Moneris™

BE PAYMENT READY

Moneris Gateway API - Integration Guide – PHP

Version: 1.2.6

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 v1.2.6

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

Previous version changes

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

Security and Compliance 2
Confidentiality 2

Changes in v1.2.6 3

1 About This Documentation 10
1.1 Purpose 10
Getting Help 10
1.2 Who Is This Guide For? 11
1.3 Adding cURL CA Root Certificate to PHP API 11

2 Basic Transaction Set 12
2.1 Purchase 13
2.2 Pre-Authorization 18
2.3 Pre-Authorization Completion 23
2.4 Re-Authorization 26
2.5 Force Post 28
2.6 Purchase Correction 31
2.7 Refund 33
2.8 Independent Refund 35
2.9 Card Verification with AVS and CVD 37
2.10 Batch Close 41
2.11 Open Totals 43

3 Credential on File 45
3.1 About Credential on File 45
3.2 Credential on File Info Object and Variables 45
3.3 Credential on File Transaction Types 46
3.4 Initial Transactions in Credential on File 46
3.5 Vault Tokenize Credit Card and Credential on File 47
3.6 Credential on File and Converting Temporary Tokens 47
3.7 Card Verification and Credential on File Transactions 47
3.7.1 When to Use Card Verification With COF 47
3.7.2 Credential on File and Vault Add Token 48
3.7.3 Credential on File and Vault Update Credit Card 48
3.7.4 Credential on File and Vault Add Credit Card 48
3.7.5 Credential on File and Recurring Billing 49

4 Vault 50
4.1 About the Vault Transaction Set 50
4.2 Vault Transaction Types 50
4.2.1 Administrative Vault Transaction types 50
4.2.2 Financial Vault Transaction types 52
4.3 Vault Administrative Transactions 52
4.3.1 Vault Add Credit Card – ResAddCC 52
4.3.1.1 Vault Data Key 56
4.3.1.2 Vault Encrypted Add Credit Card – EncResAddCC 56
4.3.2 Vault Temporary Token Add – ResTempAdd 59
4.3.3 Vault Update Credit Card – ResUpdateCC 61
4.3.3.1 Vault Encrypted Update CC - EncResUpdateCC 65
4.3.4 Vault Delete - ResDelete 68
4.3.5 Vault Lookup Full - ResLookupFull 69
4.3.6 Vault Lookup Masked – ResLookupMasked 71
4.3.7 Vault Get Expiring – ResGetExpiring 73

4.3.8 Vault Is Corporate Card - ResIscorporateCard 74
4.3.9 Vault Add Token – ResAddToken 76
4.3.10 Vault Tokenize Credit Card – ResTokenizeCC 79
4.4 Vault Financial Transactions 83
4.4.1 Customer ID Changes 84
4.4.2 Purchase with Vault – ResPurchaseCC 84
4.4.3 Pre-Authorization with Vault – ResPreauthCC 89
4.4.4 Vault Independent Refund CC - ResIndRefundCC 93
4.4.5 Force Post with Vault - ResForcePostCC 96
4.4.6 Card Verification with Vault – ResCardVerificationCC 98
4.5 Hosted Tokenization 102

5 INTERAC® Online Payment 103
5.1 About INTERAC® Online Payment Transactions 103
5.2 Other Documents and References 103
5.3 Website and Certification Requirements 104
5.3.1 Things to provide to Moneris 104
5.3.2 Certification process 104
5.3.3 Client Requirements 105
5.3.4 Delays 105
5.4 Transaction Flow for INTERAC® Online Payment 106
5.5 Sending an INTERAC® Online Payment Purchase Transaction 107
5.5.1 Fund-Guarantee Request 107
5.5.2 Online Banking Response and Fund-Confirmation Request 108
5.6 INTERAC® Online Payment Purchase 108
5.7 INTERAC® Online Payment Refund 110
5.8 INTERAC® Online Payment Field Definitions 112

6 Mag Swipe Transaction Set 116
6.1 Mag Swipe Transaction Type Definitions 116
6.1.1 Encrypted Mag Swipe Transactions 117
6.2 Mag Swipe Purchase 117
6.2.1 Encrypted Mag Swipe Purchase 120
6.3 Mag Swipe Pre-Authorization 122
6.3.1 Encrypted Mag Swipe Pre-Authorization 124
6.4 Mag Swipe Completion 126
6.5 Mag Swipe Force Post 128
6.5.1 Encrypted Mag Swipe Force Post 131
6.6 Mag Swipe Purchase Correction 133
6.7 Mag Swipe Refund 135
6.8 Mag Swipe Independent Refund 137
6.8.1 Encrypted Mag Swipe Independent Refund 139

7 Level 2/3 Transactions 143
7.1 About Level 2/3 Transactions 143
7.2 Level 2/3 Visa Transactions 143
7.2.1 Level 2/3 Transaction Types for Visa 143
7.2.2 Level 2/3 Transaction Flow for Visa 146
7.2.3 VS Completion 148
7.2.4 VS Purchase Correction 152
7.2.5 VS Force Post 154
7.2.6 VS Refund 159
7.2.7 VS Independent Refund 163
7.2.8 VS Corpais 168
7.2.8.1 VS Purcha - Corporate Card Common Data 169
7.2.8.2 VS Purchl - Line Item Details 174
7.2.8.3 Sample Code for VS Corpais 177

7.3 Level 2/3 MasterCard Transactions 179
7.3.1 Level 2/3 Transaction Types for MasterCard 179
7.3.2 Level 2/3 Transaction Flow for MasterCard 181
7.3.3 MC Completion 183
7.3.4 MC Force Post 184
7.3.5 MC Purchase Correction 187
7.3.6 MC Refund 188
7.3.7 MC Independent Refund 190
7.3.8 MC Corpais - Corporate Card Common Data with Line Item Details 192
7.3.8.1 MC Corpac - Corporate Card Common Data 194
7.3.8.2 MC Corpal - Line Item Details 201
7.3.8.3 Tax Array Object - MC Corpais 204
7.3.8.4 Sample Code for MC Corpais 206
7.4 Level 2/3 American Express Transactions 209
7.4.1 Level 2/3 Transaction Types for Amex 209
7.4.2 Level 2/3 Transaction Flow for Amex 211
7.4.3 Level 2/3 Data Objects in Amex 212
7.4.3.1 About the Level 2/3 Data Objects for Amex 212
7.4.3.2 Defining the AxLevel23 Object 212
Table 1 Object 213
Table 1 - Setting the N1Loop Object 214

Table 1 - Setting the AxRef Object 216

Table 2 Object 217
Table 2 - Setting the AxIt1Loop Object 218

Table 2 - Setting the AxIt106s Object 220

Table 2 - Setting the AxTxi Object 221

Table 3 Object 225
Table 3 - Setting the AxTxi Object 226

7.4.4 AX Completion 229
7.4.5 AX Force Post 232
7.4.6 AX Purchase Correction 235
7.4.7 AX Refund 237
7.4.8 AX Independent Refund 240

8 MPI 245
8.1 About MPI Transactions 245
8.2 3-D Secure Implementations (VbV, MCSC, SafeKey) 245
8.3 Activating 3-D Secure Functionality 246
8.4 Activating Amex SafeKey 246
8.5 Transaction Flow for MPI 246
8.6 MPI Transactions 247
8.6.1 VbV, MCSC and SafeKey Responses 248
8.6.2 MpiTxn Request Transaction 250
8.6.2.1 TXN Response and Creating the Popup 252
8.6.3 Vault MPI Transaction – ResMpiTxn 252
8.6.4 MPI ACS Request 254
8.6.4.1 ACS Response and Forming a Transaction 256
8.6.5 Purchase with 3-D Secure – cavv_purchase 256
8.6.5.1 Purchase with 3-D Secure and Recurring Billing 265
8.6.6 Pre-Authorization with 3-D Secure – cavvPreauth 266
8.6.7 Cavv Result Codes for Verified by Visa 273
8.6.8 Vault Cavv Purchase 274
8.6.9 Vault Cavv Pre-Authorization 276

9 e-Fraud Tools 279
9.1 Address Verification Service 281
9.1.1 About Address Verification Service (AVS) 281
9.1.2 AVS Info Object 281
9.1.3 AVS Response Codes 283
9.1.4 AVS Sample Code 285
9.2 Card Validation Digits (CVD) 287
9.2.1 About Card Validation Digits (CVD) 287
9.2.2 Transactions Where CVD Is Required 287
9.2.3 CVD Information Object 288
9.2.4 CVD Result Codes 290
9.2.5 Sample Purchase with CVD Info Object 290
9.3 Transaction Risk Management Tool 291
9.3.1 About the Transaction Risk Management Tool 291
9.3.2 Introduction to Queries 291
9.3.3 Session Query 292
9.3.3.1 Session Query Transaction Flow 298
9.3.4 Attribute Query 299
9.3.4.1 Attribute Query Transaction Flow 303
9.3.5 Handling Response Information 303
9.3.5.1 TRMT Response Fields 304
9.3.5.2 Understanding the Risk Score 307
9.3.5.3 Understanding the Rule Codes, Rule Names and Rule Messages 308
9.3.5.4 Examples of Risk Response 315
Session Query 315
Attribute Query 316
9.3.6 Inserting the Profiling Tags Into Your Website 316
9.4 Encorporating All Available Fraud Tools 318
9.4.1 Implementation Options for TRMT 318
9.4.2 Implementation Checklist 318
9.4.3 Making a Decision 320

10 Apple Pay In-App and on the Web Integration 321
10.1 About Apple Pay In-App and on the Web Integration 321
10.2 About API Integration of Apple Pay 321
10.2.1 Transaction Types That Use Apple Pay 321
10.3 Apple Pay In-App Process Flows 322
10.4 Cavv Purchase – Apple Pay 323
10.5 Cavv Pre-Authorization – Apple Pay 327

11 Offlinx™ 332
11.1 What Is a Pixel Tag? 332
11.2 Offlinx™ and API Transactions 332

12 Convenience Fee 333
12.1 About Convenience Fee 333
12.2 Purchase with Convenience Fee 333
12.3 Convenience Fee Purchase w/ Customer Information 336
12.4 Convenience Fee Purchase with VbV, MCSC and Amex SafeKey 340

13 Recurring Billing 344
13.1 About Recurring Billing 344
13.2 Purchase with Recurring Billing 344
13.3 Recurring Billing Update 347
13.4 Recurring Billing Response Fields and Codes 351
13.5 Credential on File and Recurring Billing 352

14 Customer Information 354

14.1 Using the Customer Information Object 354
14.1.1 Customer Info Object – Miscellaneous Properties 355
14.1.2 Customer Info Object – Billing/Shipping Information 355
14.1.2.1 Set Methods for Billing and Shipping Info 356
14.1.2.2 Using Hash Tables for Billing and Shipping Info 357
14.1.3 Customer Info Object – Item Information 358
14.1.3.1 Set Methods for Item Information 358
14.1.3.2 Using Hash Tables for Item Information 358
14.2 Customer Information Sample Code 359
15 Status Check 362
15.1 About Status Check 362
15.2 Using Status Check Response Fields 362
15.3 Sample Purchase with Status Check 363
16 Visa Checkout 364
16.1 About Visa Checkout 364
16.2 Transaction Types - Visa Checkout 364
16.3 Integrating Visa Checkout Lightbox 365
16.4 Transaction Flow for Visa Checkout 366
16.5 Visa Checkout Purchase 367
16.6 Visa Checkout Pre-Authorization 368
16.7 Visa Checkout Completion 370
16.8 Visa Checkout Purchase Correction 372
16.9 Visa Checkout Refund 374
16.10 Visa Checkout Information 376
17 Testing a Solution 379
17.1 About the Merchant Resource Center 379
17.2 Logging In to the QA Merchant Resource Center 379
17.3 Test Credentials for Merchant Resource Center 379
17.4 Getting a Unique Test Store ID and API Token 381
17.5 Processing a Transaction 383
17.5.1 Overview 383
17.5.2 HttpsPostRequest Object 384
17.5.3 Receipt Object 386
17.6 Testing INTERAC® Online Payment Solutions 386
17.7 Testing MPI Solutions 387
17.8 Testing Visa Checkout 389
17.8.1 Creating a Visa Checkout Configuration for Testing 389
17.9 Test Card Numbers 389
17.9.1 Test Card Numbers for Level 2/3 390
17.9.2 Test Cards for Visa Checkout 390
17.10 Simulator Host 390
18 Moving to Production 393
18.1 Activating a Production Store Account 393
18.2 Configuring a Store for Production 393
18.2.1 Configuring an INTERAC® Online Payment Store for Production 394
18.2.1.1 Completing the Certification Registration - Merchants 394
18.2.1.2 Third-Party Service/Shopping Cart Provider 395
18.3 Receipt Requirements 396
18.3.1 Certification Requirements 396
Appendix A Definitions of Request Fields 397
A.1 Definitions of Request Fields – Credential on File 408
A.2 Definition of Request Fields – Recurring 409
A.3 Definition of Request Fields for Level 2/3 - Visa 410

A.4 Definition of Request Fields for Level 2/3 - MasterCard 419
A.5 Definition of Request Fields for Level 2/3 - Amex 429
A.6 Definition of Request Fields – Offlinx™ 439

Appendix B Definitions of Response Fields 441

Appendix C Error Messages 455

Appendix D Process Flow for Basic Pre-Auth, Re-Auth and Completion Transactions 457

Appendix E Merchant Checklists for INTERAC® Online Payment Certification Testing 458

Appendix F Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing 462

Appendix G Merchant Checklists for INTERAC® Online Payment Certification 467

Appendix H INTERAC® Online Payment Certification Test Case Detail 470
H.1 Common Validations 470
H.2 Test Cases 470
H.3 Merchant front-end test case values 474

Copyright Notice 479

Trademarks 479

1 About This Documentation

1.1 Purpose

This document describes the transaction information for using the PHP 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 PHP programming language.

System Requirements

- PHP or above
- Port 443 open for bi-directional communication
- Web server with a SSL certificate
- cURL - PHP interface – see Adding cURL CA Root Certificate to PHP API

1.3 Adding cURL CA Root Certificate to PHP API

cURL CA Root Certificate File:

The default installation of PHP/cURL does not include the cURL CA root certificate file. In order for the Moneris Gateway PHP API to connect to the Moneris Gateway during transaction processing, the ‘mpgclasses.php’ file that’s included with the PHP API package needs to be modified to include a path to the CA root certificate file.

