictly prohibited. Only test credit and debit card numbers are to be used.

To test general transactions, use the following test card numbers:

Card Plan	Test Card Number
Mastercard	5454545454545454
Visa	4242424242424242
Amex	373599005095005
JCB	3566007770015365
Diners	36462462742008
Track2	5258968987035454=06061015454001060101?
Discover	6011000992927602
UnionPay	6250944000000771

18.9.1 Test Card Numbers for Level 2/3

When testing Level 2/3 transactions, use the card numbers below.

Card Brand	Test Card Number
Mastercard	5454545442424242
Visa	4242424254545454
Amex	373269005095005

18.9.2 Test Cards for Visa Checkout

Card Plan	Test Card Number
Visa	4005520201264821 (without card art)
Visa	4242424242424242 (with card art)
MasterCard	5500005555555559
American Express	340353278080900
Discover	6011003179988686

18.10 Simulator Host

The test environment has been designed to replicate the production environment as closely as possible. One major difference is that Moneris is unable to send test transactions onto the production authorization network. Therefore, issuer responses are simulated. Additionally, the requirement to emulate approval, decline and error situations dictates that certain transaction variables initiate various response and error situations.

The test environment approves and declines transactions based on the penny value of the amount sent. For example, a transaction made for the amount of $9.00 or $1.00 is approved because of the .00 penny value.

Transactions in the test environment must not exceed $11.00.

For a list of all current test environment responses for various penny values, please see the Test Environment Penny Response Table available at https://developer.moneris.com.

NOTE: These responses may change without notice. Check the Moneris Developer Portal (https://developer.moneris.com) regularly to access the latest documentation and downloads.

19 Moving to Production

19.1 Activating a Production Store Account

The steps below outline how to activate your production account so that you can process production transactions.

1. Obtain your activation letter/fax from Moneris.
2. Go to https://www.moneris.com/activate.
3. Input your store ID and merchant ID from the letter/fax and click **Activate**.
4. Follow the on-screen instructions to create an administrator account. This account will grant you access to the Merchant Resource Center.
5. Log into the Merchant Resource Center at https://www3.moneris.com/mpg using the user credentials created in step 19.1.
6. Proceed to **ADMIN** and then **STORE SETTINGS**.
7. Locate the API token at the top of the page. You will use this API token along with the store ID that you received in your letter/fax and to send any production transactions through the API.

When your production store is activated, you need to configure your store so that it points to the production host. To learn how do to this, see Configuring a Store for Production (page 447)

NOTE: For more information about how to use the Merchant Resource Center, see the Moneris Gateway Merchant Resource Center User's Guide, which is available at https://developer.moneris.com.

19.2 Configuring a Store for Production

After you have completed your testing and have activated your production store, you are ready to point your store to the production host.

To configure a store for production:

1. Change the test mode set method from `true` to `false`.
2. Change the Store ID to reflect the production store ID that you received when you activated your production store. To review the steps for activating a production store, see Activating a Production Store Account (page 447).
3. Change the API token to the production token that you received during activation.
4. If you haven't done so already, change the code to reflect the correct processing country (Canada for most merchants). For more on this, see

The table below illustrates the steps above using the relevant code (and where **X** is an alphanumeric character).

Step	Code in Testing	Changes for Production
1	No string changes for this item, only set method is altered: `mpgReq.setTestMode(true);`	Set method for production: `mpgReq.setTestMode(false);`

2	String:
	`String store_id = "store5";`
	Associated Set Method:
	`mpgReq.setStoreId(store_id);`
		String for Production:
		`String store_id = "monXXXXXXXX";`

3	String:
	`String api_token = "yesguy";`
	Associated Set Method:
	`mpgReq.setApiToken(api_token);`
		String for Production:
		`String api_token = "XXXX";`

19.2.1 Configuring an INTERAC® Online Payment Store for Production

Before you can process INTERAC® Online Payment transactions through your web site, you need to complete the certification registration process with Moneris, as described below. The production IDEBIT_MERCHNUM value is provided by Moneris after you have successfully completed the certification.

Acxsys' production INTERAC® Online PaymentGateway URL is https://gateway.interaconline.com/merchant_processor.do.

To access the Moneris Moneris Gateway production gateway URL, use the following:

Store ID: Provided by Moneris

API Token: Generated during your store activation process.

Processing country code: CA

The **production** Merchant Resource Center URL is https://www3.moneris.com/mpg/

19.2.1.1 Completing the Certification Registration - Merchants

To complete the certification registration, fax or email the information below to our Integration Support helpdesk:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
    - In both French and English
    - 120 × 30 pixels
    - Only PNG format is supported.

- Merchant business name
    - In both English and French
    - Maximum 30 characters.

- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

19.2.1.2 Third-Party Service/Shopping Cart Provider

In your product documentation, instruct your clients to provide the information below to the Moneris Gateway Integration Support helpdesk for certification registration:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
    - In both French and English
    - 120 × 30 pixels
    - Only PNG format is supported.

- Merchant business name
    - In both English and French
    - Maximum 30 characters.

- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

See 5.3.3, page 112 for additional client requirements.

19.3 Receipt Requirements

Visa and MasterCard expect certain details to be provided to the cardholder and on the receipt when a transaction is approved.

Receipts must comply with the standards outlined within the Integration Receipts Requirements. For all the receipt requirements covering all transaction scenarios, visit the Moneris Developer Portal at https://developer.moneris.com.

Production of the receipt must begin when the appropriate response to the transaction request is received by the application. The transaction may be any of the following:

- **Sale** (Purchase)
- **Authorization** (PreAuth, Pre-Authorization)
- **Authorization Completion** (Completion, Capture)
- **Offline Sale** (Force Post)
- **Sale Void** (Purchase Correction, Void)
- **Refund**.

The boldface terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction are indicated in brackets.

19.3.1 Certification Requirements

Card-present transaction receipts are required to complete certification.

Card-not-present integration
Certification is optional but highly recommended.

Card-present integration
After you have completed the development and testing, your application must undergo a certification process where all the applicable transaction types must be demonstrated, and the corresponding receipts properly generated.

Contact a Client Integration Specialist for the Certification Test checklist that must be completed and returned for verification. (See "Getting Help" on page 1 for contact details.) Be sure to include the application version of your product. Any further changes to the product after certification requires re-certification.

After the certification requirements are met, Moneris will provide you with an official certification letter.

Appendix A Definitions of Request Fields

This appendix deals with values that belong to transaction objects. For information on values that belong to the (HttpsPostRequest) connection object, see "Processing a Transaction" on page 437.

NOTE:

Alphanumeric fields allow the following characters: a-z A-Z 0-9 _ - : . @ spaces

All other request fields allow the following characters: a-z A-Z 0-9 _ - : . @ $ = /

Note that the values listed in Appendix A are not mandatory for **every** transaction. Check the transaction definition. If it says that a value is mandatory, a further description is found here.

Table 56: Request fields

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Order ID	String	50-character alphanumeric	`String order_id;`
	Merchant-defined transaction identifier that must be unique for every Purchase, PreAuth and Independent Refund transaction. No two transactions of these types may have the same order ID. For Refund, Completion and Purchase Correction transactions, the order ID must be the same as that of the original transaction. The last 10 characters of the order ID are displayed in the “Invoice Number” field on the Merchant Direct Reports. However only letters, numbers and spaces are sent to Merchant Direct. A minimum of 3 and a maximum of 10 valid characters are sent to Merchant Direct. Only the last characters beginning after any invalid characters are sent. For example, if the order ID is **1234-567890**, only **567890** is sent to Merchant Direct. If the order ID has fewer than 3 characters, it may display a blank or 0000000000 in the Invoice Number field.		

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`String amount;`
	Transaction amount. Used in a number of transactions. Note that this is different from the amount used in a Completion transaction, which is an alphanumeric value. This must contain at least 3 digits, two of which are penny values. The minimum allowable value is $0.01, and the maximum allowable value is 9999999.99. Transaction amounts of $0.00 are not allowed.		
Credit card number	String	20-character numeric (no spaces or dashes)	`String pan;`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		
Expiry date	String	4-character numeric (YYMM format)	`String expiry_date;`
	Note: This is the reverse of the date displayed on the physical card, which is MMYY.		

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
E-Commerce indicator	String	1-character alpha-numeric	`String crypt;`
	1: Mail Order / Telephone Order—Single 2: Mail Order / Telephone Order—Recurring 3: Mail Order / Telephone Order—Instalment 4: Mail Order / Telephone Order—Unknown classification 5: Authenticated e-commerce transaction (VbV/MCSC/SafeKey) 6: Non-authenticated e-commerce transaction (VbV/MCSC) 7: SSL-enabled merchant or non-authenticated e-commerce transaction (SafeKey only) **NOTE:** When processing a Cavv Purchase or Pre-Authorization for Apple Pay or Android Pay transactions whereby the merchant is using their own API to decrypt the payload, this field is mandatory. For Apple Pay or Android Pay, send the value returned in the eciIndicator or 3dsEciIndicator respectively. If the value is not present, please send the value as 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. Supported values for Apple Pay and Google Pay™are: 5: Authenticated e-commerce transaction 7: SSL-enabled merchant		

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Completion Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`String comp_amount;`
	Amount of a Completion transaction. This may not be equal to the amount value (described on page 451), which appeared in the original Pre-Authorization transaction.		
Shipping Indicator[1]	String	1-character alpha-numeric	`String ship_indicator;`

	Used to identify completion transactions that require multiple shipments, also referred to as multiple completions. By default, if the shipping indicator is not passed, all completions are listed as final completions. To indicate that the completion is to be left open by the issuer as supplemental shipments or completions are pending, a value of P is submitted.
	
	Possible values:
	
	P = Partial
	
	F = Final		

[1]Available to Canadian integrations only.