To add the cURL CA root certificate file to the PHP API package, do the following:

1. If cURL was not installed separately from your PHP installation, libcurl is included in your PHP installation. You need to download the ‘cacert.pem’ file from http://curl.haxx.se/docs/caextract.html and save it to the necessary directory.
2. Once downloaded, rename the file to ‘curl-ca-bundle.crt’ (e.g., 'C:\path\to\curl-ca-bundle.crt'). If cURL was installed separately from PHP, you may need to determine the path to the cURL CA root certificate bundle on your system (e.g., 'C:\path\to\curl-ca-bundle.crt').
3. Insert the code below into the ‘mpgclasses.php’ file as part of the cURL option setting, at approximately line 73 below the line ‘`curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, TRUE);`‘

   ```
   curl_setopt($ch, CURLOPT_CAINFO, 'C:\path\to\curl-ca-bundle.crt');
   ```

   For more information regarding the `CURLOPT_SSL_VERIFYPEER` option, please refer to your PHP documentation.

2 Basic Transaction Set

- 2.1 Purchase
- 2.2 Pre-Authorization
- 2.3 Pre-Authorization Completion
- 2.4 Re-Authorization
- 2.5 Force Post
- 2.6 Purchase Correction
- 2.7 Refund
- 2.8 Independent Refund
- 2.9 Card Verification with AVS and CVD
- 2.10 Batch Close
- 2.11 Open Totals

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

```
$txnArray = array(‘type’=>’purchase', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Purchase transaction values

Table 1: Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID **order_id**	String	50-character alphanumeric	'order_id'=>$order_id
Amount	String	(missing or bad snippet)	'amount'=>$amount
Credit card number	String	20-character alphanumeric	'pan'=>$pan
Expiry date	String	4-character alphanumeric (YYMM format)	'expdate'=>$expiry_date
E-commerce indicator	String	1-character alphanumeric	'crypt_type'=>$crypt

Table 2: Purchase transaction object optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Card Match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String*	50-character alpha-numeric	`'cm_id' => $transaction_id`
Customer information	Object	N/A	`$mpgTxn->setCustInfo ($mpgCustInfo);`
AVS	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
Convenience fee **NOTE:** This variable does not apply to Credential on File transactions.	Object	N/A	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`
Recurring billing	Object	N/A	`$mpgTxn->setRecur($mpgRecur);`
Dynamic descriptor	String	20-character alpha-	`'dynamic_`

Table 2: Purchase transaction object optional values

Value	Type	Limits	Set method
		numeric	`descriptor'=>$dynamic_descriptor`
Wallet indicator[1] **NOTE:** For basic Purchase and Preauthorization, the wallet indicator applies to Visa Checkout and MasterCard MasterPass only. For more, see Appendix A Definitions of Request Fields	String	3-character alphanumeric	`'wallet_indicator'=>$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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

[1] Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Purchase

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='purchase';
$cust_id='cust id';
```

Sample Purchase

```
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='4242424242424242';
$expiry_date='2011';
$crypt='7';
$dynamic_descriptor='123';
$status_check = 'false';
//Optional - Set for Multi-Currency only
//$amount must be 0.00 when using multi-currency
$mcp_amount = '500'; //penny value amount 1.25 = 125
$mcp_currency_code = '840'; //ISO-4217 country currency number
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt,
'dynamic_descriptor'=>$dynamic_descriptor
//,'wallet_indicator' => '' //Refer to documentation for details
//,'mcp_amount' => $mcp_amount
//,'mcp_currency_code' => $mcp_currency_code
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************** Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
/* Status Check Example
$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
*/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
```

Sample Purchase

```
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    print("\nStatusCode = " . $mpgResponse->getStatusCode());
    print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    print("\nMCPAmount = " . $mpgResponse->getMCPAmount());
    print("\nMCPCurrenyCode = " . $mpgResponse->getMCPCurrencyCode());
    print("\nHostId = " . $mpgResponse->getHostId());
    print("\nIssuerId = " . $mpgResponse->getIssuerId());
    ?>
```

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 457

Pre-Authorization transaction object definition

```
$txnArray = array(‘type’=>’preauth', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Pre-Authorization transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Pre-Authorization transaction values

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

Table 3: Pre-Authorization object required values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric	`'expdate'=>$expiry_date`
E-Commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 4: Pre-Authorization object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Card Match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String*	50-character alpha-numeric	`'cm_id' => $transaction_id`
Customer information	Object	N/A	`$mpgTxn->setCustInfo($mpgCustInfo);`
AVS	Object	N/A	`$mpgTxn->setAvsInfo($mpgAvsInfo);`
CVD **NOTE:** When storing credentials on the ini-	Object	N/A	`$mpgTxn->setCvdInfo($mpgCvdInfo);`

Value	Type	Limits	Set method
tial 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**.			
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Wallet indicator[1] **NOTE:** For basic Purchase and Preauthorization, the wallet indicator applies to Visa Checkout and MasterCard MasterPass only. For more, see Appendix A Definitions of Request Fields	String	3-character alphanumeric	`'wallet_indicator'=>$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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

[1] Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Pre-Authorization

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestPreAuth.php store1 yesguy
##
require "../../mpgClasses.php";
$store_id='store5';
```

Sample Pre-Authorization

```
$api_token="yesguy";
/************************* Transactional Variables ****************************/
$type='preauth';
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='4242424242424242';
$expiry_date='2011';
$crypt='7';
$dynamic_descriptor='123';
$status_check = 'false';
//Optional - Set for Multi-Currency only
//$amount must be 0.00 when using multi-currency
$mcp_amount = '500'; //penny value amount 1.25 = 125
$mcp_currency_code = '840'; //ISO-4217 country currency number
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt,
'dynamic_descriptor'=>$dynamic_descriptor
//,'wallet_indicator' => '' //Refer to documentation for details
//,'mcp_amount' => $mcp_amount,
//'mcp_currency_code' => $mcp_currency_code
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
$mpgTxn = new mpgTransaction($txnArray);
/****************** Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
print("\nMCPAmount = " . $mpgResponse->getMCPAmount());
print("\nMCPCurrenyCode = " . $mpgResponse->getMCPCurrencyCode());
?>
```

2.3 Pre-Authorization Completion

Retrieves funds that have been locked (by either a Pre-Authorization or a 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

Completion transaction object

```
$txnArray = array(‘type’=>’completion', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Completion transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Completion transaction values

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

Table 5: Completion transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Completion Amount	String	(missing or bad snippet)	`'comp_amount'=>$comp_amount`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
E-Commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 6: Completion transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Shipping indicator[1]	String	1-character alpha-numeric	`‘ship_indicator’=>$ship_indicator`

Sample Basic Pre-Authorization Completion

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$orderid='ord-150816-11:55:18';
$txnnumber='117735-0_10';
$compamount='1.00';
$dynamic_descriptor='123';
//Optional - Set for Multi-Currency only
//$compamount must be 0.00 when using multi-currency
$mcp_amount = '200'; //penny value amount 1.25 = 125
$mcp_currency_code = '840'; //ISO-4217 country currency number
$ship_indicator = "F"; //optional
## step 1) create transaction array ###
$txnArray=array('type'=>'completion',
'txn_number'=>$txnnumber,
'order_id'=>$orderid,
'comp_amount'=>$compamount,
'crypt_type'=>'7',
'cust_id'=>'customer ID',
//'mcp_amount' => $mcp_amount,
//'mcp_currency_code' => $mcp_currency_code
//'ship_indicator'=>$ship_indicator, //optional
'dynamic_descriptor'=>$dynamic_descriptor
);
## step 2) create a transaction object passing the hash created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
```

[1] Available to Canadian integrations only.

Sample Basic Pre-Authorization Completion

```
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nMCPAmount = " . $mpgResponse->getMCPAmount());
print("\nMCPCurrenyCode = " . $mpgResponse->getMCPCurrencyCode());
?>
$compamount='0.10';
$dynamic_descriptor='123';
## step 1) create transaction array ###
$txnArray=array('type'=>'completion',
'txn_number'=>$txnnumber,
'order_id'=>$orderid,
'comp_amount'=>$compamount,
'crypt_type'=>'7',
'cust_id'=>'customer ID',
'dynamic_descriptor'=>$dynamic_descriptor
);
## step 2) create a transaction object passing the hash created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
```

Sample Basic Pre-Authorization Completion

```
    print("\nMessage = " . $mpgResponse->getMessage());
    print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    ?>
```

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

```
$txnArray = array(‘type’=>’reauth', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Re-Authorization transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Re-Authorization transaction values

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

Table 7: Re-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Original order ID	String	50-character alpha-numeric	`'orig_order_id'=>orig_order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`

Table 7: Re-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Transaction number	String	255-character variable character	`'txn_number'=>$txn_number`
E-Commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 1 Re-Authorization transaction optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Customer information	Object	N/A	`$mpgTxn->setCustInfo($mpgCustInfo);`
AVS	Object	N/A	`$mpgTxn->setAvsInfo($mpgAvsInfo);`
CVD **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`$mpgTxn->setCvdInfo($mpgCvdInfo);`

Sample Re-Authorization

```
<?php
require "../../mpgClasses.php";
/******************************* Request Variables *********************************/
$store_id='store5';
$api_token="yesguy";
/*************************** Transaction Associative Array *************************/
$txnArray=array('type'=>'reauth',
'order_id'=>'ord-'.date("dmy-G:i:s"),
'cust_id'=>'my cust id',
'amount'=>'0.50',
'orig_order_id'=>'ord-110515-10:55:31', //original pre-auth order_id
'txn_number'=>'31393-0_10', //original pre-auth txn number
'crypt_type'=>'7',
'dynamic_descriptor'=>'123456'
);
/****************************** Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************************* Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object ********************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************* Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType()."<br>");
print("\nTransAmount = " . $mpgResponse->getTransAmount()."<br>");
print("\nTxnNumber = " . $mpgResponse->getTxnNumber()."<br>");
print("\nReceiptId = " . $mpgResponse->getReceiptId()."<br>");
print("\nTransType = " . $mpgResponse->getTransType()."<br>");
print("\nReferenceNum = " . $mpgResponse->getReferenceNum()."<br>");
print("\nResponseCode = " . $mpgResponse->getResponseCode()."<br>");
print("\nISO = " . $mpgResponse->getISO()."<br>");
print("\nMessage = " . $mpgResponse->getMessage()."<br>");
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit()."<br>");
print("\nAuthCode = " . $mpgResponse->getAuthCode()."<br>");
print("\nComplete = " . $mpgResponse->getComplete()."<br>");
print("\nTransDate = " . $mpgResponse->getTransDate()."<br>");
print("\nTransTime = " . $mpgResponse->getTransTime()."<br>");
print("\nTicket = " . $mpgResponse->getTicket()."<br>");
print("\nTimedOut = " . $mpgResponse->getTimedOut()."<br>");
?>
```

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

ForcePost transaction object definition

```
$txnArray = array(‘type’=>’forcepost', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for ForcePost transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Force Post transaction values

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

Table 8: Force Post transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric	`'expdate'=>$expiry_date`
Authorization code	String	8-character alphanumeric	`'auth_code'=>$auth_code`
E-Commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 9: Force Post transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Basic Force Post

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************* Transactional Variables *****************************/
$type='forcepost';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='10.00';
$pan='4242424242424242';
$expiry_date='0812';
$auth_code='123456';
$crypt='7';
$dynamic_descriptor='123456';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'auth_code'=>$auth_code,
'crypt_type'=>$crypt,
'dynamic_descriptor'=>$dynamic_descriptor
);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response *************************************/
```

Sample Basic Force Post

```
    $mpgResponse=$mpgHttpPost->getMpgResponse();
    print("\nCardType = " . $mpgResponse->getCardType());
    print("\nTransAmount = " . $mpgResponse->getTransAmount());
    print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
    print("\nReceiptId = " . $mpgResponse->getReceiptId());
    print("\nTransType = " . $mpgResponse->getTransType());
    print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
    print("\nResponseCode = " . $mpgResponse->getResponseCode());
    print("\nISO = " . $mpgResponse->getISO());
    print("\nMessage = " . $mpgResponse->getMessage());
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    print("\nIssuerId = " . $mpgResponse->getIssuerId());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’purchasecorrection', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Purchase Correction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Purchase Correction transaction object values

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

Table 10: Purchase Correction transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Transaction number	String	255-character variable character	`'txn_number'=>$txn_number`
E-Commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 11: Purchase Correction transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample Purchase Correction

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$orderid='ord-110515-10:53:03';
$txnnumber='31387-0_10';
$dynamic_descriptor='1234';
## step 1) create transaction hash ###
$txnArray=array('type'=>'purchasecorrection',
'txn_number'=>$txnnumber,
'order_id'=>$orderid,
'crypt_type'=>'7',
'cust_id'=>'customer ID',
'dynamic_descriptor'=>$dynamic_descriptor
);
## step 2) create a transaction object passing the array created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
```

Sample Purchase Correction

```
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

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.

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

```
$txnArray = array(‘type’=>’refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Refund transaction object values

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

Table 12: Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Transaction number	String	255-character variable character	`'txn_number'=>$txn_number`
E-Commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 13: Refund transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Refund

```
<?php
##
## This program takes 4 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. trans number
##
## Example php -q TestRefund.php store1 yesguy my_order_id 45109-89-0
##
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$orderid='ord-110515-11:32:49';
$txnnumber='31451-0_10';
$dynamic_descriptor='123';
## step 1) create transaction array ###
$txnArray=array('type'=>'refund',
'txn_number'=>$txnnumber,
'order_id'=>$orderid,
'amount'=>'0.10',
'crypt_type'=>'7',
'cust_id'=> 'Customer ID',
'dynamic_descriptor'=>$dynamic_descriptor
);
## step 2) create a transaction object passing the array created in
```

Sample Refund

```
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print ("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

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

```
$txnArray = array(‘type’=>’ind_refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Independent Refund transaction values

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

Table 14: Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character alpha-numeric	`'pan'=>$pan`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
E-Commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 15: Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Independent Refund

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
```

Sample Independent Refund

```
## 2. api token
## 3. order id
##
## Example php -q TestIndependentRefund.php store1 yesguy unique_order_id
##
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$orderid='ord-'.date("dmy-G:i:s");
$dynamic_descriptor='123456';
## step 1) create transaction array ###
$txnArray=array('type'=>'ind_refund',
'order_id'=>$orderid,
'cust_id'=>'my cust id',
'amount'=>'1.00',
'pan'=>'4242424242424242',
'expdate'=>'1103',
'crypt_type'=>'7',
'dynamic_descriptor'=>$dynamic_descriptor
);
## step 2) create a transaction object passing the array created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

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 and Discover
- 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 and Discover.
- 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 object definition

```
$txnArray = array('type'=>'card_verification', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Card Verification transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Card Verification transaction values

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

Table 16: Card Verification transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Credit card number	String	20-character alpha-numeric	`'pan'=>$pan`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`

Table 16: Card Verification transaction object mandatory values

Value	Type	Limits	Set method
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
AVS	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`

Table 17: Basic Card Verification transaction object optional values

Value	Type	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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Card Verification

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token="yesguy";
$txnArray=array('type'=>'card_verification',
'order_id'=>'ord-'.date("dmy-G:i:s"),
'cust_id'=>'my cust id',
'pan'=>'4242424242424242',
'expdate'=>'1512',
'crypt_type'=>'7'
);
$mpgTxn = new mpgTransaction($txnArray);
/************************** AVS Variables *****************************/
```

Sample Card Verification

```
$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';
/************************** CVD Variables ******************************/
$cvd_indicator = '1';
$cvd_value = '198';
/********************** AVS Associative Array **************************/
$avsTemplate = array(
'avs_street_number'=>$avs_street_number,
'avs_street_name' =>$avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/********************** CVD Associative Array **************************/
$cvdTemplate = array(
'cvd_indicator' => $cvd_indicator,
'cvd_value' => $cvd_value
);
/************************** AVS Object *********************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************** CVD Object *********************************/
$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
/*********************** Credential on File *************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("12345678901234");
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCvdInfo($mpgCvdInfo);
$mpgTxn->setCofInfo($cof);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

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

```
$txnArray = array(‘type’=>’batchclose', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Batch Close transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Batch Close transaction values

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

Table 18: Batch Close transaction object mandatory values

Value	Type	Limits	Set method
ECR (electronic cash register) number	String	No limit (value provided by Moneris)	`ecr_number=>$ecr_number`

Sample Batch Close

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. ecr number
##
## Example php -q TestBatchClose.php store1 yesguy 66002173
##
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$ecr_number='66013455';
## step 1) create transaction array ###
$txnArray=array('type'=>'batchclose',
'ecr_number'=>$ecr_number
);
$mpgTxn = new mpgTransaction($txnArray);
## step 2) create mpgRequest object ###
$mpgReq=new mpgRequest($mpgTxn);
$mpgReq->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgReq->setTestMode(true); //false or comment out this line for production transactions
## step 3) create mpgHttpsPost object which does an https post ##
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);
## step 4) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
##step 5) get array of all credit cards
$creditCards = $mpgResponse->getCreditCards($ecr_number);
## step 6) loop through the array of credit cards and get information
for($i=0; $i < count($creditCards); $i++)
{
print "\nCard Type = $creditCards[$i]";
```

Sample Batch Close

```
print "\nPurchase Count = "
. $mpgResponse->getPurchaseCount($ecr_number,$creditCards[$i]);
print "\nPurchase Amount = "
. $mpgResponse->getPurchaseAmount($ecr_number,$creditCards[$i]);
print "\nRefund Count = "
. $mpgResponse->getRefundCount($ecr_number,$creditCards[$i]);
print "\nRefund Amount = "
. $mpgResponse->getRefundAmount($ecr_number,$creditCards[$i]);
print "\nCorrection Count = "
. $mpgResponse->getCorrectionCount($ecr_number,$creditCards[$i]);
print "\nCorrection Amount = "
. $mpgResponse->getCorrectionAmount($ecr_number,$creditCards[$i]);
}
?>
```

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

```
$txnArray = array(‘type’=>’opentotals', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Open Totals transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Open Totals transaction values

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

Table 19: Open Totals transaction object mandatory values

Value	Type	Limits	Set method
ECR (electronic cash register) number	String	No limit (value provided by Moneris)	`ecr_number=>$ecr_number`

Sample Open Totals

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
```

Sample Open Totals

```
## 3. ecr number
##
## Example php -q TestOpenTotals.php store1 yesguy 66002163
##
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$ecr_number='66013455';
## step 1) create transaction array ###
$txnArray=array('type'=>'opentotals',
'ecr_number'=>$ecr_number
);
$mpgTxn = new mpgTransaction($txnArray);
## step 2) create mpgRequest object ###
$mpgReq= new mpgRequest($mpgTxn);
$mpgReq->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgReq->setTestMode(true); //false or comment out this line for production transactions
## step 3) create mpgHttpsPost object which does an https post ##
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);
## step 4) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
##step 5) get array of all credit cards
$creditCards = $mpgResponse->getCreditCards($ecr_number);
## step 6) loop through the array of credit cards and get information
for($i=0; $i < count($creditCards); $i++)
{
print "\nCard Type = $creditCards[$i]";
print "\nPurchase Count = "
. $mpgResponse->getPurchaseCount($ecr_number,$creditCards[$i]);
print "\nPurchase Amount = "
. $mpgResponse->getPurchaseAmount($ecr_number,$creditCards[$i]);
print "\nRefund Count = "
. $mpgResponse->getRefundCount($ecr_number,$creditCards[$i]);
print "\nRefund Amount = "
. $mpgResponse->getRefundAmount($ecr_number,$creditCards[$i]);
print "\nCorrection Count = "
. $mpgResponse->getCorrectionCount($ecr_number,$creditCards[$i]);
print "\nCorrection Amount = "
. $mpgResponse->getCorrectionAmount($ecr_number,$creditCards[$i]);
}
?>
```

3 Credential on File

- 3.1 About Credential on File
- 3.2 Credential on File Info Object and Variables
- 3.3 Credential on File Transaction Types
- 3.4 Initial Transactions in Credential on File
- 3.6 Credential on File and Converting Temporary Tokens
- 3.5 Vault Tokenize Credit Card and Credential on File
- 3.7 Card Verification and Credential on File Transactions

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.

Object:

cof

Variables in the cof object:

Payment Indicator
Payment Information
Issuer ID

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

For more information, see 1 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 – cavv_purchase
- 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
- Purchase
- Pre-Authorization
- Purchase with 3-D Secure – cavv_purchase
- Pre-Authorization with 3-D Secure – cavv_preauth
- Purchase with Vault
- Pre-Authorization with Vault
- Card Verification
- Card Verification with Vault
- Vault Add Credit Card
- Vault Update Credit Card
- Recurring Billing transactions

3.4 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

3.5 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.

3.6 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 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.

For more information about Vault Temporary Token Add transaction, see 1 Vault Temporary Token Add.

3.7 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 9.2 Card Validation Digits (CVD).

For a complete list of these variables, see each transaction type or Definitions 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. see 1 Card Verification.

3.7.1 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
Vault Add Credit Card – res_add_cc
Vault Update Credit Card – res_update_cc
Recurring Billing transactions, if:

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

3.7.2 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.

3.7.3 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.

3.7.4 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.

3.7.5 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
- 4.2 Vault Transaction Types
- 4.3 Vault Administrative Transactions
- 4.4 Vault Financial Transactions
- 4.5 Hosted Tokenization

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.

ResAddCC transaction object definition

```
$txnArray = array(‘type’=>’res_add_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for ResAddCC transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

ResAddCC transaction values

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

Table 20: Vault Add Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Credit card number	String	20-character alphanumeric	`'pan'=>$pan`
Expiry date	String	4-character alphanumeric (YYMM format)	`'expdate'=>$expiry_date`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Table 21: Vault Add Credit Card transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Email address	String	30-character alphanumeric	`'email'=>$email`

Table 21: Vault Add Credit Card transaction optional values

Value	Type	Limits	Set method
Phone number	String	30-character alpha-numeric	`'phone'=>$phone`
Note	String	30-character alpha-numeric	`'note'=>$note`
Data key format	String	2-character alpha-numeric	`'data_key_format'=>$data_key_format`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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 alpha-numeric variable length	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Add Credit Card

```
<?php
##
## Example php -q TestResAddCC.php store3 yesguy
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_add_cc';
$cust_id='customer1';
$phone = '5555551234';
$email = 'bob@smith.com';
$note = 'this is my note';
$pan='5454545454545454';
$expiry_date='1412';
```

Sample Vault Add Credit Card

```
$crypt_type='1';
$data_key_format = "0";
$avs_street_number = '123';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '90210';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
'note'=>$note,
'pan'=>$pan,
'expdate'=>$expiry_date,
//'data_key_format'=>$data_key_format, //optional
'crypt_type'=>$crypt_type
);
/********************** AVS Associative Array *********************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************** AVS Object ***************************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setIssuerId("139X3130ASCXAS9");
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCofInfo($cof);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.3.1.1 Vault Data Key

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

```
print("\nDataKey = " . $mpgResponse->getDataKey());
```

The data key response field is populated when you send a Vault Add Credit Card – ResAddCC (page 52), Vault Encrypted Add Credit Card – EncResAddCC (page 56), Vault Tokenize Credit Card – ResTokenizeCC (page 79), Vault Temporary Token Add – ResTempAdd (page 59) or Vault Add Token – ResAddToken (page 76) 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

```
$txnArray = array('type'=>'enc_res_add_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Encrypted Add Credit Card transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Encrypted Add Credit Card transaction values

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

Table 22: Vault Encrypted Add Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Encrypted Track2 data	String	40-character numeric	`'enc_track2'=>$enc_track2`
Device type	String	30-character alphanumeric	`'device_type'=>$device_type`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 23: Vault Encrypted Add Credit Card transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
AVS information	Object	Not applicable. Click hereSee 9.1 (page 281).	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Email address	String	30-character alpha-numeric	`'email'=>$email`

Phone number	String	30-character alpha-numeric	`enc_res_add_cc`
			`'phone'=>$phone`

Note	String	30-character alpha-numeric	`'note'=>$note`
Data key format[1]	String	2-character alpha-numeric	`'data_key_format'=>$data_key_format`

Sample Vault Encrypted Add Credit Card

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables *****************************/
$type='enc_res_add_cc';
$cust_id='cust1';
$phone = '6479996999';
$email = 'bob@smith.com';
$note = 'this is my note';
$enc_track2 = 'ENCRYPTEDTRACK2DATA';
$device_type='idtech_bdk';
$data_key_format="0";
$crypt_type='7';
$avs_street_number = '11';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = 'm8x2x2';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
```

[1] Available to Canadian integrations only.

Sample Vault Encrypted Add Credit Card

```
'note'=>$note,
'enc_track2'=>$enc_track2,
'device_type'=>$device_type,
//'data_key_format'=>$data_key_format, //optional
'crypt_type'=>$crypt_type
);
/********************** AVS Associative Array **********************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************** AVS Object ****************************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

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

```
$txnArray = array(‘type’=>’res_temp_add', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Temporary Token Add transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Temporary Token Add transaction values

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

Table 24: Vault Temporary Token Add transaction object mandatory values

Value	Type	Limits	Set method
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric	`'expdate'=>$expiry_date`
Duration	String	3-character numeric maximum 15 minutes	`'duration'=>$duration`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 25: Vault Temporary Token Add transaction optional values

Value	Type	Limits	Set method
Data key format[1]	String	2-character alphanumeric	`'data_key_format'=>$data_key_format`

Sample Vault Temporary Token Add

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_temp_add';
$pan='5454545454545454';
$expiry_date='1509';
$duration='900';
$data_key_format = "0";
$crypt_type='7';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'pan'=>$pan,
'expdate'=>$expiry_date,
'duration'=>$duration,
//'data_key_format'=>$data_key_format, //optional
'crypt_type'=>$crypt_type
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\Masked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
?>
'crypt_type'=>$crypt_type
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
```

[1] Available to Canadian integrations only.

Sample Vault Temporary Token Add

```
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\Masked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
?>
```

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 441).

4.3.3 Vault Update Credit Card – 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.

Things to Consider:

- This will update a profile to contain Credit Card information by referencing the profile’s unique data key. If the profile which is being updated was already a Credit Card profile, all information contained within it will simply be updated as indicated by the submitted fields. This means that all fields are optional, and only those fields that are submitted will be updated.
- To update a specific field on the profile, only set that specific element using the corresponding set method.

Vault Update Credit Card transaction object definition

```
$txnArray = array(‘type’=>’res_update_cc', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Update Credit Card transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Update Credit Card transaction values

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

Table 26: Vault Update Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$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.

Table 27: Vault Update Credit Card transaction optional values

Value	Type	Limits	Set method
Credit card number	String	20-character alpha-numeric	`'pan'=>$pan`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
AVS information	Object	n/a	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Email address	String	30-character alpha-numeric	`'email'=>$email`

Value	Type	Limits	Set method
Phone number	String	30-character alpha-numeric	`'phone'=>$phone`
Note	String	30-character alpha-numeric	`'note'=>$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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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 alpha-numeric variable length	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Update Credit Card

```
<?php
##
```

Sample Vault Update Credit Card

```
## Example php -q TestResUpdateCC.php store3 yesguy
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_update_cc';
$data_key='D8cpd4r7REXoN8NIJPi512xPh';
$cust_id='customer1';
$phone = '5555555555';
$email = 'bob@smith.com';
$note = 'stuff';
$pan='5454545454545454';
$expiry_date='0909';
$crypt_type='7';
$avs_street_number = '123';
$avs_street_name = 'stuff dr';
$avs_zipcode = '90215';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key,
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
'note'=>$note,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt_type
);
/********************** AVS Associative Array *********************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************** AVS Object ***************************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setIssuerId("168451306048014");
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCofInfo($cof);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
```

Sample Vault Update Credit Card

```
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.3.3.1 Vault Encrypted Update CC - EncResUpdateCC

Vault Encrypted Update CC transaction object definition

```
$txnArray = array(‘type’=>’enc_res_update_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Encrypted Update CC transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Encrypted Update CC transaction values

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

Table 28: Vault Encrypted Update CC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`
Encrypted Track2 data	String	Variable length	`'enc_track2'=>$enc_track2`
Device type	String	30-character alpha-numeric	`'device_type'=>$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.