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Transaction number	String	255-character alphanumeric	`String txn_number;`
	Used when performing follow-on transactions. (That is, Completion, Purchase Correction or Refund.) This must be the value that was returned as the transaction number in the response of the original transaction. When performing a Completion, this value must reference the Pre-Authorization. When performing a Refund or a Purchase Correction, this value must reference the Completion or the Purchase.		
Authorization code	String	8-character alphanumeric	`String auth_code;`
	Authorization code provided in the transaction response from the issuing bank. This is required for Force Post transactions.		
ECR number	String	8-character alphanumeric	`String ecr_no;`
	Electronic cash register number, also referred to as TID or Terminal ID.		
MPI transaction values			
XID	String	20-character alphanumeric	`String xid;`
	Can also be used as your order ID when using Moneris Gateway. Fixed length — must be exactly 20 characters.		
MD (Merchant Data)	String	1024-character alphanumeric	`String MD;`
	Information to be echoed back in the response.		
Merchant URL	String	Variable length	`String merchantUrl;`
	URL to which the MPI response is to be sent.		
Accept	String	Variable length	`String accept;`
	MIME types that the browser accepts		
User Agent	String	Variable length	`String userAgent;`
	Browser details		
PARes	String	Variable length	(Not shown)
	Value passed back to the API during the TXN, and returned to the MPI when an ACS request is made.		

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Cardholder Authentication Verification Value (CAVV)	String	50-character alpha-numeric	`String cavv;`
	Value provided by the Moneris MPI or by a third-party MPI. It is part of a Verified by Visa/MasterCard SecureCode/American Express SafeKey transaction. **NOTE:** For Apple Pay and Android Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram.		
Vault transaction values			
Data key	String	28-character alpha-numeric	`String data_key;`
	Profile identifier that all future financial Vault transactions (that is, they occur after the profile was registered by a Vault Add Credit Card- ResAddCC, Vault Encrypted Add Credit Card - EncResAddCC, Vault Tokenize Credit Card - ResTokenizeCC, Vault Add Temporary Token - ResTempAdd or Vault Add Token - ResAddToken transaction) will use to associate with the saved information. The data key is generated by Moneris, and is returned to the merchant (via the Receipt object) when the profile is first registered.		
Duration	String	3-character numeric	`String duration;`
	Amount of time the temporary token should be available, up to 900 seconds.		

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Data key format[1]	String	2-character alpha-numeric	`String data_key_format;`
	This field will specify the data key format being returned. If left blank, Data Key format will default to 25-character alphanumeric. Valid values: no value sent or 0 = 25-character alpha-numeric Data Key By using the following values, a unique token is generated specifically for the PAN that is presented for tokenization. Any subsequent tokenization requests for the same PAN will result in the same token 0U = 25-character alpha-numeric Data Key, Unique		
Mag Swipe transaction values			
POS code	String	20-character numeric	`String pos_code;`
	Under normal presentment situations, the value is `00`. If a Pre-Authorization transaction was card-present and keyed-in, then the POS code for the corresponding Completion transaction is `71`. In an unmanned kiosk environment where the card is present, the value is `27`. If the solution is not "merchant and cardholder present", contact Moneris for the proper POS code.		
Track2 data	String	40-character alphanumeric	`String track2;`
	Retrieved from the mag stripe of a credit card by swiping it through a card reader, **or** the "fund guarantee" value returned by the INTERAC® Online Payment system.		
Encrypted track2 data	String	Variable length	`String enc_track2;`
	String that is retrieved by swiping or keying in a credit card number through a Moneris-provided encrypted mag swipe card reader. It is part of an encrypted keyed or swiped transaction only. This string must be retrieved by a specific device. (See below for the list of current available devices.)		

[1]Available to Canadian integrations only.

Table 56: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Device type	String	30-character alpha-numeric	`String device_type;`
	Type of encrypted mag swipe reader that was read the credit card. This must be a Moneris-provided device so that the values are properly encrypted and decrypted. This field is case-sensitive. Available values are: "idtech_bdk"		

Note that the values listed in Appendix A are not supported by **every** transaction. Check the transaction definition. If it says that a value is optional, a further description is found here.

Table 57: Optional transaction values

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Customer ID	String	30-character alphanumeric	`String cust_id;`
	This can be used for policy number, membership number, student ID, invoice number and so on. This field is searchable from the Moneris Merchant Resource Center.		
Status Check	String	true/false	`String status_check;`
	See .		
Dynamic descriptor	String	20-character alphanumeric Combined with merchant's business name cannot exceed 25 characters.	`String dynamic_descriptor;`
	Merchant-defined description sent on a per-transaction basis that will appear on the credit card statement appended to the merchant’s business name.		

Table 57: Optional transaction values (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Wallet indicator[1]	String	3-character alphanumeric	`String wallet_indicator;`
	Optional value to indicate when the credit card details were collected from a wallet such as Apple Pay, Android Pay, Visa Checkout, MasterCard MasterPass. This field is applicable to Apple Pay and Android Pay transactions whereby the merchant is using their own API to decrypt the payload. This is a **mandatory** field for these types of Apple Pay and Android Pay transactions. • Apple Pay and Android Pay wallet indicator is applicable to Cavv Purchase – Apple Pay and Cavv Pre-Authorization – Apple Pay • Visa Checkout and MasterCard MasterPass wallet indicator is applicable to basic Purchase and Pre-Authorization Possible values are: • APP = Apple Pay In-App • APW = Apple Pay on the Web • ANP = Android Pay In-App • VCO = Visa Checkout • MMP = MasterCard MasterPass **NOTE:** Please note that if this field is included to indicate Apple Pay or Android Pay, then Convenience Fee is not supported.		
Vault transaction values			
Phone number	String	30-character alphanumeric	`String phone;`
	Phone number of the customer. Can be sent in when creating or updating a Vault profile.		
Email address	String	30-character alphanumeric	`String email;`
	Email address of the customer. Can be sent in when creating or updating a Vault profile.		
Additional notes	String	30-character alphanumeric	`String note;`
	This optional field can be used for supplementary information to be sent in with the transaction. This field can be sent in when creating or updating a Vault profile.		

For information about Customer Information request fields see 15 Customer Information

[1]Available to Canadian integrations only.

For information about Address Verification Service (AVS) request fields see 10.1 Address Verification Service

For information about Card Validation Digits (CVD) request fields see

For information about Recurring Billing request fields see Appendix A Recurring Billing.

For information about Convenience Fee request fields see Appendix A Convenience Fee.

For information about Level 2/3 Visa, Level 2/3 MasterCard and Level 2/3 American Express, see A.3 Definition of Request Fields for Level 2/3 - Visa, A.5 Definition of Request Fields for Level 2/3 - Amex

A.1 Definition of Request Fields – Credential on File

Variable Name	Type and Limits	Description
issuer ID **NOTE:** This variable is required for all merchant-initiated transactions following the first one; upon sending the first transaction, the issuer ID value is received in the transaction response and then used in subsequent transaction requests.	*String* 15-character alphanumeric variable length	Unique identifier for the cardholder's stored credentials Sent back in the response from the card brand when processing a Credential on File transaction If the cardholder's credentials are being stored for the first time, and the issuer ID was returned in the response, you must save the issuer ID on your system to use in subsequent Credential on File transactions (applies to merchant-initiated transactions only) The issuer ID must be saved to your systems when returned from Moneris Gateway in the response data, regardless if the value was received or not As a best practice, if the issuer ID is not returned and you received a value of NULL instead, store that value and send it in the subsequent transaction
payment indicator	*String* 1-character alphabetic	Indicates the current or intended use of the credentials Possible values for first transactions: C - unscheduled Credential on File (first trans-

Variable Name	Type and Limits	Description
		actions only) R - recurring Possible values for subsequent transactions: R - recurring U - unscheduled merchant-initiated transaction Z - unscheduled customer-initiated transaction
payment information	*String* 1-character numeric	Describes whether the transaction is the first or subsequent in the series Possible values: 0 - first transaction in a series (storing payment details provided by the cardholder) 2 - subsequent transactions (using previously stored payment details)

A.2 Definition of Request Fields – Recurring

Recurring Billing Info Object Request Fields

Variable Name	Type and Limits	Description
number of recurs `num_recurs`	*String* numeric 1-99	The number of times that the transaction must recur
period `period`	*String* numeric 1-999	Number of recur unit intervals that must pass between recurring billings
start date `start_date`	*String* YYMMDD format	Date of the first future recurring billing transaction; this must be a date in the future If an additional charge will be made immediately, the start now variable must be set to true

Variable Name	Type and Limits	Description
start now `start_now`	*String* true/false	Set to true if a charge will be made against the card immediately; otherwise set to false When set to false, use Card Verification prior to sending the Purchase with Recurring Billing and Credential on File objects **NOTE:** Amount to be billed immediately can differ from the subsequent recurring amounts
recurring amount `recur_amount`	*String* 10-character decimal, minimum three digits Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	Dollar amount of the recurring transaction This amount will be billed on the start date, and then billed repeatedly based on the interval defined by period and recur unit
recur unit `recur_unit`	*String* day, week, month or eom	Unit to be used as a basis for the interval Works in conjunction with the period variable to define the billing frequency

A.3 Definition of Request Fields for Level 2/3 - Visa

Table 1 Visa - Corporate Card Common Data - Level 2 Request Fields

Req*	Field Name	Limits	Set Method	Description
Y	National Tax	12-character decimal	`TRANSACTIONNAME .setNationalTax (national_tax);`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01

Req*	Field Name	Limits	Set Method	Description
				Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference Number	20-character alpha-numeric	`TRANSACTIONNAME .setMerchantVatNo (merchant_vat_no);`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`TRANSACTIONNAME .setLocalTax (local_tax);`	Must reflect the amount of Local Tax (PST or QST) appearing on the invoice If Local Tax included then must not be all spaces or all zer-oes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alpha-numeric	`TRANSACTIONNAME .setLocalTaxNo (local_tax_no);`	Merchant's Local Tax (PST/QST) Regis-tration Number