Table 29: Vault Encrypted Update CC transaction optional values

Value	Type	Limits	Set method
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
AVS information	Object	Not applicable. Click hereSee 9.1 (page 281).	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Email address	String	30-character alphanumeric	`'email'=>$email`
Phone number	String	30-character alphanumeric	`'phone'=>$phone`
Note	String	30-character alphanumeric	`'note'=>$note`

Sample Vault Encrypted Update CC - CA

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='enc_res_update_cc';
$data_key='F91LyeEJjv8OvpOdmXYWKh7dV';
```

Sample Vault Encrypted Update CC - CA

```
$cust_id='cust2';
$phone = '4169996999';
$email = 'bob@email.com';
$note = 'note4';
$enc_track2 = 'ENCRYPTEDTRACK2DATA';
$device_type='idtech_bdk';
$crypt_type='7';
$avs_street_number = '3300';
$avs_street_name = 'bloor street west';
$avs_zipcode = 'm8x2x3';
/********************** Transactional Associative Array *********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key,
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
'note'=>$note,
'enc_track2'=>$enc_track2,
'device_type'=>$device_type,
'crypt_type'=>$crypt_type
);
/********************* AVS Associative Array *******************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************* AVS Object *************************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/*************************** Transaction Object ***************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
/***************************** Request Object *****************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/**************************** HTTPS Post Object ***************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/****************************** Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

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

```
$txnArray = array(‘type’=>’res_delete', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Delete transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Delete transaction values

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

Table 30: Vault Delete transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`

Sample Vault Delete

```
<?php
##
## Example php -q TestResDelete.php store3 yesguy
##
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables *****************************/
$type='res_delete';
$data_key='YjNEwYw6U2pPwquXOkOme3G7g';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key
);
```

Sample Vault Delete

```
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.3.5 Vault Lookup Full - ResLookupFull

Vault Lookup Full transaction object definition

```
$txnArray = array(‘type’=>’res_lookup_full', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Lookup Full transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Lookup Full transaction values

Table 31: Vault Lookup Full transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`

Sample Vault Lookup Full

```
<?php
##
## Example php -q TestResLookupFull.php store3 yesguy
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_lookup_full'; //will return both the full & masked card number
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nPan = " . $mpgResponse->getResDataPan());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.3.6 Vault Lookup Masked – ResLookupMasked

Vault Lookup Masked transaction object definition

```
$txnArray = array(‘type’=>’res_lookup_masked', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Lookup Masked transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Lookup Masked transaction values

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

Table 32: Vault Lookup Masked transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`

Sample Vault Lookup Masked

```
<?php
##
## Example php -q TestResLookupMasked.php store3 yesguy
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_lookup_masked'; //will only return the masked card number
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
```

Sample Vault Lookup Masked

```
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.3.7 Vault Get Expiring – ResGetExpiring

Vault Get Expiring transaction object definition

```
$txnArray = array(‘type’=>’res_get_expiring', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Get Expiring transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Get Expiring transaction values

ResGetExpiring transaction object mandatory values: None.

Sample Vault Get Expiring

```
<?php
##
## Example php -q TestResGetExpiring.php store3 yesguy
##
//There is a max number of attempts set for this transaction per calendar day
//Can not surpass or will receive Invalid Transaction error
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_get_expiring';
/*********************** Transactional Associative Array **********************/
$txnArray = array( 'type'=>$type );
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
```

Sample Vault Get Expiring

```
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/**************************** HTTPS Post Object ****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/****************************** Response ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
$DataKeys = $mpgResponse->getDataKeys();
for($i=0; $i < count($DataKeys); $i++)
{
$mpgResponse->setResolveData($DataKeys[$i]);
print("\n\nData Key = " . $DataKeys[$i]);
print("\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
}
?>
```

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 441).

4.3.8 Vault Is Corporate Card - ResIscorporateCard

Vault Is Corporate Card transaction object definition

```
$txnArray = array(‘type’=>’res_iscorporatecard', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Is Corporate Card transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Is Corporate Card transaction values

Table 33: Vault Is Corporate Card transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`

Sample Vault Is Corporate Card

```
<?php
##
## Example php -q TestResIscorporatecard.php moneris hurgle
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_iscorporatecard';
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'data_key'=>$data_key
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
?>
```

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 441).

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

```
$txnArray = array(‘type’=>’res_add_token', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Add Token transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Add Token transaction values

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

Table 34: Vault Add Token transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	28-character alpha-numeric	`'data_key'=>$data_key`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Value	Type	Limits	Set method
when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".			

Table 35: Vault Add Token transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Email address	String	30-character alphanumeric	`'email'=>$email`
Phone number	String	30-character alphanumeric	`'phone'=>$phone`
Note	String	30-character alphanumeric	`'note'=>$note`
Data key format[1]	String	2-character alphanumeric	`'data_key_format'=>$data_key_format`

[1] Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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 numeric variable length	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Add Token

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='res_add_token';
$temp_data_key='ot-mtNKdu8NcxDoChqOJKZJZ1BOB';
$cust_id='customer1';
$phone = '5555551234';
$email = 'bob@smith.com';
$note = 'this is my note';
$expiry_date='1811';
$data_key_format = "0";
$crypt_type='1';
$avs_street_number = '123';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '90210';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'data_key'=>$temp_data_key,
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
'note'=>$note,
'expdate'=>$expiry_date,
//'data_key_format'=>$data_key_format, //optional
'crypt_type'=>$crypt_type
);
/********************** AVS Associative Array *********************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************** AVS Object ***************************************/
```

Sample Vault Add Token

```
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setIssuerId("168451306048014");
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCofInfo($cof);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

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 :

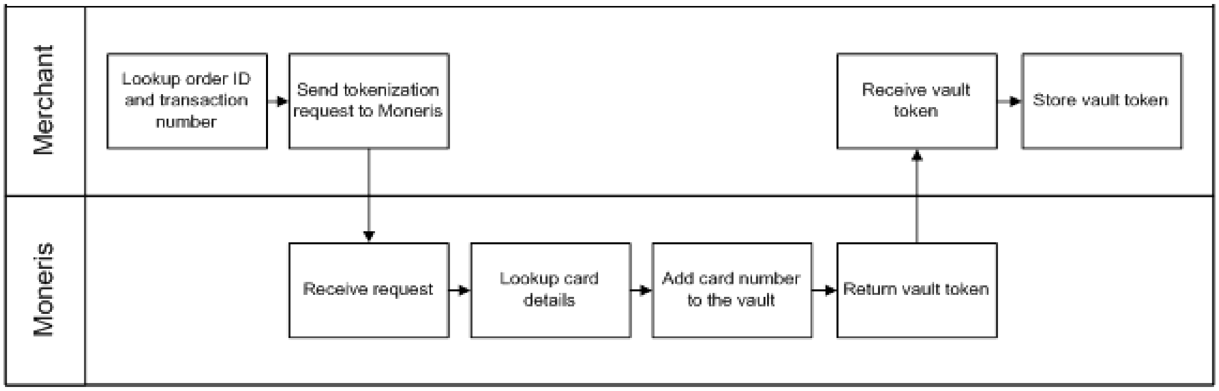


Figure 1: Tokenize process diagram

Vault Tokenize Credit Card transaction object definition

```
$txnArray = array(‘type’=>’res_tokenize_cc', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Tokenize Credit Card transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Tokenize Credit Card transaction values

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.

Table 36: Vault Tokenize Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`

Table 37: Vault Tokenize Credit Card transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Email address	String	30-character alpha-numeric	`'email'=>$email`
Phone number	String	30-character alpha-numeric	`'phone'=>$phone`
Note	String	30-character alpha-numeric	`'note'=>$note`
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Data key format[1]	String	2-character alpha-numeric	`'data_key_format'=>$data_key_format`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	*Object*	N/A	`$mpgTxn->setCofInfo($cof);`

[1] Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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 Definitions 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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables *****************************/
$type='res_tokenize_cc';
$order_id='res-purch-110515-12:56:49';
$txn_number='31570-0_10';
$data_key_format = "0";
$cust_id='customer1';
$phone = '4165555555';
$email = 'bob@smith.com';
$note = 'this is my note';
$avs_street_number = '123';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '90210';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=>$txn_number,
//'data_key_format'=>$data_key_format, //optional
'cust_id'=>$cust_id,
'phone'=>$phone,
'email'=>$email,
'note'=>$note
);
/********************** AVS Associative Array **********************************/
$avsTemplate = array(
```

Sample Vault Tokenize Credit Card

```
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
/************************** AVS Object ****************************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/******************* Credential on File ***********************************/
$cof = new CofInfo();
$cof->setIssuerId("168451306048014");
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCofInfo($cof);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

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 38: 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

```
$txnArray = array(‘type’=>’res_purchase_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Purchase with Vault transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Purchase with Vault transaction values

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

Table 39: Purchase with Vault transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-	`'amount'=>$amount`

Value	Type	Limits	Set method
		pet)	
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Table 40: Purchase with Vault transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Expiry date	String	4-character numeric YYMM format. (Note that this is reversed from the date displayed on the card, which is MMYY)	`'expdate'=>$expiry_date`
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Customer information	Object	N/A	`$mpgTxn->setCustInfo`

Value	Type	Limits	Set method
			`($mpgCustInfo);`
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
Recurring billing	Object	N/A	`$mpgTxn->setRecur($mpgRecur);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Purchase with Vault

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC.php store3 yesguy unique_order_id 1.00
##
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='store5';
$api_token='yesguy';
```

Sample Purchase with Vault

```
/************************ Transaction Variables ******************************/
$data_key='ot-odvn9lBTZm0lSWyQgansBqQi3';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';
$expdate='1911'; //For Temp Tokens only
/************************ Transaction Array **********************************/
$txnArray=array('type'=>'res_purchase_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'crypt_type'=>$crypt_type,
//'expdate'=>$expdate,
'dynamic_descriptor'=>'12484'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("12345678901234");
$mpgTxn->setCofInfo($cof);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
```

Sample Purchase with Vault

```
    print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
    print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
    print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
    print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
    ?>
```

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 441).

4.4.3 Pre-Authorization with Vault – ResPreauthCC

Pre-Authorization with Vault transaction object definition

```
$txnArray = array(‘type’=>’res_preauth_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Pre-Authorization with Vault transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Pre-Authorization with Vault transaction values

Table 41: Pre-Authorization with Vault transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25- character alpha-numeric	`'data_key'=>$data_key`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`'amount'=>$amount`
E-commerce indicator	String	1-character alpha-	`'crypt_type'=>$crypt`

Table 41: Pre-Authorization with Vault transaction object mandatory values (continued)

Value	Type	Limits	Set method
		numeric	
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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Table 42: Pre-Authorization with Vault transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`

Value	Type	Limits	Set method
Customer information	Object	N/A	`$mpgTxn->setCustInfo ($mpgCustInfo);`
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
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	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Inform-	*String*	1-character numeric	

Value	Type	Limits	Set Method
ation			$cof->setPaymentInformation ("PAYMENT_INFO_VALUE"); **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Pre-Authorization with Vault

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPreauthCC.php store3 yesguy unique_order_id cust_id 15.00 1
##
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables *******************************/
$data_key='ot-H0q8anK6eeHm0NDe9cwXkDvUw';
$orderid='res-preauth-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust'; //if sent will be submitted, otherwise cust_id from profile will be used
$crypt_type='1';
//$expdate='1512';
/************************ Transaction Array ***********************************/
$txnArray =array('type'=>'res_preauth_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'crypt_type'=>$crypt_type,
//'expdate=>$expdate,
'dynamic_descriptor'=>'12424'
);
/************************ Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File ***********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("12345678901234");
$mpgTxn->setCofInfo($cof);
/************************ Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object *******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
```

Sample Pre-Authorization with Vault

```
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 441).

4.4.4 Vault Independent Refund CC - ResIndRefundCC

Vault Independent Refund transaction object definition

```
$txnArray = array(‘type’=>’res_ind_refund_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Independent Refund transaction values

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

Table 43: Vault Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`'amount'=>$amount`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 44: Vault Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample Vault Independent Refund

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResIndRefundCC.php store3 yesguy unique_order_id cust_id 15.00 1
##
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
$orderid='res-ind-refund-'.date("dmy-G:i:s");
$amount='1.00';
$custid='';
$crypt_type='1';
/************************ Transaction Array **********************************/
$txnArray =array('type'=>'res_ind_refund_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'crypt_type'=>$crypt_type,
'dynamic_descriptor'=>'12346'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
```

Sample Vault Independent Refund

```
    print("\nNote = " . $mpgResponse->getResDataNote());
    print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
    print("\nExp Date = " . $mpgResponse->getResDataExpDate());
    print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
    print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
    print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
    print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
    ?>
```

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 441).

4.4.5 Force Post with Vault - ResForcePostCC

Force Post with Vault transaction object definition

```
$txnArray = array(‘type’=>’res_forcepost_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Force Post with Vault transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Force Post with Vault transaction object values

Table 1 Force Post with Vault transaction object mandatory values

Value	Type	Limits	Set Method
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Data key	String	25-character alphanumeric	`'data_key'=>$data_key`
Authorization code	String	8-character alphanumeric	`'auth_code'=>$auth_code`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 2 Force Post with Vault transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic Descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Force Post with Vault

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables *******************************/
$data_key='uroyVNSxzjk5hHoT0kpQDBCw4';
$orderid='res-forcepost-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='7';
$auth_code='256452';
$dynamic_descriptor='my descriptor';
/************************ Transaction Array ***********************************/
$txnArray=array('type'=>'res_forcepost_cc',
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'data_key'=>$data_key,
'crypt_type'=>$crypt_type,
'auth_code'=>$auth_code,
'dynamic_descriptor'=>$dynamic_descriptor
);
/************************ Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
```

Sample Force Post with Vault

```
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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

```
$txnArray = array(‘type’=>’res_card_verification_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Card Verification with Vault transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Card Verification with Vault transaction values

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

Table 45: Card Verification with Vault transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
AVS	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD	Object	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`$mpgTxn->setCofInfo($cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Card Verification with Vault

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC.php store3 yesguy unique_order_id 1.00
##
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
```

Sample Card Verification with Vault

```
/************************ Transaction Variables *******************************/
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
$orderid='res-purch-'.date("dmy-G:i:s");
$crypt_type='1';
$expdate='1911'; //for temp token
/************************ Transaction Array ***********************************/
$txnArray=array('type'=>'res_card_verification_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'crypt_type'=>$crypt_type,
'expdate'=>$expdate
);
/************************** CVD Variables ******************************/
$cvd_indicator = '1';
$cvd_value = '198';
/********************** CVD Associative Array **************************/
$cvdTemplate = array(
'cvd_indicator' => $cvd_indicator,
'cvd_value' => $cvd_value
);
$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
/************************** AVS Variables ******************************/
//The AVS portion is optional if AVS details are already stored in this profile
//If AVS details are resent in Purchase transaction, they will replace stored details
$avs_street_number = '';
$avs_street_name = 'bloor st';
$avs_zipcode = '111111';
/********************** AVS Associative Array **************************/
$avsTemplate = array(
'avs_street_number' => $avs_street_number,
'avs_street_name' => $avs_street_name,
'avs_zipcode' => $avs_zipcode
);
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************ Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCvdInfo($mpgCvdInfo);
$mpgTxn->setAvsInfo($mpgAvsInfo);
/******************* Credential on File ***********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("12345678901234");
$mpgTxn->setCofInfo($cof);
/************************ Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
```

Sample Card Verification with Vault

```
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCVDResponse = " . $mpgResponse->getCvdResultCode());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

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 (https://developer.moneris.com).

5 INTERAC® Online Payment

- 5.1 About INTERAC® Online Payment Transactions
- 5.2 Other Documents and References
- 5.3 Website and Certification Requirements
- 5.4 Transaction Flow for INTERAC® Online Payment
- 5.5 Sending an INTERAC® Online Payment Purchase Transaction
- 5.6 INTERAC® Online Payment Purchase
- 5.7 INTERAC® Online Payment Refund
- 5.8 INTERAC® Online Payment Field Definitions

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 Gateway API solution to allow online payments using credit and debit cards.

INTERAC® Online Payment transactions via the 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 API.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 17.5 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 458) and "Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing" on page 462 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 470) 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 458) or Appendix F (page 462)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 103). 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 467). They will **not** be required to complete any of the test cases.

Your clients must also complete the Merchant Requirement checklist (Appendix G, page 467). 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 46 may experience delays in the certification or registration process of up to 7 days.

Table 46: 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

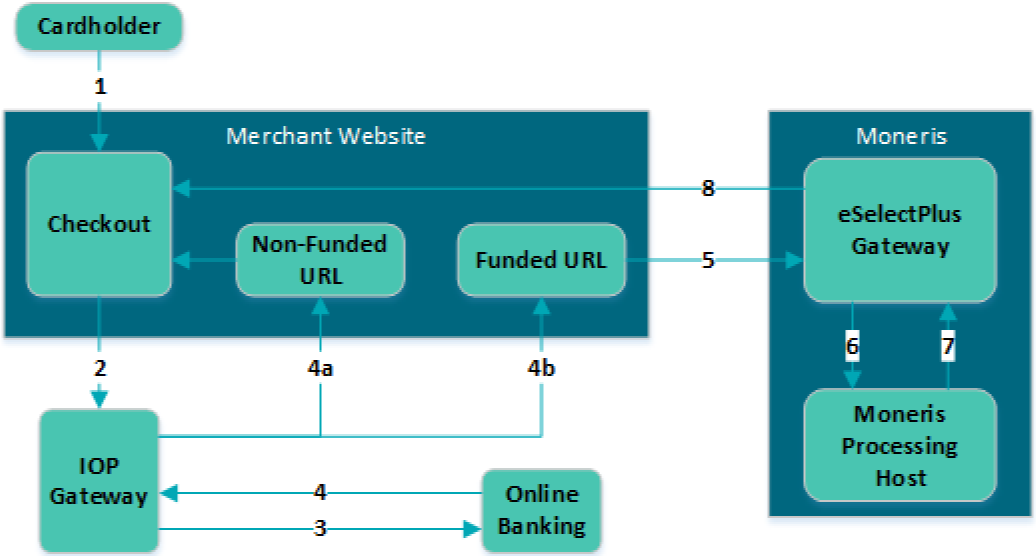


Figure 2: INTERAC® Online Payment transaction flow diagram

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 112) 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 47: 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 112):

- 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

```
$txnArray = array(‘type’=>’idebit_purchase', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for INTERAC® Online Payment Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

INTERAC® Online Payment Purchase transaction values

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

Table 48: INTERAC® Online Payment transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Track2 data	String	40-character alpha-numeric	`'idebit_track2'=>$idebit_track2`

Table 49: INTERAC® Online Payment Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Customer information	Object	Not applicable. Click hereSee Section 14 (page 354).	`$mpgTxn->setCustInfo($mpgCustInfo);`

Sample INTERAC® Online Payment Purchase

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token= 'yesguy';
$orderid= 'ord-'.date("dmy-G:i:s");
## step 1) create transaction hash ###
$txnArray=array('type'=>'idebit_purchase',
'order_id'=>$orderid,
'cust_id'=>'my cust id',
'amount'=>'50.00',
'idebit_track2'=>'3728024906540591206=0609AAAAAAAAAAAAA'
);
## step 2) create a transaction object passing the hash created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
```

Sample INTERAC® Online Payment Purchase

```
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

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

```
$txnArray = array(‘type’=>’idebit_refund', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for INTERAC® Online Payment Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

INTERAC® Online Payment Refund transaction object values

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

Table 50: INTERAC® Online Payment Refund transaction object mandatory variables

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`

Table 51: INTERAC® Online Payment Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample code

Sample INTERAC® Online Payment Refund

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token= 'yesguy';
$orderid= 'ord-080515-12:37:07';
$txn_number='20186-0_10';
## step 1) create transaction hash ###
$txnArray=array('type'=>'idebit_refund',
'order_id'=>$orderid,
'amount'=>'50.00',
'txn_number'=>$txn_number
);
## step 2) create a transaction object passing the hash created in
## step 1.
$mpgTxn = new mpgTransaction($txnArray);
## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();
## step 6) retrieve data using get methods
print ("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

5.8 INTERAC® Online Payment Field Definitions

Table 52: Field Definitions

Value	Characters	Limits
	Description	
IDEBIT_MERCHNUM	5-14	Numbers and uppercase letters
	This field is provided by Moneris. For example, 0003MONMPGXXXX.	

Table 52: Field Definitions (continued)

Value	Characters	Limits
	Description	
IDEBIT_TERMID	8	Numbers and uppercase letters
	Optional field	
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.	

Table 52: Field Definitions (continued)

Value	Characters	Limits
	Description	
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.	
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.	

Table 52: Field Definitions (continued)

Value	Characters	Limits
	Description	
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

- 6.1 Mag Swipe Transaction Type Definitions
- 6.2 Mag Swipe Purchase
  - 6.2.1 Encrypted Mag Swipe Purchase
- 6.3 Mag Swipe Pre-Authorization
  - 6.3.1 Encrypted Mag Swipe Pre-Authorization
- 6.4 Mag Swipe Completion
- 6.5 Mag Swipe Force Post
  - 6.5.1 Encrypted Mag Swipe Force Post
- 6.6 Mag Swipe Purchase Correction
- 6.7 Mag Swipe Refund
- 6.8 Mag Swipe Independent Refund
  - 6.8.1 Encrypted Mag Swipe Independent Refund

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

```
$txnArray = array(‘type’=>’track2_purchase', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Purchase transaction values

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

Table 53: Mag Swipe Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`

Credit card number
OR
Track2 data
	String	20-character numeric
		OR
		40-character numeric
			`'pan'=>$pan`
			OR
			`track2=>$track`

Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
POS code	String	2-character numeric	`'pos_code'=>$pos_code`

Table 54: Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
AVS information	Object	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
CVD information	Object	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Mag Swipe Purchase

```
<?php
require "../../mpgClasses.php";
```

Sample Mag Swipe Purchase

```
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables ******************************/
$orderid='ord-'.date("dmy-G:i:s");
$custid='customerID';
$amount='1.00';
/************ Swipe card and read Track1 and/or Track2 ***********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);
$startDelim = ";";
$firstChar = $track1{0};
$track = '';
if($firstChar==$startDelim)
{
$track = $track1;
}
else
{
$track2 = fgets ($stdin);
$track = $track2;
}
$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_purchase',
order_id=>$orderid,
cust_id=>$custid,
amount=>$amount,
track2=>$track,
pan=>'',
expdate=>'',
pos_code=>'00',
dynamic_descriptor=>'nqa'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
```

Sample Mag Swipe Purchase

```
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

6.2.1 Encrypted Mag Swipe Purchase

Encrypted Mag Swipe Purchase transaction object definition

```
$txnArray = array(‘type’=>’enc_track2_purchase', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Encrypted Mag Swipe Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Encrypted Mag Swipe Purchase transaction values

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

Table 55: Encrypted Mag Swipe Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Encrypted Track2 data	String	n/a	`'enc_track2'=>$enc_track2`
POS code	String	2-character numeric	`'pos_code'=>$pos_code`
Device type	String	30-character alpha-numeric	`'device_type'=>$device_type`

Table 56: Encrypted Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
AVS information	Object	n/a	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample Encrypted Mag Swipe Purchase

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$orderid="ord_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="ENCRYPTEDTRACK2DATA";
$pos_code="00";
$device_type='idtech_bdk';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'enc_track2_purchase',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
enc_track2=>$enc_track2,
pos_code=>$pos_code,
device_type=>$device_type
);
/********************** AVS Associative Array *************************/
$avsTemplate = array(
avs_street_number=>"123",
avs_street_name =>"bloor st w",
avs_zipcode => "90210"
);
/************************** AVS Object ********************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set AVS and CVD *****************************/
$mpgTxn->setAvsInfo($mpgAvsInfo);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
```

Sample Encrypted Mag Swipe Purchase

```
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

6.3 Mag Swipe Pre-Authorization

Mag Swipe Pre-Authorization transaction object definition

```
$txnArray = array(‘type’=>’track2preauth', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Pre-Authorization transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Pre-Authorization transaction values

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

Table 57: Mag Swipe Pre-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`

Credit card number
OR
	String	20-character numeric
		OR
			`'pan'=>$pan`
			OR
			`track2=>$track`

Table 57: Mag Swipe Pre-Authorization transaction object mandatory values (continued)

Value	Type	Limits	Set method
Track2 data		40-character numeric	
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
POS code	String	2-character numeric	`'pos_code'=>$pos_code`

Table 58: Mag Swipe Pre-Authorization transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Mag Swipe Pre-Authorization

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables *******************************/
$orderid='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='';
$expdate='';
/************ Swipe card and read Track1 and/or Track2 ***********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);
$startDelim = ";";
$firstChar = $track1{0};
$track = '';
if($firstChar==$startDelim)
{
$track = $track1;
}
else
```

Sample Mag Swipe Pre-Authorization

```
{
$track2 = fgets ($stdin);
$track = $track2;
}
$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_preauth',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
track2=>$track,
pan=>$pan,
expdate=>$expdate,
pos_code=>'00',
dynamic_descriptor=>'nqa'
);
/************************ Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

6.3.1 Encrypted Mag Swipe Pre-Authorization

Encrypted Mag Swipe Pre-Authorization transaction object definition

```
$txnArray = array(‘type’=>’enc_track2_preauth', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Encrypted Mag Swipe Pre-Authorization transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Encrypted Mag Swipe Pre-Authorization transaction values

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

Table 59: Encrypted Mag Swipe Pre-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	'order_id'=>$order_id
Amount	String	(missing or bad snip-pet)	'amount'=>$amount
Credit card number OR Encrypted Track2	String	20-character numeric OR n/a	'pan'=>$pan OR 'enc_track2'=>$enc_track2
POS code	String	2-character numeric	'pos_code'=>$pos_code
Device type	String	30-character alpha-numeric	'device_type'=>$device_type

Table 60: Encrypted Mag Swipe Pre-Authorization transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	'cust_id'=>$cust_id
Status Check	Boolean	true/false	$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);

Sample Encrypted Mag Swipe Pre-Authorization

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$orderid="ord_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="ENCRYPTEDTRACK2DATA";
$pos_code="00";
$device_type='idtech_bdk';
```

Sample Encrypted Mag Swipe Pre-Authorization

```
/************************ Transaction Array **********************************/
$txnArray=array(type=>'enc_track2_preauth',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
enc_track2=>$enc_track2,
pos_code=>$pos_code,
device_type=>$device_type,
dynamic_descriptor=>'12345'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

6.4 Mag Swipe Completion

Mag Swipe Completion transaction object definition

```
$txnArray = array(‘type’=>’track2_completion', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Completion transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Completion transaction values

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

Table 61: Mag Swipe Completion transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Transaction number	String	255-character variable character	`'txn_number'=>$txn_number`
Completion Amount	String	(missing or bad snip-pet)	`'comp_amount'=>$comp_amount`

POS code	String	2-character numeric	`track2completion`
			`'pos_code'=>$pos_code`

Table 62: Mag Swipe Completion transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample Mag Swipe Completion

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
//$status='false';
/************************ Transaction Variables ******************************/
$orderid='ord-110515-15:44:10';
$txnnumber='32083-0_10';
$compamount='1.00';
$dynamic_descriptor='nqa';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_completion',
order_id=>$orderid,
comp_amount=>$compamount,
txn_number=>$txnnumber,
pos_code=>'00',
```

Sample Mag Swipe Completion

```
dynamic_descriptor=>$dynamic_descriptor
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

6.5 Mag Swipe Force Post

Mag Swipe Force Post transaction object definition

```
$txnArray = array(‘type’=>’track2_forcepost', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Force Post transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Force Post transaction mandatory arguments

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

Table 63: Mag Swipe Force Post transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	'order_id'=>$order_id
Amount	String	(missing or bad snippet)	'amount'=>$amount
Credit card number OR Track2 data	String	20-character numeric OR 40-character numeric	'pan'=>$pan OR track2=>$track
Expiry date	String	4-character alphanumeric (YYMM format)	'expdate'=>$expiry_date
POS code	String	2-character numeric	'pos_code'=>$pos_code
Authorization code	String	8-character alphanumeric	'auth_code'=>$auth_code

Table 64: Mag Swipe Force Post transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	'cust_id'=>$cust_id
Status Check	Boolean	true/false	$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
Dynamic descriptor	String	20-character alphanumeric	track2forcePost 'dynamic_descriptor'=>$dynamic_descriptor

Sample Mag Swipe Force Post

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
```

Sample Mag Swipe Force Post

```
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables *******************************/
$orderid='ord-'.date("dmy-G:i:s");
$custid='cust id';
$amount='1.00';
$authcode='123456';
/*************** Swipe Card and read Track1 and/or Track2 *********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);
$startDelim = ";";
$firstChar = $track1{0};
$track = '';
if($firstChar!==$startDelim)
{
$track = $track1;
}
else
{
$track2 = fgets ($stdin);
$track = $track2;
}
$track = trim($track);
/************************ Transaction Array ***********************************/
$txnArray=array(type=>'track2_forcepost',
order_id=>$orderid,
cust_id=>$custid,
amount=>$amount,
track2=>$track,
pan=>'',
expdate=>'',
pos_code=>'00',
auth_code=>$authcode,
dynamic_descriptor=>'nqa'
);
/************************ Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
```

Sample Mag Swipe Force Post

```
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    ?>
```

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

```
$txnArray=array(type=>'enc_track2_forcepost', ...);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Encrypted Mag Swipe Force Post transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Encrypted Track2 data	String	n/a	`'enc_track2'=>$enc_track2`

Value	Type	Limits	Set Method
POS Code	String	2-character numeric	'pos_code'=>$pos_code
Device type	String	30-character alpha-numeric	'device_type'=>$device_type
Authorization Code	String	8-character alpha-numeric	'auth_code'=>$auth_code