Req*	Field Name	Limits	Set Method	Description
				Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alphanumeric	`TRANSACTIONNAME .setCustomerVatNo (customer_vat_no);`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	`TRANSACTIONNAME .setCri(cri);`	Value which the customer may choose to provide to the supplier at the point of sale – must be provided if given by the customer
N	Customer Code	17-character alphanumeric	`TRANSACTIONNAME .setCustomerCode (customer_code);`	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alphanumeric	`TRANSACTIONNAME .setInvoiceNumber`	Optional invoice number field

Req*	Field Name	Limits	Set Method	Description
			`(invoice_number);`	that will not be passed along to Visa, but will be included on Moneris report-ing

*Y = Required, N = Optional, C = Conditional

Table 2 Visa - Corporate Card Common Data- Level 2 Request Fields (VSPurcha)

Req	Variable Name	Field Name	Size/Type	Description
C*	Buyer Name	buyer_name	30-character alpha-numeric	Buyer/Receipient Name *only required by CRA if transaction is >$150
C*	Local tax rate	local_tax_rate	4-character decimal	Indicates the detailed tax rate applied in rela-tionship to a local tax amount **EXAMPLE:** 8% PST should be 8.0. maximum 99.99 *Must be provided if Local Tax (PST or QST) applies.
N	Duty Amount	duty_amount	9-character decimal	Duty on total pur-chase amount A minus sign means 'amount is a credit', plus sign or no sign means 'amount is a debit' maximum without sign is 999999.99
N	Invoice Discount	discount_treatment	1-character numeric	Indicates how the

Req	Variable Name	Field Name	Size/Type	Description
	Treatment			merchant is managing discounts Must be one of the following values: 0 - if no invoice level discounts apply for this invoice 1 - if Tax was calculated on Post-Discount totals 2 - if Tax was calculated on Pre-Discount totals
N	Invoice Level Discount Amount	discount_amt	9-character decimal	Amount of discount (if provided at the invoice level according to the Invoice Discount Treatment) Must be non-zero if Invoice Discount Treatment is 1 or 2 Minimum amount is 0.00 and maximum is 999999.99
C*	Ship To Postal Code / Zip Code	ship_to_pos_code	10-character alphanumeric	The postal code or zip code for the destination where goods will be delivered *Required if shipment is involved Full alpha postal code - Valid ANA<space>NAN format required if shipping to an address within Canada
C	Ship From Postal	ship_from_pos_code	10-character alpha-	The postal code or

Req	Variable Name	Field Name	Size/Type	Description
	Code / Zip Code		numeric	zip code from which items were shipped For Canadian addresses,requires full alpha postal code for the merchant with Valid ANA<space>NAN format
C*	Destination Country Code	des_cou_code	2-character alphanumeric	Code of country where purchased goods will be delivered Use ISO 3166-1 alpha-2 format **NOTE:** Required if it appears on the invoice for an international transaction
Y	Unique VAT Invoice Reference Number	vat_ref_num	25-character alphanumeric	Unique Value Added Tax Invoice Reference Number Must be populated with the invoice number and this cannot be all spaces or zeroes
Y	Tax Treatment	tax_treatment	1-character numeric	Must be one of the following values: 0 = Net Prices with tax calculated at line item level; 1 = Net Prices with tax calculated at invoice level; 2 = Gross prices given with tax information provided at line item

Req	Variable Name	Field Name	Size/Type	Description
				level; 3 = Gross prices given with tax information provided at invoice level; 4 = No tax applies (small merchant) on the invoice for the transaction
N	Freight/Shipping Amount (Ship Amount)	freight_amount	9-character decimal	Freight charges on total purchase If shipping is not provided as a line item it must be provided here, if applicable Signed monetary amount: minus sign means 'amount is a credit', plus sign or no sign means 'amount is a debit', maximum without sign is 999999.99
C	GST HST Freight Rate	gst_hst_freight_rate	4-character decimal	Rate of GST (excludes PST) or HST charged on the shipping amount (in accordance with the Tax Treatment) If Freight/Shipping Amount is provided then this (National GST or HST) tax rate must be provided. Monetary amount, maximum is 99.99. Such as 13% HST is 13.00

Req	Variable Name	Field Name	Size/Type	Description
C	GST HST Freight Amount	gst_hst_freight_amount	9-character decimal	Amount of GST (excludes PST) or HST charged on the shipping amount If Freight/Shipping Amount is provided then this (National GST or HST) tax amount must be provided if taxTreatment is 0 or 2 Signed monetary amount: maximum without sign is 999999.99.

Table 3 Visa - Line Item Details - Level 3 Request Fields (VSPurchl)

Req	Variable Name	Field Name	Size/Type	Description
C	Item Commodity Code	item_com_code	12-character alphanumeric	Line item Commodity Code (if this field is not sent, then productCode must be sent)
Y	Product Code	product_code	12-character alphanumeric	Product code for this line item – merchant's product code, manufacturer's product code or buyer's product code Typically this will be the SKU or identifier by which the merchant tracks and prices the item or service This should always be provided for

Req	Variable Name	Field Name	Size/Type	Description
				every line item
Y	Item Description	item_description	35-character alpha-numeric	Line item description

Y	Item Quantity	item_quantity	12-character decimal	Quantity invoiced for this line item
				
				Up to 4 decimal places supported, whole numbers are accepted
				
				Minimum = 0.0001
				
				Maximum = 999999999999

Y	Item Unit of Measure	item_uom	2-character alphanumeric	Unit of Measure Use ANSI X-12 EDI Allowable Units of Measure and Codes

Y	Item Unit Cost	unit_cost	12-character decimal	Line item cost per unit
				
				2-4 decimal places accepted
				
				Minimum = 0.0001
				
				Maximum = 999999.9999

N	VAT Tax Amount	vat_tax_amt	12-character decimal	Any value-added tax or other sales tax amount
				
				Must have 2 decimal places
				
				Minimum = 0.01
				
				Maximum = 999999.99

N	VAT Tax Rate	vat_tax_rate	4-character decimal	Sales tax rate

Req	Variable Name	Field Name	Size/Type	Description
				EXAMPLE: 8% PST should be 8.0 maximum 99.99
Y	Discount Treat-ment	discount_treatmentL	1-character numeric	Must be one of the following values: 0 if no invoice level dis-counts apply for this invoice 1 if Tax was calculated on Post-Discount totals 2 if Tax was calculated on Pre-Discount totals.
C	Discount Amount	discount_amtL	12-character decimal	Amount of dis-count, if provided for this line item according to the Line Item Discount Treatment Must be non-zero if Line Item Discount Treatment is 1 or 2 Must have 2 decimal places Minimum = 0.01 Maximum = 999999.99

A.4 Definition of Request Fields for Level 2/3 - MasterCard

Table 1 Objects - Level 2/3 MasterCard

MCCorpais Objects	Description
MCCorpac	Corporate Card Common data
MCCorpal	Line Item Details

Table 2 MasterCard - Corporate Card Common Data (MCCorpac) - Level 2 Request Fields

Req	Variable Name	Field Name	Size/Type	Description
N	AustinTetraNumber	Austin-Tetra Number	15-character alpha-numeric	Merchant’s Austin-Tetra Number
N	NaicsCode	NAICS Code	15-character alpha-numeric	North American Industry Classification System (NAICS) code assigned to the merchant
N	CustomerCode	Customer Code	25-character alpha-numeric	A control number, such as purchase order number, project number, department allocation number or name that the purchaser supplied the merchant. Left-justified; may be spaces
N	UniqueInvoiceNumber	Unique Invoice Number	17-character alpha-numeric	Unique number associated with the individual transaction provided by the merchant
N	CommodityCode	Commodity Code	15-character alpha-numeric	Code assigned by the merchant that best categorizes the item(s) being purchased
N	OrderDate	Order Date	6-character numeric	The date the item was ordered. If present, must contain a valid date in the format YYMMDD.
N	CorporationVatNumber	Corporation VAT Number	20-character alpha-numeric	Contains a corporation’s value added tax (VAT) number
N	CustomerVatNumber	Customer VAT Number	20-character alpha-numeric	Contains the VAT number for the customer/cardholder used to identify the customer when purchasing goods and services from the merchant
N	FreightAmount	Freight Amount	12-character decimal	The freight on the total purchase. Must have 2 decimals
N	DutyAmount	Duty Amount	12-character decimal	The duty on the total purchase, Must have 2 decimals

Req	Variable Name	Field Name	Size/Type	Description
N	DestinationProvinceCode	Destination State / Province Code	3-character alpha-numeric	State or Province of the country where the goods will be delivered. Left justified with trailing spaces. e.g., ONT - Ontario
N	DestinationCountryCode	Destination Country Code	3-character alpha-numeric	The country code where goods will be delivered. Left justified with trailing spaces. e.g., CAN - Canada
N	ShipFromPosCode	Ship From Postal Code	10-character alpha-numeric	The postal code or zip code from which items were shipped
N	ShipToPosCode	Destination Postal Code	10-character alpha-numeric	The postal code or zip code where goods will be delivered
N	AuthorizedContactName	Authorized Contact Name	36-character alpha-numeric	Name of an individual or company contacted for company authorized purchases
N	AuthorizedContactPhone	Authorized Contact Phone	17-character alpha-numeric	Phone number of an individual or company contacted for company authorized purchases
N	AdditionalCardAcceptordata	Additional Card Acceptor Data	40-character alpha-numeric	Information pertaining to the card acceptor
N	CardAcceptorType	Card Acceptor Type	8-character alpha-numeric	Various classifications of business ownership characteristics This field takes 8 characters. Each character represents a different component, as follows: 1st character represents ‘Business Type’ and contains a code to identify the specific classification or type of business: 1. Corporation