Table 2 Encrypted Mag Swipe Force Post transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
Customer ID	String	50-character alpha-numeric	'cust_id'=>$cust_id
Dynamic descriptor	String	20-character alpha-numeric	'dynamic_descriptor'=>$dynamic_descriptor

Sample Encrypted Mag Swipe Force Post

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$orderid="ord_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="ENCRYPTEDTRACK2DATA";
$pos_code="00";
$device_type='idtech_bdk';
$auth_code='123456';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'enc_track2_forcepost',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
enc_track2=>$enc_track2,
pos_code=>$pos_code,
device_type=>$device_type,
auth_code=>$auth_code,
dynamic_descriptor=>'12345'
);
```

Sample Encrypted Mag Swipe Force Post

```
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

6.6 Mag Swipe Purchase Correction

Mag Swipe Purchase Correction transaction object definition

```
$txnArray = array(‘type’=>’track2_purchasecorrection', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Purchase Correction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Purchase Correction transaction values

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

Table 65: Mag Swipe Purchase Correction transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`

Table 66: Mag Swipe Purchase Correction transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Mag Swipe Purchase Correction

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables ******************************/
$orderid='ord-110515-15:27:18';
$txnnumber='31999-0_10';
$dynamic_descriptor='nqa';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_purchasecorrection',
order_id=>$orderid,
txn_number=>$txnnumber,
dynamic_descriptor=>$dynamic_descriptor
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
```

Sample Mag Swipe Purchase Correction

```
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

6.7 Mag Swipe Refund

Mag Swipe Refund transaction object definition

```
$txnArray = array(‘type’=>’track2_refund', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Refund transaction values

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

Table 67: Mag Swipe Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`

Table 68: Mag Swipe Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	'cust_id'=>$cust_id
Status Check	Boolean	true/false	$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
Dynamic descriptor	String	20-character alpha-numeric	'dynamic_descriptor'=>$dynamic_descriptor

Sample Mag Swipe Refund

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables ******************************/
$orderid='ord-110515-15:44:10';
$amount='1.00';
$txnnumber='32087-1_10';
$dynamic_descriptor='nqa';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_refund',
order_id=>$orderid,
amount=>$amount,
txn_number=>$txnnumber,
dynamic_descriptor=>$dynamic_descriptor
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
```

Sample Mag Swipe Refund

```
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’track2_ind_refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Mag Swipe Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Mag Swipe Independent Refund transaction values

Table 69: Mag Swipe Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`

Table 69: Mag Swipe Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Track2 data	String	40-character numeric	`track2=>$track`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
POS code	String	2-character numeric	`'pos_code'=>$pos_code`

Table 70: Mag Swipe Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample Mag Swipe Independent Refund

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
//$status = 'false';
/************************ Transaction Variables ******************************/
$orderid='ord-'.date("dmy-G:i:s");
$custid='cust id';
$amount='1.00';
/*************** Swipe Card and read Track1 and/or Track2 *******************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);
$startDelim = ";";
$firstChar = $track1{0};
$track = '';
if($firstChar==$startDelim)
{
$track = $track1;
}
else
{
$track2 = fgets ($stdin);
```

Sample Mag Swipe Independent Refund

```
$track = $track2;
}
$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'track2_ind_refund',
order_id=>$orderid,
cust_id=>$custid,
amount=>$amount,
track2=>$track,
pan=>'',
expdate=>'',
pos_code=>'00',
dynamic_descriptor=>'nqa'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’enc_track2_ind_refund', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Encrypted Mag Swipe Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Encrypted Track 2 data	String	n/a	`'enc_track2'=>$enc_track2`
Device Type	String	30-character alpha-numeric	`'device_type'=>$device_type`
POS Code	String	2-character numeric	`'pos_code'=>$pos_code`

Table 2 Encrypted Mag Swipe Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`

Sample Encrypted Mag Swipe Independent Refund

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$orderid="ord_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="ENCRYPTEDTRACK2DATA";
$pos_code="00";
$device_type='idtech_bdk';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'enc_track2_ind_refund',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
enc_track2=>$enc_track2,
pos_code=>$pos_code,
device_type=>$device_type,
dynamic_descriptor=>'12345'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

7 Level 2/3 Transactions

- 7.1 About Level 2/3 Transactions
- 7.2 Level 2/3 Visa Transactions
- 7.3 Level 2/3 MasterCard Transactions
- 7.4 Level 2/3 American Express 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
- 7.2.2 Level 2/3 Transaction Flow for Visa
- 7.2.3 VS Completion
- 7.2.5 VS Force Post
- 7.2.4 VS Purchase Correction
- 7.2.6 VS Refund
- 7.2.7 VS Independent Refund
- 7.2.8 VS Corpais

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.

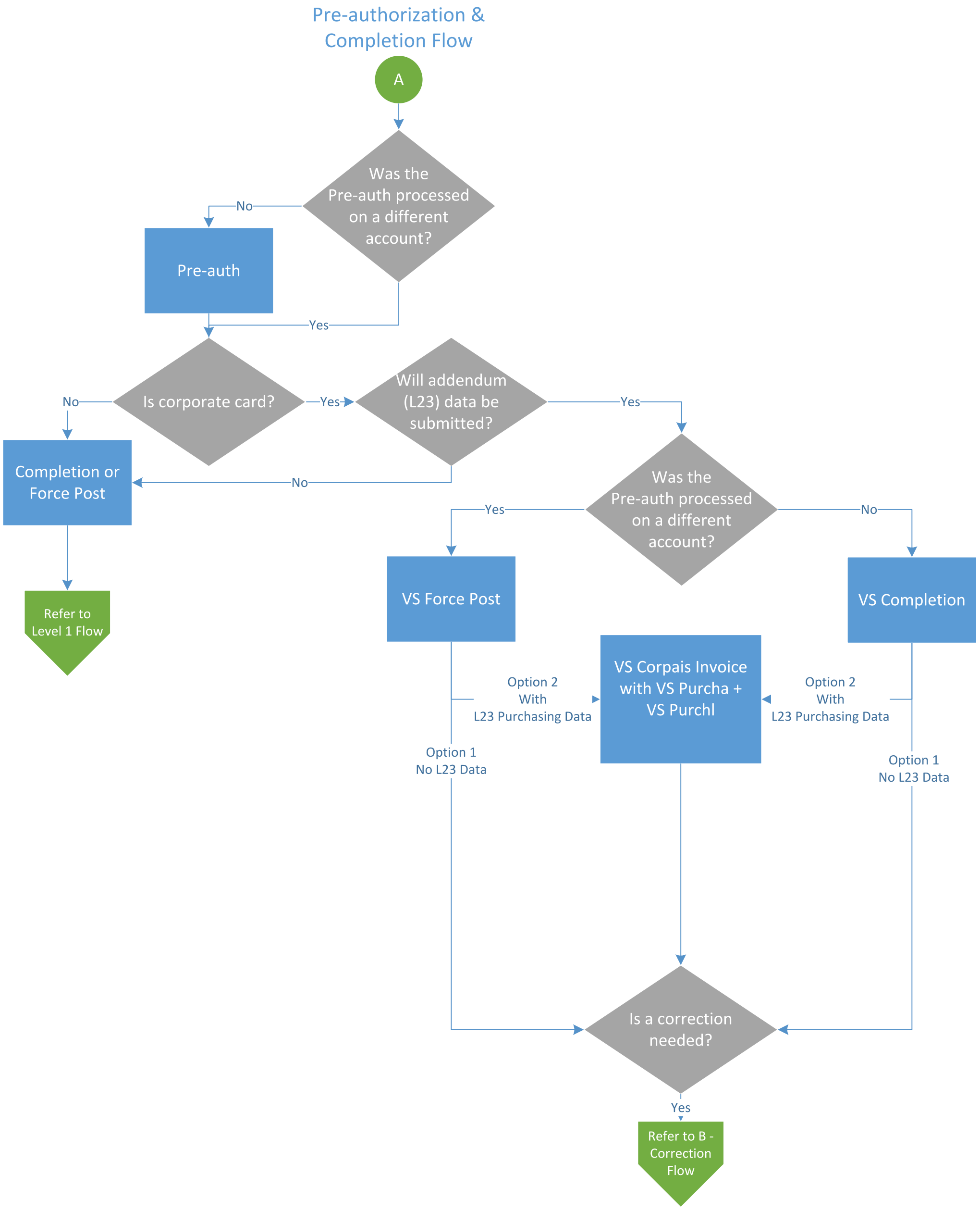
VS Corpais – (Level 2/3 Data)

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.

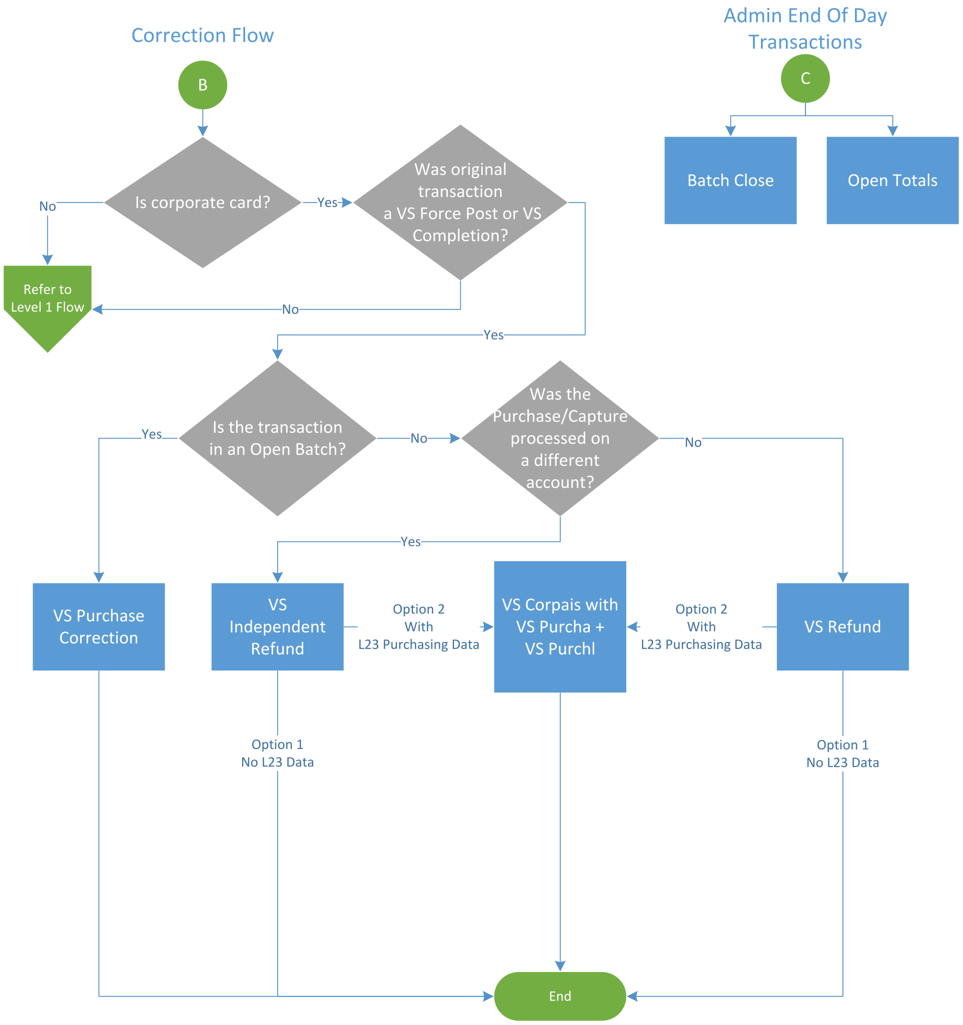
* A VS Purchase Correction can be performed against a transaction as long as the batch that contains the original transaction remains open. When using the automated closing feature, the batch close occurs daily between 10 – 11 pm EST.

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?
Yes
No
Was original transaction a VS Force Post or VS Completion?
Refer to Level 1 Flow
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

```
$txnArray = array(‘type’=>’vscompletion', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Completion transaction object

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VS Completion transaction object values

Table 1 VS Completion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Completion amount	String	(missing or bad snippet)	`'comp_amount'=>$comp_amount`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
E-Commerce Indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

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

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`'national_tax'=>$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	`'merchant_vat_no'=>$merchant_vat_no`	Merchant's Tax Registration Number must be provided if tax is included on the invoice **NOTE:** Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`'local_tax'=>$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 = 0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST)	15-character alpha-	`'local_tax_no'=>$local_tax_no`	Merchant's

Req*	Value	Limits	Set Method	Description
	Registration Number	numeric		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	`'customer_vat_no'=>$customer_vat_no`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	`'cri'=>$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	`'customer_code'=>$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	`'invoice_number'=>$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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='vscompletion';
$order_id='ord-210916-15:14:46';
$comp_amount='5.00';
$txn_number = '19002-0_11';
$crypt='7';
$national_tax = "1.23";
$merchant_vat_no = "gstno111";
$local_tax = "2.34";
$customer_vat_no = "gstno999";
$cri = "CUST-REF-002";
$customer_code="ccvsfp";
$invoice_number="invsfp";
$local_tax_no="ltaxno";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'comp_amount'=>$comp_amount,
'txn_number'=>$txn_number,
'crypt_type'=>$crypt,
'national_tax'=>$national_tax,
'merchant_vat_no'=>$merchant_vat_no,
'local_tax'=>$local_tax,
'customer_vat_no'=>$customer_vat_no,
'cri'=>$cri,
'local_tax_no'=>$local_tax_no
);
/**************************** Transaction Object *****************************/
```

Sample VS Completion

```
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’vspurchasecorrection', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Purchase Correction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`
E-Commerce Indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Sample VS Purchase Correction

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables *****************************/
$type='vspurchasecorrection';
$order_id='ord-210916-15:28:01';
$amount='5.00';
$txn_number = '19017-0_11';
$crypt='7';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'txn_number'=>$txn_number,
'crypt_type'=>$crypt
);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
```

Sample VS Purchase Correction

```
    print("\nMessage = " . $mpgResponse->getMessage());
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    ?>
```

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

```
$txnArray = array(‘type’=>’vsforcepost', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Force Post transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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 alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`

Value	Type	Limits	Set Method
Expiry Date	String	4-character numeric YYMM format	`'expdate'=>$expiry_date`
Authorization code	String	8-character alpha-numeric	`'auth_code'=>$auth_code`
E-commerce Indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 2 VS Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$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	`'national_tax'=>$national_tax`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`'merchant_vat_no'=>$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

Req*	Value	Limits	Set Method	Description
				zeroes
C	Local Tax	12-character decimal	`'local_tax'=>$local_tax`	Must reflect the amount of Local Tax (PST or QST) appearing on the invoice If Local Tax included then must not be all spaces or all zeroes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alphanumeric	`'local_tax_no'=>$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

Req*	Value	Limits	Set Method	Description
				or QST) applies
C	Customer VAT Registration Number	13-character alphanumeric	`'customer_vat_no'=>$customer_vat_no`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	`'cri'=>$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	`'customer_code'=>$customer_code`	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alphanumeric	`'invoice_number'=>$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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='vsforcepost';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='5.00';
$pan='4242424254545454';
$expiry_date='2012';
$auth_code='123456';
$crypt='7';
$national_tax = "1.23";
$merchant_vat_no = "gstno111";
$local_tax = "2.34";
$customer_vat_no = "gstno999";
$cri = "CUST-REF-002";
$customerCode="ccvsfp";
$invoiceNumber="invsfp";
$local_tax_no="ltaxno";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'auth_code'=>$auth_code,
'crypt_type'=>$crypt,
'national_tax'=>$national_tax,
'merchant_vat_no'=>$merchant_vat_no,
'local_tax'=>$local_tax,
'customer_vat_no'=>$customer_vat_no,
'cri'=>$cri,
'local_tax_no'=>$local_tax_no
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
```

Sample VS Force Post

```
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    ?>
```

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

```
$txnArray = array(‘type’=>’vsrefund’, …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
E-Commerce Indicator	String	1-character alpha-	`'crypt_type'=>$crypt`

Value	Type	Limits	Set Method
		numeric	

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

Req*	Value	Limits	Set Method	Description
Y	National Tax	12-character decimal	`'national_tax'=>$national_tax`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference	20-character alpha-numeric	`'merchant_vat_no'=>$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	`'local_tax'=>$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

Req*	Value	Limits	Set Method	Description

				spaces or all zeroes; 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 alphanumeric	`'local_tax_no'=>$local_tax_no`	Merchant's Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alphanumeric	`'customer_vat_no'=>$customer_vat_no`	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here

Req*	Value	Limits	Set Method	Description
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	`'cri'=>$cri`	Value which the customer may choose to provide to the supplier at the point of sale – must be provided if given by the customer
N	Customer Code	17-character alphanumeric	`'customer_code'=>$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	`'invoice_number'=>$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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables *****************************/
$type='vsrefund';
$order_id='ord-210916-15:14:46';
$amount='5.00';
$txn_number = '19003-1_11';
$crypt='7';
$national_tax = "1.23";
$merchant_vat_no = "gstno111";
$local_tax = "2.34";
```

Sample VS Refund

```
$customer_vat_no = "gstno999";
$cri = "CUST-REF-002";
$customerCode="ccvsfp";
$invoiceNumber="invsfp";
$local_tax_no="ltaxno";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'txn_number'=>$txn_number,
'crypt_type'=>$crypt,
'national_tax'=>$national_tax,
'merchant_vat_no'=>$merchant_vat_no,
'local_tax'=>$local_tax,
'customer_vat_no'=>$customer_vat_no,
'cri'=>$cri,
'local_tax_no'=>$local_tax_no
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’vsind_refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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 alphanumeric	'order_id'=>$order_id
Amount	String	(missing or bad snippet)	'amount'=>$amount
Credit card number	String	20-character numeric	'pan'=>$pan
Expiry date	String	4-character numeric YYMM format	'expdate'=>$expiry_date
E-commerce indicator	String	1-character alphanumeric	'crypt_type'=>$crypt

Table 2 VS Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	'cust_id'=>$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	'national_tax'=>$national_	Must reflect

Req*	Value	Limits	Set Method	Description
			`tax`	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	`'merchant_vat_no'=>$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	`'local_tax'=>$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 = 0.01
				Maximum = 999999.99

Req*	Value	Limits	Set Method	Description
				Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alphanumeric	`'local_tax_no'=>$local_tax_no`	Merchant's Local Tax (PST/QST) Registration Number Must be provided if tax is included on the invoice; If Local Tax included then must not be all spaces or all zeroes Must be provided if Local Tax (PST or QST) applies
C	Customer VAT Registration Number	13-character alphanumeric	`'customer_vat_no'=>$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	`'cri'=>$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	`'customer_code'=>$customer_code`	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alpha-numeric	`'invoice_number'=>$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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='vsind_refund';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='5.00';
$pan='4242424254545454';
$expiry_date='2012';
$crypt='7';
$national_tax = "1.23";
$merchant_vat_no = "gstno111";
$local_tax = "2.34";
$customer_vat_no = "gstno999";
$cri = "CUST-REF-002";
$customerCode="ccvsfp";
$invoiceNumber="invsfp";
$local_tax_no="ltaxno";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
```

Sample VS Independent Refund

```
'crypt_type'=>$crypt,
'national_tax'=>$national_tax,
'merchant_vat_no'=>$merchant_vat_no,
'local_tax'=>$local_tax,
'customer_vat_no'=>$customer_vat_no,
'cri'=>$cri,
'local_tax_no'=>$local_tax_no
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’vscorpais', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VS Corpais transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`

vsPurcha
For a list of the variables that appear in this object, see the table below
	Object	n/a	`$vsPurcha = new vsPurcha();`
			`$mpgVsLevel23 = new mpgVsLevel23();`
			`$mpgVsLevel23->setVsPurch ($vsPurcha, $vsPurchl);`

vsPurchl
For a list of the variables that appear in this object, see the table below
	Object	n/a	`$vsPurchl = new vsPurchl();`
			`$mpgVsLevel23 = new mpgVsLevel23();`
			`$mpgVsLevel23->setVsPurch ($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

Req*	Value	Limits	Set Method	Description
				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 invoice 1 - if Tax was calculated on Post-Discount totals

Req*	Value	Limits	Set Method	Description
				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 addresses,requires full alpha postal code for the merchant with Valid ANA<space>NAN

Req*	Value	Limits	Set Method	Description
				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; 4 = No tax applies (small merchant) on the invoice for the transaction

Req*	Value	Limits	Set Method	Description
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	`$vsPurcha->setGstHstFreightAmount($gst_hst_freight_amount);`	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	`$vsPurcha->setGstHstFreightRate($gst_hst_freight_rate);`	Amount of GST (excludes PST) or HST charged on the shipping amount If Freight/Shipping Amount is

Req*	Value	Limits	Set Method	Description
				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

```
$item_com_code = array("X3101", "X84802");

$product_code = array("CHR123", "DDSK200");

$item_description = array("Office Chair", "Disk Drive");

$item_quantity = array("3", "1");

$item_uom = array("EA", "EA");

$unit_cost = array("0.20", "0.40");

$vat_tax_amt = array("0.00", "0.00");

$vat_tax_rate = array("13.00", "13.00");

$discount_treatmentL = array("0", "0");

$discount_amtL = array("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]);
```

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 Comodity Code (if this field is not sent,

Req*	Value	Limits	Variable/Field	Description
				then Product Code must be sent)
Y	Product Code	12-character alpha-numeric	product_code	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 every line item
Y	Item Description	35-character alpha-numeric	item_description	Line item description

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 Codes
Y	Item Unit Cost	12-character decimal	unit_cost	Line item cost per unit

Req*	Value	Limits	Variable/Field	Description

				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 Treatment	1-character numeric	discount_treatmentL	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
C	Discount Amount	12-character decimal	discount_amtL	Amount of discount, if provided for this line item according to the Line Item Discount Treatment Must be non-zero if Line Item Discount Treatment is

Req*	Value	Limits	Variable/Field	Description

				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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='vscorpais';
$cust_id='CUST13343';
$order_id='ord-160916-15:31:39';
$txn_number='18306-0_11';
$buyer_name = "Buyer Manager";
$local_tax_rate = "13.00";
$duty_amount = "0.00";
$discount_treatment = "0";
$discount_amt = "0.00";
$freight_amount = "0.20";
$ship_to_pos_code = "M8X 2W8";
$ship_from_pos_code = "M1K 2Y7";
$des_cou_code = "CAN";
$vat_ref_num = "VAT12345";
$tax_treatment = "3";//3 = Gross prices given with tax information provided at invoice level
$gst_hst_freight_amount = "0.00";
$gst_hst_freight_rate = "13.00";
$item_com_code = array("X3101", "X84802");
$product_code = array("CHR123", "DDSK200");
$item_description = array("Office Chair", "Disk Drive");
$item_quantity = array("3", "1");
$item_uom = array("EA", "EA");
$unit_cost = array("0.20", "0.40");
$vat_tax_amt = array("0.00", "0.00");
$vat_tax_rate = array("13.00", "13.00");
$discount_treatmentL = array("0", "0");
$discount_amtL = array("0.00", "0.00");
//Create and set 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);
```

Sample VS Corpais

```
$vsPurcha->setFreightAmount($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 = 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]);
//Create and set VsLevel23
$mpgVsLevel23 = new mpgVsLevel23();
$mpgVsLevel23->setVsPurch($vsPurcha, $vsPurchl);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=>$txn_number,
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgVsLevel23);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

7.3 Level 2/3 MasterCard Transactions

- 7.3.1 Level 2/3 Transaction Types for MasterCard
- 7.3.2 Level 2/3 Transaction Flow for MasterCard
- 7.3.3 MC Completion
- 7.3.4 MC Force Post
- 7.3.5 MC Purchase Correction
- 7.3.6 MC Refund
- 7.3.7 MC Independent Refund
- 7.3.8 MC Corpais - Corporate Card Common Data with Line Item Details

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.

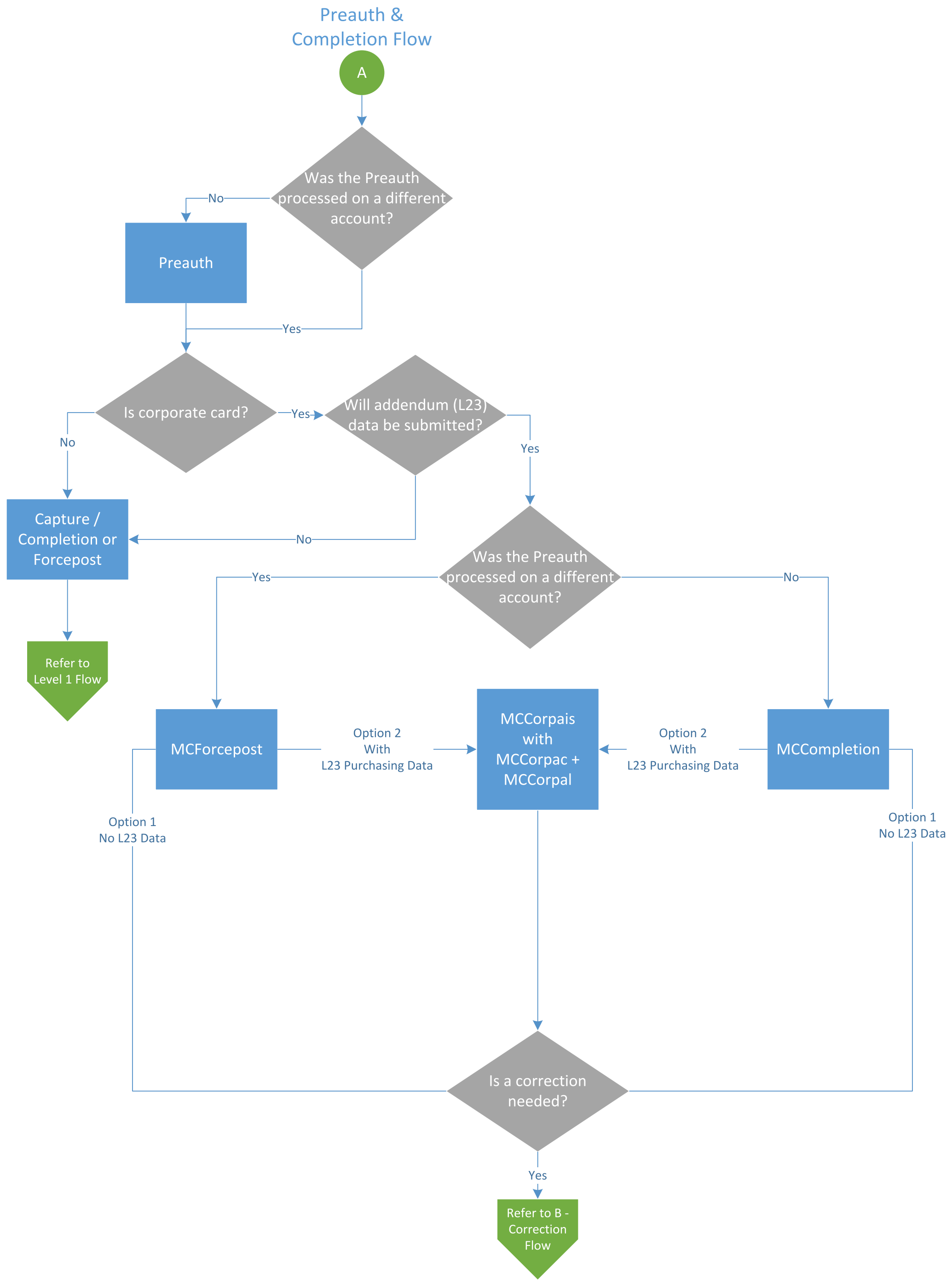
MC Corpais Common Line Item – (Level 2/3 Data)

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:

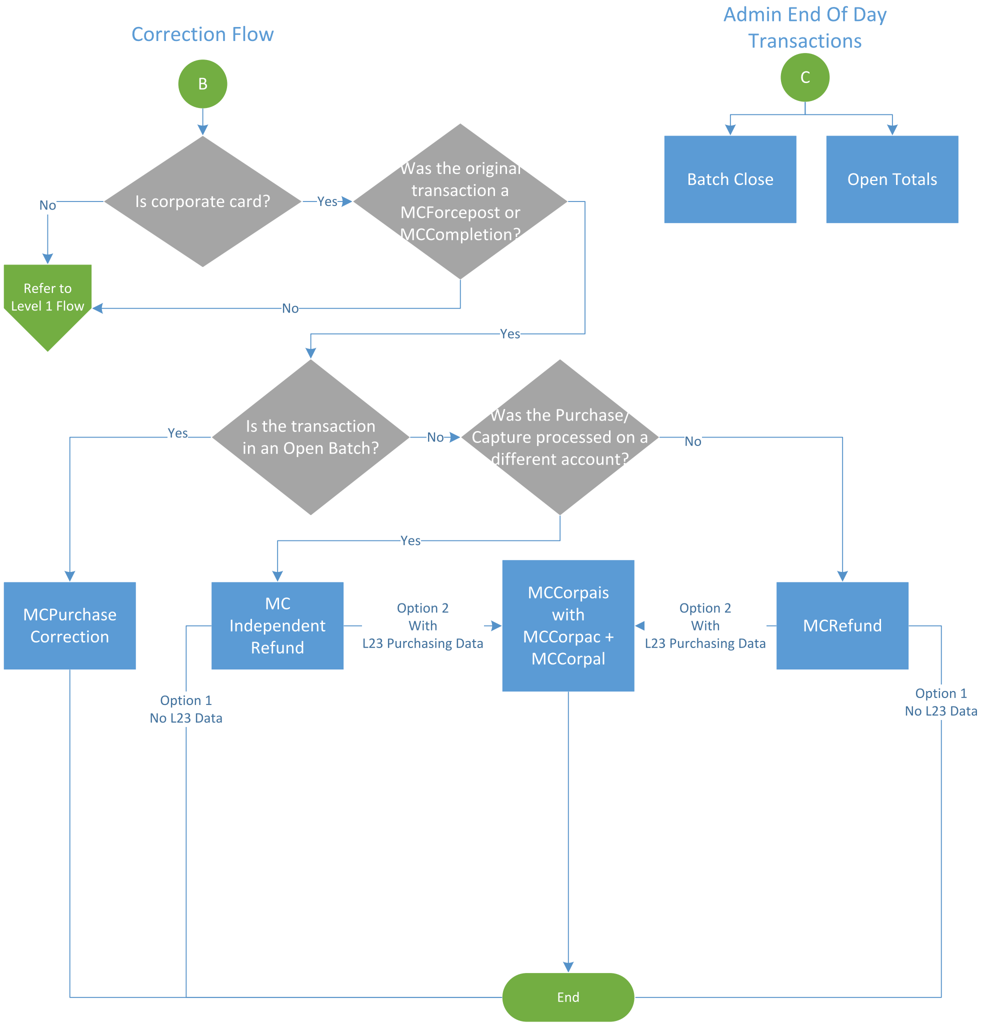
- Purchasing Card Data:
  - Corporate card common data with Line Item Details

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
Was the Preauth processed on a different account?
Yes
No
Refer to Level 1 Flow
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