Req	Variable Name	Field Name	Size/Type	Description
				2. Not known 3. Individual/Sole Proprietorship 4. Partnership 5. Association/Estate/Trust 6. Tax Exempt Organizations (501C) 7. International Organization 8. Limited Liability Company (LLC) 9. Government Agency 2nd character represents 'Business Owner Type'. Contains a code to identify specific characteristics about the business owner. 1 - No application classification 2 - Female business owner 3 - Physically handicapped female business owner 4 - Physically handicapped male business owner 0 - Unknown 3rd character represents 'Business Certification Type'. Contains a code to identify specific characteristics about the business certification type, such as small business, disadvantaged, or other certification type: 1 - Not certified 2 - Small Business Administration (SBA) certification small busi-

Req	Variable Name	Field Name	Size/Type	Description
				ness 3 - SBA certification as small disadvantaged business 4 - Other government or agency-recognized certification (such as Minority Supplier Development Council) 5 - Self-certified small business 6 - SBA certification as small and other government or agency-recognized certification 7 - SBA certification as small disadvantaged business and other government or agency-recognized certification 8 - Other government or agency-recognized certification and self-certified small business A - SBA certification as 8 (a) B - Self-certified small disadvantaged business (SDB) C - SBA certification as HUBZone 0 - Unknown 4th character represents 'Business Racial/Ethnic Type'. Contains a code identifying the racial or ethnic type of the majority owner of the business. 1 - African American 2 - Asian Pacific American

Req	Variable Name	Field Name	Size/Type	Description
				3 - Subcontinent Asian American 4 - Hispanic American 5 - Native American Indian 6 - Native Hawaiian 7 - Native Alaskan 8 - Caucasian 9 - Other 0 - Unknown 5th character represents 'Business Type Provided Code' Y - Business type is provided. N - Business type was not provided. R - Card acceptor refused to provide business type 6th character represents 'Business Owner Type Provided Code' Y - Business owner type is provided. N - Business owner type was not provided. R - Card acceptor refused to provide business type 7th character represents 'Business Certification Type Provided Code' Y - Business certification type is provided. N - Business certification type was not provided. R - Card acceptor

Req	Variable Name	Field Name	Size/Type	Description
				refused to provide business type 8th character represents 'Business Racial/Ethnic Type' Y - Business racial/ethnic type is provided. N - Business racial/ethnic type was not provided. R - Card acceptor refused to provide business racial/ethnic type
N	CardAcceptorTaxId	Card Acceptor Tax ID	20-character alpha-numeric	US Federal tax ID number for value added tax (VAT) ID.
N	CardAcceptorReferenceNumber	Card Acceptor Reference Number	25-character alpha-numeric	Code that facilitates card acceptor/corporation communication and record keeping
N	CardAcceptorVatNumber	Card Acceptor VAT Number	20-character alpha-numeric	Value added tax (VAT) number for the card acceptor location used to identify the card acceptor when collecting and reporting taxes
C-*	Tax	Tax	up to 6 arrays	Can have up to 6 arrays contains different tax details. See Tax Array below for each field description. *This field is conditionally mandatory — if you use this array, you must fill in all tax array fields as listed in the Tax Array Request Fields below.

Table 3 MasterCard - Line Item Details (MCCorpal) - Level 3 Request Fields

Req	Variable Name	Field Name	Size/Type	Description
N	CustomerCode	Customer Code	25-character alpha-numeric	A control number, such as purchase order number, project number, department allocation number or name that the purchaser supplied the merchant. Left-justified; may be spaces
N	LineItemDate	Line Item Date	6-character numeric	The purchase date of the line item referenced in the associated Corporate Card Line Item Detail. YYMMDD format
N	ShipDate	Ship Date	6-character numeric	The date the merchandise was shipped to the destination. YYMMDD format
N	OrderDate	Order Date	6-character numeric	The date the item was ordered YYMMDD format
Y	ProductCode	Product Code	12-character alpha-numeric	Line item Product Code (if this field is not sent, then itemComCode) If the order has a Freight/Shipping line item, the productCode value has to be “Freight/Shipping” If the order has a Discount line item,

Req	Variable Name	Field Name	Size/Type	Description
				the productCode value has to be “Discount”
Y	ItemDescription	Item Description	35-character alpha-numeric	Line Item description
Y	ItemQuantity	Item Quantity	12-character alpha-numeric	Quantity of line item
Y	UnitCost	Unit Cost	12-character decimal	Line item cost per unit. Must contain a minimum of 2 decimal places, up to 5 decimal places supported. Minimum amount is 0.00001 and maximum is 999999.99999
Y	ItemUnitMeasure	Item Unit Measure	12-character alpha-numeric	The line item unit of measurement code
Y	ExtItemAmount	Extended Item Amount	9-character decimal	Contains the individual item amount that is normally calculated as price multiplied by quantity Must contain 2 decimal places Minimum amount is 0.00 and maximum is 999999.99
N	DiscountAmount	Discount Amount	9-character decimal	Contains the item discount amount Must contain 2 decimal places

Req	Variable Name	Field Name	Size/Type	Description
				Minimum amount is 0.00 and maximum is 999999.99
N	CommodityCode	Commodity Code	15-character alpha-numeric	Code assigned to the merchant that best categorizes the item(s) being purchased
C*	Tax	Tax	Up to 6 arrays	Can have up to 6 arrays contains different tax details. See Tax Array below for each field description. *This field is conditionally mandatory — if you use this array, you must fill in all tax array fields as listed in the Tax Array Request Fields below.

Table 4 Tax Array Request Fields - MasterCard Level 2/3 Transactions

Req	Variable Name	Field Name	Size/Type	Description
M	tax_amount	Tax Amount	12-character decimal	Contains detail tax amount for purchase of goods or service Must be 2 decimal places Maximum 999999.99
M	tax_rate	Tax Rate	5-character decimal	Contains the detailed tax rate applied in relationship to a spe-

Req	Variable Name	Field Name	Size/Type	Description
				cific tax amount **EXAMPLE:** 5% GST should be '5.0' or or 9.975% QST should be '9.975' May contain up to 3 decimals, minimum 0.001, maximum up to 9999.9
M	tax_type	Tax Type	4-character alphanumeric	Contains tax type such as GST,QST,PST,HST
M	tax_id	Tax ID	20-character alphanumeric	Provides an identification number used by the card acceptor with the tax authority in relationship to a specific tax amount such as GST/HST number
M	tax_included_in_sales	Tax included in sales indicator	1-character alphanumeric	This is the indicator used to reflect additional tax capture and reporting. Valid values are: Y = Tax included in total purchase amount N = Tax not included in total purchase amount

A.5 Definition of Request Fields for Level 2/3 - Amex

Table 1 Amex- Level 2/3 Request Fields - Table 1 - Heading Fields

Req	Variable Name	Field Name	Size/Type	Description
C	big04	Purchase Order Number	22-character alphanumeric	The cardholder supplied Purchase Order Number, which is entered by the merchant at the point-of-sale This entry is used in the Statement/Reporting process and may include accounting information specific to the client Mandatory if the merchant's customer provides a Purchase Order Number
N	big05	Release Number	30-character alphanumeric	A number that identifies a release against a Purchase Order previously placed by the parties involved in the transaction
N	big10	Invoice Number	8-character alphanumeric	Contains the Amex invoice/reference number
Y	n101	Entity Identifier Code	2-character alphanumeric	Supported values: ‘R6’ - Requester (required) ‘BG’ - Buying Group (optional) ‘SF’ - Ship From (optional) ‘ST’ - Ship To (optional) ‘40’ - Receiver (optional)

Req	Variable Name	Field Name	Size/Type	Description
Y	n102	Name	40-character alpha-numeric	**n101 code** **n102 meaning** R6 Requester Name BG Buying Group Name SF Ship From Name ST Ship To Name 40 Receiver Name
N	n301	Address	40-character alpha-numeric	Address
N	n401	City	30-character alpha-numeric	City
N	n402	State or Province	2-character alpha-numeric	State or Province
N	n403	Postal Code	15-character alpha-numeric	Postal Code
Y	ref01	Reference Identification Qualifier	2-character alpha-numeric	This element may contain the following qualifiers for the corresponding occurrences of the N1Loop:

Req	Variable Name	Field Name	Size/Type	Description
				n101 value / **ref01 denotation** R6: Supported values: 4C - Shipment Destination Code (mandatory); CR - Customer Reference Number (conditional) BG: n/a SF: n/a ST: n/a 40: n/a
Y	ref02	Reference Identification	15-character alphanumeric	VR is the Vendor ID Number, other codes describe the following: **ref01 code** / **ref02 denotation** 4C: Ship to Zip or Canadian Postal Code (required) CR: Cardmember Reference Number (optional)

Table 2 Amex - Level 2/3 Request Fields - Table 2 - Detail Fields

Req	Variable Name	Field Name	Size/Type	Description
Y	it102	Line Item Quantity Invoiced	10-character decimal	Quantity of line item. Up to 2 decimal places supported. Minimum amount is 0.0 and max-

Req	Variable Name	Field Name	Size/Type	Description
				imum is 9999999999.
Y	it103	Unit or Basis for Meas-urement Code	2-character alphanumeric	The line item unit of measurement code Must contain a code that specifies the units in which the value is expressed or the manner in which a measurement is taken **EXAMPLE:** EA = each, E5=inches See ANSI X-12 EDI Allowable Units of Measure and Codes for the list of codes
Y	it104	Unit Price	15-character decimal	Line item cost per unit Must contain 2 decimal places Minimum amount is 0.00 and max-imum is 999999.99
N	it105	Basis or Unit Price Code	2-character alphanumeric	Code identifying the type of unit price for an item **EXAMPLE:** DR = dealer, AP = advise price See ASC X12 004010 Element 639 for list of codes
N	it10618	Product/Service ID	2-character alphanumeric	Supported values:

Req	Variable Name	Field Name	Size/Type	Description
		Qualifier		'MG' - Manufacturer's Part Number 'VC' - Supplier Catalog Number 'SK' - Supplier Stock Keeping Unit Number 'UP' - Universal Product Code 'VP' – Vendor Part Num-ber 'PO' – Purchase Order Number 'AN' – Client Defined Asset Code
N	it10719	Product/Service ID	**it10618** | **it10719 - size/type** VC | 20-character alphanumeric PO | 22-character alphanumeric Other | 30-character alphanumeric	Product/Service ID corresponds to the preceding qualifier defined in it10618 The maximum length depends on the qualifier defined in it10618
C	txi01	Tax Type code	2-character alphanumeric	Supported values: 'CA' – City Tax (optional) 'CT' – County/Tax (optional) 'EV' – Environmental Tax (optional) 'GS' – Good and Ser-vices Tax (GST) (optional) 'LS' – State and Local Sales Tax (optional) 'LT' – Local Sales Tax (optional) 'PG' – Provincial Sales Tax (PST) (optional) 'SP' – State/Provincial Tax a.k.a. Quebec Sales

Req	Variable Name	Field Name	Size/Type	Description
				Tax (QST) (optional) 'ST' – State Sales Tax (optional) 'TX' – All Taxes (required) 'VA' – Value-Added Tax a.k.a. Canadian Harmonized Sales Tax (HST) (optional)
C	txi02	Monetary Amount	6-character decimal	This element may contain the monetary tax amount that corresponds to the Tax Type Code in txi01 **NOTE:** If txi02 is used in mandatory occurrence txi01=TX, txi02 must contain the total tax amount applicable to the entire invoice (transaction) If taxes are not applicable for the entire invoice (transaction), txi02 must be 0.00. The maximum value that can be entered in this field is "9999.99", which is $9,999.99 (CAD) A debit is entered as: 9999.99 A credit is entered as: –9999.99
C	txi03	Percent	10-character decimal	Contains the tax percentage (in decimal format) that corresponds to the tax type code defined in txi01

Req	Variable Name	Field Name	Size/Type	Description
				Up to 2 decimal places supported
C	txi06	Tax Exempt Code	1-character alphanumeric	This element may contain the Tax Exempt Code that identifies the exemption status from sales and tax that corresponds to the Tax Type Code in txi01 Supported values: 1 – Yes (Tax Exempt) 2 – No (Not Tax Exempt) 4 – Not Exempt/For Resale A – Labor Taxable, Material Exempt B – Material Taxable, Labor Exempt C – Not Taxable F – Exempt (Goods / Services Tax) G – Exempt (Provincial Sales Tax) L – Exempt Local Ser-vice R – Recurring Exempt U – Usage Exempt
Y	pam05	Line Item Extended Amount	8-character decimal	Contains the indi-vidual item amount that is normally cal-culated as price mul-tiplied by quantity Must contain 2 decimal places Minimum amount

Req	Variable Name	Field Name	Size/Type	Description
				is 0.00 and maximum is 99999.99
Y	pid05	Line Item Description	80-character alphanumeric	Line Item description Contains the description of the individual item purchased This field pertain to each line item in the transaction

Table 3 Amex - Level 2/3 Request Fields - Table 3 - Summary Fields

Req	Variable Name	Field Name	Size/Type	Description
C	txi01	Tax Type code	2-character alphanumeric	Supported values: 'CA' – City Tax (optional) 'CT' – County/Tax (optional) 'EV' – Environmental Tax (optional) 'GS' – Good and Services Tax (GST) (optional) 'LS' – State and Local Sales Tax (optional) 'LT' – Local Sales Tax (optional) 'PG' – Provincial Sales Tax (PST) (optional) 'SP' – State/Provincial Tax a.k.a. Quebec Sales Tax (QST) (optional) 'ST' – State Sales Tax (optional) 'TX' – All Taxes (required)

Req	Variable Name	Field Name	Size/Type	Description
				'VA' – Value-Added Tax a.k.a. Canadian Harmonized Sales Tax (HST) (optional)
C	txi02	Monetary Amount	6-character decimal	This element may contain the monetary tax amount that corresponds to the Tax Type Code in txi01 **NOTE:** If txi02 is used in mandatory occurrence txi01=TX, txi02 must contain the total tax amount applicable to the entire invoice (transaction) If taxes are not applicable for the entire invoice (transaction), txi02 must be 0.00. The maximum value that can be entered in this field is "9999.99", which is $9,999.99 (CAD) A debit is entered as: 9999.99 A credit is entered as: –9999.99
C	txi03	Percent	10-character decimal	Contains the tax percentage (in decimal format) that corresponds to the tax type code defined in txi01 Up to 2 decimal places supported
C	txi06	Tax Exempt Code	1-character alphanumeric	Supported values: 1 – Yes (Tax Exempt)

Req	Variable Name	Field Name	Size/Type	Description
				2 – No (Not Tax Exempt) 4 – Not Exempt/For Resale A – Labor Taxable, Material Exempt B – Material Taxable, Labor Exempt C – Not Taxable F – Exempt (Goods / Services Tax) G – Exempt (Provincial Sales Tax) L – Exempt Local Service R – Recurring Exempt U – Usage Exempt

A.6 Definition of Request Fields – MCP

Variable Name	Type and Limits	Description
MCP version number	*String* numeric current version is 1.0	Release version number for MCP
cardholder amount	*String* 12-character numeric smallest discrete unit of foreign currency	Amount, in units of foreign currency, the cardholder will be charged on the transaction
cardholder currency code	*String* 3-character numeric	ISO code representing the foreign currency of the cardholder

Optional MCP fields

Variable Name	Type and Limits	Description
MCP rate token	*String* N/A	Token representing a temporarily locked-in foreign exchange rate, obtained in the response of the MCP Get Rate transaction and used in subsequent MCP financial transaction requests in order to redeem that rate

MCP Get Rate transaction request fields

Variable Name	Type and Limits	Description
MCP version number	*String* numeric current version is 1.0	Release version number for MCP
rate transaction type	*String* 1-character alphabetic	Value representing the type of subsequent transaction request that the rate token will be used for. Allowable values: P – Purchase R – Refund
MCP Rate Info	*Object* N/A	Nested object in the MCP Get Rate transaction containing the **add cardholder amount** and **add merchant settlement** fields

MCP Rate Info object request fields

At least one of the following variables must be sent:

Variable Name	Type and Limits	Description
add cardholder amount	*String array* 12-character numeric, 3-character numeric (smallest discrete unit of foreign currency, currency code)	A string array representing: • the amount, in units of foreign currency, the cardholder will be charged, and • the ISO currency code corresponding to the foreign cur-

Variable Name	Type and Limits	Description
		rency of the cardholder
add merchant settlement amount	*String array* 12-character numeric, 3-character numeric (amount in CAD pennies, currency code)	A string array representing: • the amount the merchant will receive in the transaction, in Canadian dollars • the ISO currency code corresponding to the foreign currency of the cardholder

A.7 Definition of Request Fields – Offlinx™

Applies to Offlinx™ integration only

Variable Name	Type and Limits	Description
card match ID	*String* 50-character alphanumeric	Corresponds to the Transaction ID used for the Offlinx™ Card Match Pixel Tag, a unique identifier created by the merchant Must be unique value for each transaction

Appendix B Definitions of Response Fields

Table 58: Receipt object response values

Value	Type	Limits	Get Method
	Description		
General response fields			
Card type	String	2-character alphabetic (min. 1)	`receipt.getCardType();`
	Represents the type of card in the transaction, e.g., Visa, Mastercard. Possible values: • V = Visa • M = Mastercard • AX = American Express • DC = Diner's Card • NO = Novus/Discover • SE = Sears • D = Debit • C1 = JCB		
Transaction amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`receipt.getTransAmount();`
	Transaction amount that was processed.		
Transaction number	String	255-character alphanumeric	`receipt.getTxnNumber();`
	Gateway Transaction identifier often needed for follow-on transactions (such as Refund and Purchase Correction) to reference the originally processed transaction.		
Receipt ID	String	50-character alphanumeric	`receipt.getReceiptId();`
	Order ID that was specified in the transaction request.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Transaction type	String	2-character alphanumeric	`receipt.getTransType();`
	• 0 = Purchase • 1 = Pre-Authorization • 2 = Completion • 4 = Refund • 11 = Void		
Reference number	String	18-character numeric	`receipt.getReferenceNum();`
	Terminal used to process the transaction as well as the shift, batch and sequence number. This data is typically used to reference transactions on the host systems, and must be displayed on any receipt presented to the customer. This information is to be stored by the merchant. Example: 660123450010690030 • 66012345: Terminal ID • 001: Shift number • 069: Batch number • 003: Transaction number within the batch.		
Response code	String	3-character numeric	`receipt.getResponseCode();`
	• < 50: Transaction approved • ≥ 50: Transaction declined • Null: Transaction incomplete. For further details on the response codes that are returned, see the Response Codes document at https://developer.moneris.com.		
ISO	String	2-character numeric	`receipt.getISO();`
	ISO response code		
Bank totals	Object		`receipt.getBankTotals();`
	Response data returned in a Batch Close and Open Totals request. See "Definitions of Response Fields" on the previous page.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Message	String	100-character alpha-numeric	`receipt.getMessage();`
	Response description returned from issuer. The message returned from the issuer is intended for merchant information only, and is **not** intended for customer receipts.		
Authorization code	String	8-character alphanumeric	`receipt.getAuthCode();`
	Authorization code returned from the issuing institution.		
Complete	String	true/false	`receipt.getComplete();`
	Transaction was sent to authorization host and a response was received		
Transaction date	String	Format: yyyy-mm-dd	`receipt.getTransDate();`
	Processing host date stamp		
Transaction time	String	Format: ##:##:##	`receipt.getTransTime();`
	Processing host time stamp		
Ticket	String	N/A	`receipt.getTicket();`
	Reserved field.		
Timed out	String	true/false	`receipt.getTimedOut();`
	Transaction failed due to a process timing out.		
Is Visa Debit	String	true/false	`receipt.getIsVisaDebit();`
	Indicates whether the card processed is a Visa Debit.		
Batch Close/Open Totals response fields			
Processed card types	String Array	N/A	`receipt.getCreditCards(ecr_no);`
	Returns all of the processed card types in the current batch for the terminal ID/ECR Number from the request.		
Terminal IDs	String	8-character alpha-numeric	`receipt.getTerminalIDs();` code to come
	Returns the terminal ID/ECR Number from the request.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Purchase count	String	4-character numeric	`receipt.getPurchaseCount(ecr, cardType);`
	Indicates the # of Purchase, Pre-Authorization Completion and Force Post transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Purchase amount	String	11-character alphanumeric	`receipt.getPurchaseAmount(ecr, cardType);`
	Indicates the dollar amount processed for Purchase, Pre-Authorization Completion or Force Post transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. **EXAMPLE:** +0000000000 = 0.00 and +0000041625 = 416.25		
Refund count	String	4-character numeric	`receipt.getRefundCount(ecr, cardType);`
	Indicates the # of Refund or Independent Refund transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Refund amount	String	11-character alphanumeric	`receipt.getRefundAmount(ecr, cardType);`
	Indicates the dollar amount processed for Refund, Independent Refund or ACH Credit transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. Example, +0000000000 = 0.00 and +0000041625 = 416.25		
Correction count	String	4-character numeric	`receipt.getCorrectionCount(ecr, cardType);`
	Indicates the # of Purchase Correction transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Correction amount	String	11-character alphanumeric	`receipt.getCorrectionAmount(ecr, cardType);`
	Indicates the dollar amount processed for Purchase Correction transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. **EXAMPLE:** +0000000000 = 0.00 and +0000041625 = 416.25		
Recurring Billing Response Fields (see Appendix A, page 1)			

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Recurring billing success	String	true/false	`receipt.getRecurSuccess();`
	Indicates whether the recurring billing transaction has been successfully set up for future billing.		
Recur update success	String	true/false	`receipt.getRecurUpdateSuccess();`
	Indicates recur update success.		
Next recur date	String	yyyy-mm-dd	`receipt.getNextRecurDate();`
	Indicates next recur billing date.		
Recur end date	String	yyyy-mm-dd	`receipt.getRecurEndDate();`
	Indicates final recur billing date.		
Status Check response fields (see)			
Status code	String	3-character alpha-numeric	`receipt.getStatusCode();`
	• < 50: Transaction found and successful • ≥ 50: Transaction not found and not successful **NOTE:** the status code is only populated if the connection object's Status Check property is set to **true**.		
Status message	String	found/not found	`receipt.getStatusMessage();`
	• Found: 0 ≤ Status Code ≤ 49 • Not Found or null: 50 ≤ Status Code ≤ 999. **NOTE:** The status message is only populated if the connection object's Status Check property is set to **true**.		
AVS response fields (see 10.1, page 334)			
AVS result code	String	1-character alpha-numeric	`receipt.getAvsResultCode();`
	Indicates the address verification result. For a full list of possible response codes refer to Section Appendix B.		
CVD response fields (see)			

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
CVD result code	String	2-character alpha-numeric	`receipt.getCvdResultCode();`
	Indicates the CVD validation result. The first byte is the numeric CVD indicator sent in the request; the second byte is the response code. Possible response codes are shown in Appendix B		
MPI response fields (see "MPI" on page 1)			
Type	String	99-character alpha-numeric	
	VERes, PARes or error defines what type of response you are receiving .		
Success	Boolean	true/false	`receipt.getMpiSuccess();`
	True if attempt was successful, false if attempt was unsuccessful.		
Message	String	100-character alphabetic	`receipt.getMpiMessage();`
	MPI TXN transactions can produce the following values: • Y: Create VBV verification form popup window. • N: Send purchase or preauth with crypt type 6 • U: Send purchase or preauth with crypt type 7. MPI ACS transactions can produce the following values: • Y or A: (Also `receipt.getMpiSuccess()=true`) Proceed with cavv purchase or cavv preauth. • N: Authentication failed or high-risk transaction. It is recommended that you do not to proceed with the transaction. Depending on a merchant's risk tolerance and results from other methods of fraud detection, transaction may proceed with crypt type 7. • U or time out: Send purchase or preauth as crypt type 7.		
Term URL	String	255-character alpha-numeric	
	URL to which the PARes is returned		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
MD	String	1024-character alphanumeric	
	Merchant-defined data that was echoed back		
ACS URL	String	255-character alpha-numeric	
	URL that will be for the generated pop-up		
MPI CAVV	String	28-character alpha-numeric	`receipt.getMpiCavv();`
	VbV/MCSC/American Express SafeKey authentication data		
MPI E-Commerce Indicator	String	1-character alpha-numeric	`receipt.getMPIEci();`
CAVV result code	String	1-character alpha-numeric	`receipt.getCavvResultCode();`
	Indicates the Visa CAVV result. For more information, see 8.6.7 Cavv Result Codes for Verified by Visa. • 0 = CAVV authentication results invalid • 1 = CAVV failed validation; authentication • 2 = CAVV passed validation; authentication • 3 = CAVV passed validation; attempt • 4 = CAVV failed validation; attempt • 7 = CAVV failed validation; attempt (US issued cards only) • 8 = CAVV passed validation; attempt (US issued cards only) • The CAVV result code indicates the result of the CAVV validation.		
MPI inline form			`receipt.getMpiInLineForm();`
Vault response fields (see 4.1, page 57)			
Data key	String	28-character alpha-numeric	`receipt.getDataKey();`
	The data key response field is populated when you send a Vault Add Credit Card – ResAddCC (page 59), Vault Encrypted Add Credit Card – EncResAddCC (page 63), Vault Tokenize Credit Card – ResTokenizeCC (page 86), Vault Temporary Token Add – ResTempAdd (page 65) or Vault Add Token – ResAddToken (page 82) transaction. It is the profile identifier that all future financial Vault transactions will use to associate with the saved information.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Vault payment type	String	cc	`receipt.getPaymentType();`
	Indicates the payment type associated with a Vault profile		
Expiring card's Payment type	String	cc	`receipt.getExpPaymentType();`
	Indicates the payment type associated with a Vault profile. Applicable to Vault Get Expiring transaction type.		
Vault masked PAN	String	20-character numeric	`receipt.getResMaskedPan();`
	Returns the first 4 and/or last 4 of the card number saved in the profile.		
Expiring card's Masked PAN	String	20-character numeric	`receipt.getExpMaskedPan();`
	Returns the first 4 and/or last 4 of the card number saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault success	String	true/false	`receipt.getResSuccess();`
	Indicates whether Vault transaction was successful.		
Vault customer ID	String	30-character alpha-numeric	`receipt.getResCustId();`
	Returns the customer ID saved in the profile.		
Expiring card's customer ID	String	30-character alpha-numeric	`receipt.getExpCustId();`
	Returns the customer ID saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault phone number	String	30-character alpha-numeric	`receipt.getResPhone();`
	Returns the phone number saved in the profile.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method / Description
Expiring card's phone number	String	30-character alpha-numeric	`receipt.getExpPhone();`
			Returns the phone number saved in the profile. Applicable to Vault Get Expiring transaction type.
Vault email address	String	30-character alpha-numeric	`receipt.getResEmail();`
			Returns the email address saved in the profile.
Expiring card's email address	String	30-character alpha-numeric	`receipt.getExpEmail();`
			Returns the email address saved in the profile. Applicable to Vault Get Expiring transaction type.
Vault note	String	30-character alpha-numeric	`receipt.getResNote();`
			Returns the note saved in the profile.
Expiring card's note	String	30-character alpha-numeric	`receipt.getExpNote();`
			Returns the note saved in the profile. Applicable to Vault Get Expiring transaction type.
Vault expiry date	String	4-character numeric	`receipt.getResExpdate();`
			Returns the expiry date of the card number saved in the profile. YYMM format.
Expiring card's expiry date	String	4-character numeric	`receipt.getExpExpdate();`
			Returns the expiry date of the card number saved in the profile. YYMM format. Applicable to Vault Get Expiring transaction type.
Vault E-commerce indicator	String	1-character numeric	`receipt.getResCryptType();`
			Returns the e-commerce indicator saved in the profile.
Expiring card's E-commerce indicator	String	1-character numeric	`receipt.getExpCryptType();`
			Returns the e-commerce indicator saved in the profile. Applicable to Vault Get Expiring transaction type.

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Vault AVS street number	String	19-character alpha-numeric	`receipt.getResAvsStreetNumber();`
	Returns the AVS street number saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		
Expiring card's AVS street number	String	19-character alpha-numeric	`receipt.getExpAvsStreetNumber();`
	Returns the AVS street number saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault AVS street name	String	19-character alpha-numeric	`receipt.getResAvsStreetName();`
	Returns the AVS street name saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		
Expiring card's AVS street name	String	19-character alpha-numeric	`receipt.getExpAvsStreetName();`
	Returns the AVS street name saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault AVS ZIP code	String	9-character alpha-numeric	`receipt.getResAvsZipcode();`
	Returns the AVS zip/postal code saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		
Expiring card's AVS ZIP code	String	9-character alpha-numeric	`receipt.getExpAvsZipcode();`
	Returns the AVS zip/postal code saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault credit card number	String	20-character numeric	`receipt.getResPan();`
	Returns the full credit card number saved in the Vault profile. Applicable to Vault Lookup Full transaction only.		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Corporate card	String	true/false	`receipt.getCorporateCard();`
	Indicates whether the card associated with the Vault profile is a corporate card.		
Encrypted Mag Swipe response fields (see 6, page 123)			
Masked credit card number	String	20-character alpha-numeric	`receipt.getMaskedPan();`
Convenience Fee response fields (see Appendix A, page 1)			
Convenience fee success	String	true/false	`receipt.getCfSuccess();`
	Indicates whether the Convenience Fee transaction processed successfully.		
Convenience fee status	String	2-character alpha-numeric	`receipt.getCfStatus();`
	Indicates the status of the merchant and convenience fee transactions. The CfStatus field provides details about the transaction behavior and should be referenced when contacting Moneris Customer Support. Possible values are: • 1 or 1F – Completed 1st purchase transaction • 2 or 2F – Completed 2nd purchase transaction • 3 – Completed void transaction • 4A or 4D – Completed refund transaction • 7 or 7F – Completed merchant independent refund transaction • 8 or 8F – Completed merchant refund transaction • 9 or 9F – Completed 1st void transaction • 10 or 10F – Completed 2nd void transaction • 11A or 11D – Completed refund transaction		
Convenience fee amount	String	9-character decimal	`receipt.getFeeAmount();`
	The expected Convenience Fee amount. This field will return the amount submitted by the merchant for a successful transaction. For an unsuccessful transaction, it will return the expected convenience fee amount		
Convenience fee rate	String	9-character decimal	`receipt.getFeeRate();`
	The convenience fee rate that has been defined on the merchant's profile. For example: 1.00 – a fixed amount or 10.0 - a percentage amount		

Table 58: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Convenience fee type	String	AMT/PCT	`receipt.getFeeType();`
	The type of convenience fee that has been defined on the merchant's profile. Available options are: AMT – fixed amount PCT – percentage		

Table 59: Financial transaction response codes

Code	Description
< 50	Transaction approved
≥ 50	Transaction declined
NULL	Transaction was not sent for authorization

For more details on the response codes that are returned, see the Response Codes document available at https://developer.moneris.com

Table 60: Vault Admin Responses

Code	Description
001	Successfully registered CC details. Successfully updated CC details. Successfully deleted CC details. Successfully located CC details. Successfully located # expiring cards. (NOTE: # = the number of cards located)
983	Cannot find previous
986	Incomplete: timed out
987	Invalid transaction
988	Cannot find expiring cards
Null	Error: Malformed XML

B.1 Definition of Response Fields – MCP

MCP response fields

Variable Name	Description	Get Method
MCP rate	The foreign exchange rate (foreign currency to CAD) that will be used for the transaction If a **MCP rate token** was used, it will reflect the rate secured by the MCP Get Rate transaction; if no token was used, the rate is the current exchange rate retrieved by the Moneris Gateway	`receipt.getMCPRate();`
merchant settlement currency	Currency that the merchant is settling in	`receipt.getMerchantSettlementCurrency();`
merchant settlement amount	Amount that will be paid to the merchant, in Canadian dollars	`receipt.getMerchantSettlementAmount();`
cardholder currency code	ISO code for the foreign currency the cardholder is using to pay	`receipt.getCardholderCurrencyCode();`
cardholder amount	Amount, in units of foreign currency, the cardholder will pay on the transaction	`receipt.getCardholderAmount();`
MCP error status code	A number representing a MCP error code response	`receipt.getMCPErrorStatusCode();`
MCP error message	Message corresponding with an	`receipt.getMCPErrorMessage();`

Variable Name	Description	Get Method
	MCP error code	
host ID	Unique identifier used across the Moneris platform	`receipt.getHostId();`

Response fields specific to MCP Get Rate

Variable Name	Description	Get Method
rate transaction type	Reflects the transaction type being sent in the request	`receipt.getRateTxnType();`
MCP rate token	Time-limited token representing a temporarily locked in foreign exchange rate for use in financial transactions This field is returned in the response to a MCP Get Rate request	`receipt.GetMCPRateToken();`
rate inquiry start time	The local time (ISO 8601) when the rate is requested	`receipt.getRateInqStartTime();`
rate inquiry end time	The local time (ISO 8601) when the rate is returned	`receipt.getRateInqEndTime();`
rate validity start time	The time (unix UTC) of when the rate is valid from	`receipt.getRateValidityStartTime());`
rate validity end time	The time (unix UTC) of when the rate is valid until	`receipt.getRateValidityEndTime();`
rate validity period	The time in minutes this rate is valid for	`receipt.getRateValidityPeriod();`

Appendix C Error Messages

Error messages that are returned if the gateway is unreachable

Global Error Receipt

You are not connecting to our servers. This can be caused by a firewall or your internet connection.

Response Code = NULL

The response code can be returned as null for a variety of reasons. The majority of the time, the explanation is contained within the Message field.

When a 'NULL' response is returned, it can indicate that the issuer, the credit card host, or the gateway is unavailable. This may be because they are offline or because you are unable to connect to the internet.

A 'NULL' can also be returned when a transaction message is improperly formatted.

Error messages that are returned in the Message field of the response

XML Parse Error in Request: <System specific detail>
An improper XML document was sent from the API to the servlet.

XML Parse Error in Response: <System specific detail>
An improper XML document was sent back from the servlet.

Transaction Not Completed Timed Out
Transaction timed out before the host responds to the gateway.

Request was not allowed at this time
The host is disconnected.

Could not establish connection with the gateway: <System specific detail>
Gateway is not accepting transactions or server does not have proper access to internet.

Input/Output Error: <System specific detail>
Servlet is not running.

The transaction was not sent to the host because of a duplicate order id
Tried to use an order id which was already in use.

The transaction was not sent to the host because of a duplicate order id
Expiry Date was sent in the wrong format.

Vault error messages

Can not find previous
Data key provided was not found in our records or profile is no longer active.

Invalid Transaction
Transaction cannot be performed because improper data was sent.

or

Mandatory field is missing or an invalid SEC code was sent.

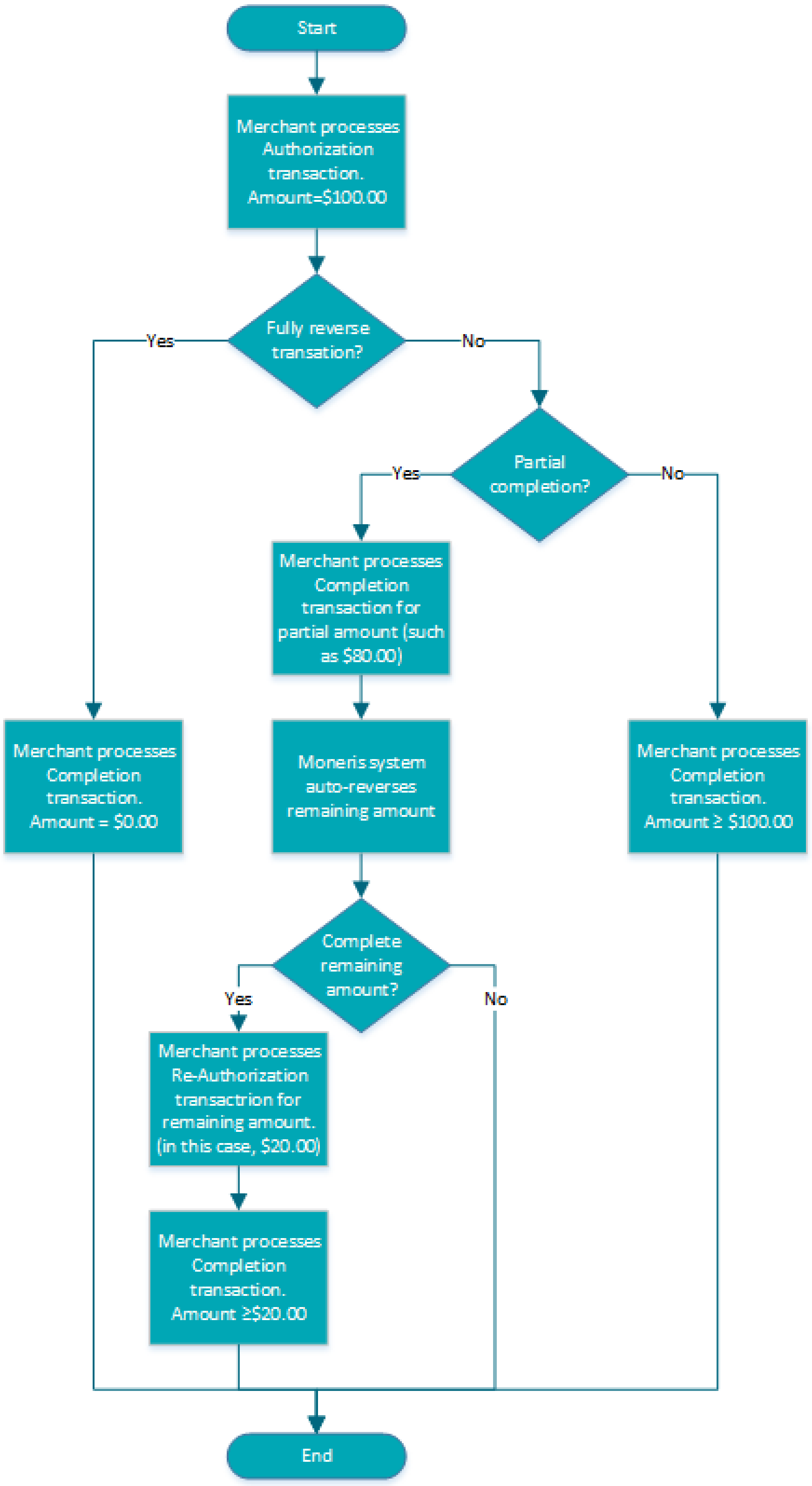
Malformed XML
Parse error.

Incomplete
Timed out.

or

Cannot find expiring cards.

Appendix D Process Flow for Basic Pre-Auth, Re-Auth and Completion Transactions


Start
Merchant processes Authorization transaction. Amount=$100.00
Fully reverse transation?
Yes
No
Partial completion?
Yes
No
Merchant processes Completion transaction for partial amount (such as $80.00)
Merchant processes Completion transaction. Amount = $0.00
Moneris system auto-reverses remaining amount
Merchant processes Completion transaction. Amount ≥ $100.00
Complete remaining amount?
Yes
No
Merchant processes Re-Authorization transactrion for remaining amount. (in this case, $20.00)
Merchant processes Completion transaction. Amount ≥$20.00
End

Appendix E Merchant Checklists for INTERAC® Online Payment Certification Testing

Merchant Information

Name and URL	Merchant Name (English)	
	Homepage URL (English)	
	Merchant Name (French)	
	Homepage URL (French)	
Number	Merchant Number	
Transaction fee cat-egory (Circle one)	Government Education General	

Checklist for Front-End Tests

Case #	Date Completed	Remarks
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

Case #	Date Completed	Remarks
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

Merchant Requirements

Table 61: Checklist for web display requirements

Done	Requirement
Checkout page	

Table 61: Checklist for web display requirements (continued)

Done	Requirement
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both
Design and Wordmark Requirements (any page)	
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*^{MD}>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: • ® Trademark of Interac Inc. Used under licence" • ^{MD} Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides ability to print

Table 61: Checklist for web display requirements (continued)

Done	Requirement
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 62: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce
Provided screenshots	
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix F Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing

Third-Party Service Provider Information

Name	English	
	French	
Merchant Web Application	Solution Name	
	Version	
Acquirer		

Interaconline.com/Interacenlgne.com Web Site Listing Information

See http://www.interaconline.com/merchants_thirdparty.php for examples.

English contact inform-ation	5 lines maximum. 35 characters/line maximum. For example, contact name and title, department, telephone, web site, email.
English logo	File type: PNG. Maximum size: 120x120 pixels.
French contact inform-ation	5 lines maximum. 35 characters/line maximum. For example, contact name and title, department, telephone, web site, email.
French logo	File type: PNG. Maximum size: 120x120 pixels.

Table 63: Checklist for front-end tests

Case #	Date Completed	Remarks
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		

Table 63: Checklist for front-end tests

Case #	Date Completed	Remarks
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

Merchant Requirements

Table 64: Checklist for web display requirements

Done	Requirement
Checkout page	
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both
Design and Wordmark Requirements (any page)	
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.

Table 64: Checklist for web display requirements (continued)

Done	Requirement
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*^{MD}>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: • ® Trademark of Interac Inc. Used under licence" • ^{MD} Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides the ability to print
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 65: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce

Table 66: Checklist for required screenshots

Done	Requirement
Provided screenshots	
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix G Merchant Checklists for INTERAC® Online Payment Certification

Merchant Information

Name and URL	Merchant Name (English)	
	Homepage URL (English)	
	Merchant Name (French)	
	Homepage URL (French)	
Number	Merchant Number	
Transaction fee cat-egory (Circle one)	Government Education General	
Third-party service provider	Company name	
Service provider's merchant web application	Solution name	
	Version	

Merchant Requirements

Table 67: Checklist for web display requirements

Done	Requirement
Checkout page	
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both
Design and Wordmark Requirements (any page)	
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.

Table 67: Checklist for web display requirements (continued)

Done	Requirement
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*^{MD}>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: ◦ ® Trademark of Interac Inc. Used under licence" ◦ ^{MD} Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides ability to print
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 68: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce
Provided screenshots	
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix H INTERAC® Online Payment Certification Test Case Detail

H.1 Common Validations

The Merchant sends a request to the INTERAC Online Merchant Test Tool, which validates the fields as follows:

- All mandatory fields are present.
- All fields are valid according to their definition in the *INTERAC Online Functional Specifications* (including field lengths, valid characters and so on).
- Merchant number is that of a valid registered merchant.
- Funded URL matches one of the merchant's registered funded URLs that were provided during merchant registration.
- The not funded URL matches one of the merchant's registered Not Funded URLs that were provided during merchant registration.
- No additional fields are present.

H.2 Test Cases

Table 69: Cases 1-3

Objective	To test that the merchant can do all of the following: • Send a valid request to the Gateway page • Receive a valid confirmation of funding from the Issuer Online Banking application • Issue a request for purchase completion to the acquirer • Receive an approved response from the acquirer.
Pre-requisites	None
Configuration	Merchant sends form posts to the Merchant Test Tool, which in turn responds to either the Funded or Not Funded URL. The Merchant is connected to an acquirer emulator, which can be set to confirm any request for payment confirmation. (That is, the back-end process of sending a 0200 Message to the issuer is emulated to always accept the purchase request).
Special tools required	None

Table 69: Cases 1-3 (continued)

Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 3, the format of the amount must be ### ### #03.##.
Expected out-come	The merchant indicates to the customer that the purchase was completed and presents a confirmation screen that includes (depending on the test case) the correct amount, the issuer name and the issuer confirmation number. Test case 1 • Issuer name: 123Bank • Issuer confirmation number: CONF#123 Test case 2 • Issuer name: Bank Éàêëï#$.,-/=?@' • Issuer confirmation number: #$.,-/=?@'UPdn9 Test case 3 • Issuer name: B • Issuer confirmation number: C
Applicable logs	• Merchant Test Tool logs • Screen capture of the merchant's confirmation page.

Table 70: Case 4

Objective	To test that the merchant handles a rejection in response to the acquirer
Pre-requisites	None
Configuration	Same as test cases 1-3 except that the acquirer emulator must be set to decline the request for mayment confirmation. (That is, to emulate the scenario in which an issuer sends a delcine in the 0210 response to the acquirer's 0200 message.)

Table 70: Case 4 (continued)

Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant for any amount where the two least significant dollar digits are 04. (That is, of the form ### ### #04.##.)
Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 71: Cases 5-22

Objective	To test that a merchant safely handles redirections to the Funded URL with invalid data, and treats the transaction as funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.
Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 13, the format of the amount must be ### ### #13.##.

Table 71: Cases 5-22 (continued)

Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 72: Case 23

Objective	To test that a merchant can receive a valid redirection from the issuer that indicates the payment was not funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.
Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data is provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant for any amount where the two least significant dollar digits are 23. (That is, of the form ### ### #23.##.)
Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 73: Cases 24-39

Objective	To test that a merchant safely handles redirections to the Not Funded URL with invalid data, and treats the transaction as not funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.

Table 73: Cases 24-39 (continued)

Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data is provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 27, the format of the amount must be ### ### #27.##.
Expected out-come	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

H.3 Merchant front-end test case values

These values are automatically sent by the INTERAC Online Merchant Test Tool. They are provided here for reference only.

Table 74: Test cases 1 and 4—Funded URL

Redirection URL	Funded
ISSLANG	en
TRACK2	3728024906540591206=12010123456789XYZ
ISSCONF	CONF#123
ISSNAME	123Bank
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 75: Test case 2—Funded URL

Redirection URL	Funded
ISSLANG	en

Table 75: Test case 2—Funded URL

TRACK2	5268051119993326=29129999999999999000
ISSCONF	#$.,-/=?@'UPdn9
ISSNAME	987Bank Éàêëï#$.,-/=?@'Àôùûüÿç
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 76: Test case 3—Funded URL

Redirection URL	Funded
ISSLANG	fr
TRACK2	453781122255=1001ABC11223344550000000
ISSCONF	C
ISSNAME	B
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	123

Table 77: Test cases 5-22—invalid fields, Funded URL

Test case	Purpose	Field	Value
5	missing field	IDEBIT_INVOICE	(missing)
6	missing field	IDEBIT_MERCHDATA	(missing)
7	missing field	IDEBIT_ISSLANG	(missing)
8	missing field	IDEBIT_TRACK2	(missing)
9	missing field	IDEBIT_ISSCONF	(missing)
10	missing field	IDEBIT_ISSNAME	(missing)
11	missing field	IDEBIT_VERSION	(missing)
12	missing field	IDEBIT_TRACK2, IDEBIT_ ISSCONF, IDEBIT_ISSNAME	(missing)
13	wrong value	IDEBIT_INVOICE	XXX
14	wrong value	IDEBIT_MERCHDATA	XXX

Table 77: Test cases 5-22—invalid fields, Funded URL (continued)

Test case	Purpose	Field	Value
15	invalid value	IDEBIT_ISSLANG	de
16	value too long	IDEBIT_TRACK2	3728024906540591206=12010123456789XYZA
17	invalid check digit	IDEBIT_TRACK2	3728024906540591207=12010123456789XYZ
18	field too long	IDEBIT_ISSCONF	Too long confirm
19	invalid character	IDEBIT_ISSCONF	CONF<123
20	field too long	IDEBIT_ISSNAME	Very, very, very long issuer name
21	invalid character	IDEBIT_ISSNAME	123<Bank
22	invalid value	IDEBIT_VERSION	2

Table 78: Test case 23—valid data, Not Funded URL

Redirection URL	Not funded
ISSLANG	en
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 79: Test cases 5-22—invalid fields, Funded URL

Test case	Purpose	Field	Value
24	missing field	IDEBIT_INVOICE	(missing)
25	missing field	IDEBIT_MERCHDATA	(missing)
26	missing field	IDEBIT_ISSLANG	(missing)
27	IDEBIT_TRACK2 is present and valid	IDEBIT_TRACK2	3728024906540591206=12010123456789XYZ
28	IDEBIT_ISSCONF is present and valid	IDEBIT_ISSCONF	CONF#123
29	IDEBIT_ISSNAME is present and valid	IDEBIT_ISSNAME	12Bank
30	missing field	IDEBIT_VERSION	(missing)

Table 79: Test cases 5-22—invalid fields, Funded URL (continued)

Test case	Purpose	Field	Value
31	wrong value	IDEBIT_INVOICE	XXX
32	invalid value	IDEBIT_INVOICE	invalid </html> tricky data
33	wrong value	IDEBIT_MERCHDATA	XXX
34	invalid value	IDEBIT_MERCHDATA	<2000 characters in the range hex 20-7E
35	invalid value	IDEBIT_ISSLANG	de
36	invalid IDEBIT_ TRACK2 is present	IDEBIT_TRACK2	INVALIDTRACK2, incorrect format and too long
37	invalid IDEBIT_ ISSCONF is present	IDEBIT_ISSCONF	Too long confirm
38	invalid IDEBIT_ ISSNAME is present	IDEBIT_ISSNAME	Very, very, very long issuer name
39	invalid value	IDEBIT_VERSION	2