```
$txnArray = array(‘type’=>’mccompletion', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Completion transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Completion amount	String	(missing or bad snippet)	`'comp_amount'=>$comp_amount`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`
Merchant reference number	String	19-character alpha-numeric	`'merchant_ref_no'=>$merchant_ref_no`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Sample MC Completion

```
<?php
require "../../mpgClasses.php";
```

Sample MC Completion

```
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='mccompletion';
$order_id='ord-210916-16:13:11';
$comp_amount='5.00';
$txn_number='19021-0_11';
$crypt='7';
$merchant_ref_no = "319038";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'comp_amount'=>$comp_amount,
'txn_number'=>$txn_number,
'merchant_ref_no' => $merchant_ref_no,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array('type'=>'mcforcepost', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Force Post transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Credit card number	String	20-character alpha-numeric	`'pan'=>$pan`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
Authorization code	String	8-character alpha-numeric	`'auth_code'=>$auth_code`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Merchant reference number	String	19-character alpha-numeric	`'merchant_ref_no'=>$merchant_ref_no`

Table 2 MC Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`

Sample MC Force Post

```php
<?php
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables *****************************/
$type='mcforcepost';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='5.00';
$pan='5454545442424242';
$expiry_date='2012';
$auth_code='123456';
$crypt='7';
$merchant_ref_no = "319038";
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'auth_code'=>$auth_code,
'merchant_ref_no' => $merchant_ref_no,
'crypt_type'=>$crypt
);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’mcpurchasecorrection', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Purchase Correction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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 alphanumeric	`'order_id'=>$order_id`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Sample MC Purchase Correction

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='mcpurchasecorrection';
$order_id='ord-210916-16:15:50';
$txn_number='660117316420162651615509 29-0_11';
$crypt='7';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
```

Sample MC Purchase Correction

```
'txn_number'=>$txn_number,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’mcrefund', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Merchant reference number	String	19-character alpha-numeric	`'merchant_ref_no'=>$merchant_ref_no`

Sample MC Refund

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='mcrefund';
$order_id='ord-210916-16:13:11';
$amount='5.00';
$txn_number='19021-1_11';
$crypt='7';
$merchant_ref_no = "319038";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'txn_number'=>$txn_number,
'merchant_ref_no' => $merchant_ref_no,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Sample MC Refund

```
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/****************************** Response ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’mcind_refund', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric (YYMM format)	`'expdate'=>$expiry_date`
Merchant reference number	String	19-character alphanumeric	`'merchant_ref_no'=>$merchant_ref_no`

Table 2 MC Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`

Sample MC Independent Refund

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='mcind_refund';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
```

Sample MC Independent Refund

```
$amount='5.00';
$pan='5454545442424242';
$expiry_date='2012';
$crypt='7';
$merchant_ref_no = "319038";
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'merchant_ref_no' => $merchant_ref_no,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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 419).

MC Corpais transaction object definition

```
$txnArray = array(‘type’=>’mccorpais', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MC Corpais transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

MC Corpais transaction object values

Table 1 MC Corpais transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`

MCCorpac	Object	n/a	`$mpgMcLevel23 = new mpgMcLevel23();`
			`$mpgMcLevel23->setMcCorpac ($mcCorpac);`

MC Corpal	Object	n/a	`$mpgMcLevel23 = new mpgMcLevel23();`

Value	Type	Limits	Set Method
			`$mpgMcLevel23->setMcCorpal ($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 alphanumeric	`$mcCorpac->setAustinTetraNumber ($austin_tetra_number);`	The Austin-Tetra Number assigned to the card acceptor
N	NAICS Code	15-character alphanumeric	`$mcCorpac->setNaicsCode ($naics_code);`	North American Industry Classification System (NAICS) code assigned to the card acceptor
N	Customer Code	25-character alphanumeric	`$mcCorpac->setCustomerCode1 ($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 Number	17-character alphanumeric	`$mcCorpac->setUniqueInvoiceNumber ($unique_invoice_number_c);`	Unique number associated with the individual transaction 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	`$mcCorpac->setOrderDate ($order_date_c);`	The date the item was ordered

Req*	Value	Limits	Set Method	Description
		format		**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	`$vsPurcha->setDutyAmount($duty_amount);`	The duty on the total purchase
				Must have 2 decimals
				Minimum = 0.00
				Maximum = 999999.99

N	Destination State / Province Code	3-character alphanumeric	`$mcCorpac->setDestinationProvinceCode($destination_province_code);`	State or Province of the country where the goods will be delivered Left justified with trailing spaces **EXAMPLE:** ONT = Ontario
N	Destination Country Code	3-character alphanumeric	`$mcCorpac->setDestinationCountryCode($destination_country_code);`	The country code where goods will be delivered Left justified with trailing spaces

Req*	Value	Limits	Set Method	Description
		ISO 3166-1 alpha-3 format		ISO 3166-1 alpha-3 format **EXAMPLE:** CAN = Canada
N	Ship From Postal Code	10-character alphanumeric 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 alphanumeric	`$mcCorpac->setShipToPosCode($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 alphanumeric	`$mcCorpac->setAuthorizedContactName($authorized_contact_name_c);`	Name of an individual or company contacted for company authorized purchases
N	Authorized Contact Phone	17-character alphanumeric	`$mcCorpac->setAuthorizedContactPhone($authorized_contact_phone);`	Phone number of an individual or company contacted for company authorized purchases
N	Additional Card Acceptor Data	40-character alphanumeric	`$mcCorpac->setAdditionalCardAcceptorData($additional_card_acceptor_data);`	Information pertaining to the card acceptor
N	Card Acceptor Type	8-character alphanumeric	`$mcCorpac->setCardAcceptorType($card_acceptor_type);`	Various classifications of business ownership characteristics This field takes 8 characters. Each character represents a different component, as follows:

Req*	Value	Limits	Set Method	Description
				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 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 busi-

Req*	Value	Limits	Set Method	Description
				ness 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 business and other government or agency-recognized certification 8 - Other government or agency-recognized certification and self-certified small business A - SBA certification

Req*	Value	Limits	Set Method	Description
				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' 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

Req*	Value	Limits	Set Method	Description
				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 racial/ethnic type was not provided. R - Card acceptor refused to provide business racial/ethnic type
N	Card Acceptor	20-character alpha-	`$mcCorpac->setCardAcceptorTaxTd`	US federal tax ID number or value-added tax (VAT) ID

Req*	Value	Limits	Set Method	Description
	Tax ID	numeric	`($card_acceptor_tax_id_c);`	
N	Card Acceptor Reference Number	25-character alphanumeric	`$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($mcTax_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], $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], $mcTax_l[0]);
```

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

Req*	Value	Limits	Variable	Description
N	Customer Code	25-character alphanumeric	`customer_code1_l`	A control number, such as purchase order number, project number, department allocation number or name that the pur-

Req*	Value	Limits	Variable	Description
				chaser 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 alphanumeric	`product_code1_ll`	Line item Product Code Contains the non-fuel related product code of the individual item purchased
Y	Item Description	35-character alphanumeric	`item_description_ll`	Line Item description Contains the description of the individual item pur-

Req*	Value	Limits	Variable	Description
				chased
Y	Item Quantity	12-character alphanumeric	`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 alphanumeric	`item_unit_measure_ll`	The line item unit of measurement code ANSI X-12 EDI Allowable Units of Measure and Codes
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

Req*	Value	Limits	Variable	Description
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

$tax_amount_c = array("1.19", "1.29");

$tax_rate_c = array("6.0", "7.0");

$tax_type_c = array("GST", "PST");

$tax_id_c = array("gst1298", "pst1298");
```

```
$tax_included_in_sales_c = array("Y", "N");

//Create and set Tax for McCorpac

$mcTax_c = new mcTax();

$mcTax_c->setTax($tax_amount_c[0], $tax_rate_c[0], $tax_type_c[0], $tax_id_c
[0], $tax_included_in_sales_c[0]);

$mcTax_c->setTax($tax_amount_c[1], $tax_rate_c[1], $tax_type_c[1], $tax_id_c
[1], $tax_included_in_sales_c[1]);
```

Setting tax array for MC Corpal

```
//Tax Details for Items

$tax_amount_l = array("0.52", "1.48");

$tax_rate_l = array("13.0", "13.0");

$tax_type_l = array("HST", "HST");

$tax_id_l = array("hst1298", "hst1298");

$tax_included_in_sales_l = array("Y", "Y");

//Create and set Tax for McCorpal

$mcTax_l = array(new mcTax(), new mcTax());

$mcTax_l[0]->setTax($tax_amount_l[0], $tax_rate_l[0], $tax_type_l[0], $tax_id_
l[0], $tax_included_in_sales_l[0]);

$mcTax_l[1]->setTax($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 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

Req*	Value	Limits	Variable	Description
				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

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
```

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

```
/************************ Transactional Variables ***************************/
$type='mccorpais';
$cust_id='CUST13343';
$order_id='ord-200916-13:29:27';
$txn_number='66011731632016264132927986-0_11';
$customer_code1_c ="CustomerCode123";
$card_acceptor_tax_id_c ="UrTaxId";//Merchant tax id which is mandatory
$corporation_vat_number_c ="cvn123";
$freight_amount_c ="1.23";
$duty_amount_c ="2.34";
$ship_to_pos_code_c ="M1R 1W5";
$order_date_c ="141211";
$customer_vat_number_c ="customervn231";
$unique_invoice_number_c ="uin567";
$authorized_contact_name_c ="John Walker";
//Tax Details
$tax_amount_c = array("1.19", "1.29");
$tax_rate_c = array("6.0", "7.0");
$tax_type_c = array("GST", "PST");
$tax_id_c = array("gst1298", "pst1298");
$tax_included_in_sales_c = array("Y", "N");
//Item Details
$customer_code1_l = array("customer code", "customer code2");
$line_item_date_l = array("150114", "150114");
$ship_date_l = array("150120", "150122");
$order_date1_l = array("150114", "150114");
$medical_services_ship_to_health_industry_number_l = array(null, null);
$contract_number_l = array(null, null);
$medical_services_adjustment_l = array(null, null);
$medical_services_product_number_qualifier_l = array(null, null);
$product_code1_l = array("pc11", "pc12");
$item_description_l = array("Good item", "Better item");
$item_quantity_l = array("4", "5");
$unit_cost_l =array("1.25", "10.00");
$item_unit_measure_l = array("EA", "EA");
$ext_item_amount_l =array("5.00", "50.00");
$discount_amount_l =array("1.00", "50.00");
$commodity_code_l =array("cCode11", "cCode12");
$type_of_supply_l = array(null, null);
$vat_ref_num_l = array(null, null);
//Tax Details for Items
$tax_amount_l = array("0.52", "1.48");
$tax_rate_l = array("13.0", "13.0");
$tax_type_l = array("HST", "HST");
$tax_id_l = array("hst1298", "hst1298");
$tax_included_in_sales_l = array("Y", "Y");
//Create and set Tax for McCorpac
$mcTax_c = new mcTax();
$mcTax_c->setTax($tax_amount_c[0], $tax_rate_c[0], $tax_type_c[0], $tax_id_c[0], $tax_included_in_
sales_c[0]);
$mcTax_c->setTax($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 = 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);
```

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

```
$mcCorpac->setShipToPosCode($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);
$mcCorpac->setTax($mcTax_c);
//Create and set Tax for McCorpal
$mcTax_l = array(new mcTax(), new mcTax());
$mcTax_l[0]->setTax($tax_amount_l[0], $tax_rate_l[0], $tax_type_l[0], $tax_id_l[0], $tax_included_
in_sales_l[0]);
$mcTax_l[1]->setTax($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 = 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], $mcTax_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], $mcTax_l[1]);
//Create and set McLevel23
$mpgMcLevel23 = new mpgMcLevel23();
$mpgMcLevel23->setMcCorpac($mcCorpac);
$mpgMcLevel23->setMcCorpal($mcCorpal);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=>$txn_number,
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgMcLevel23);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
```

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

```
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

7.4 Level 2/3 American Express Transactions

- 7.4.1 Level 2/3 Transaction Types for Amex
- 7.4.2 Level 2/3 Transaction Flow for Amex
- 7.4.4 AX Completion
- 7.4.5 AX Force Post
- 7.4.6 AX Purchase Correction
- 7.4.7 AX Refund
- 7.4.8 AX Independent Refund

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.

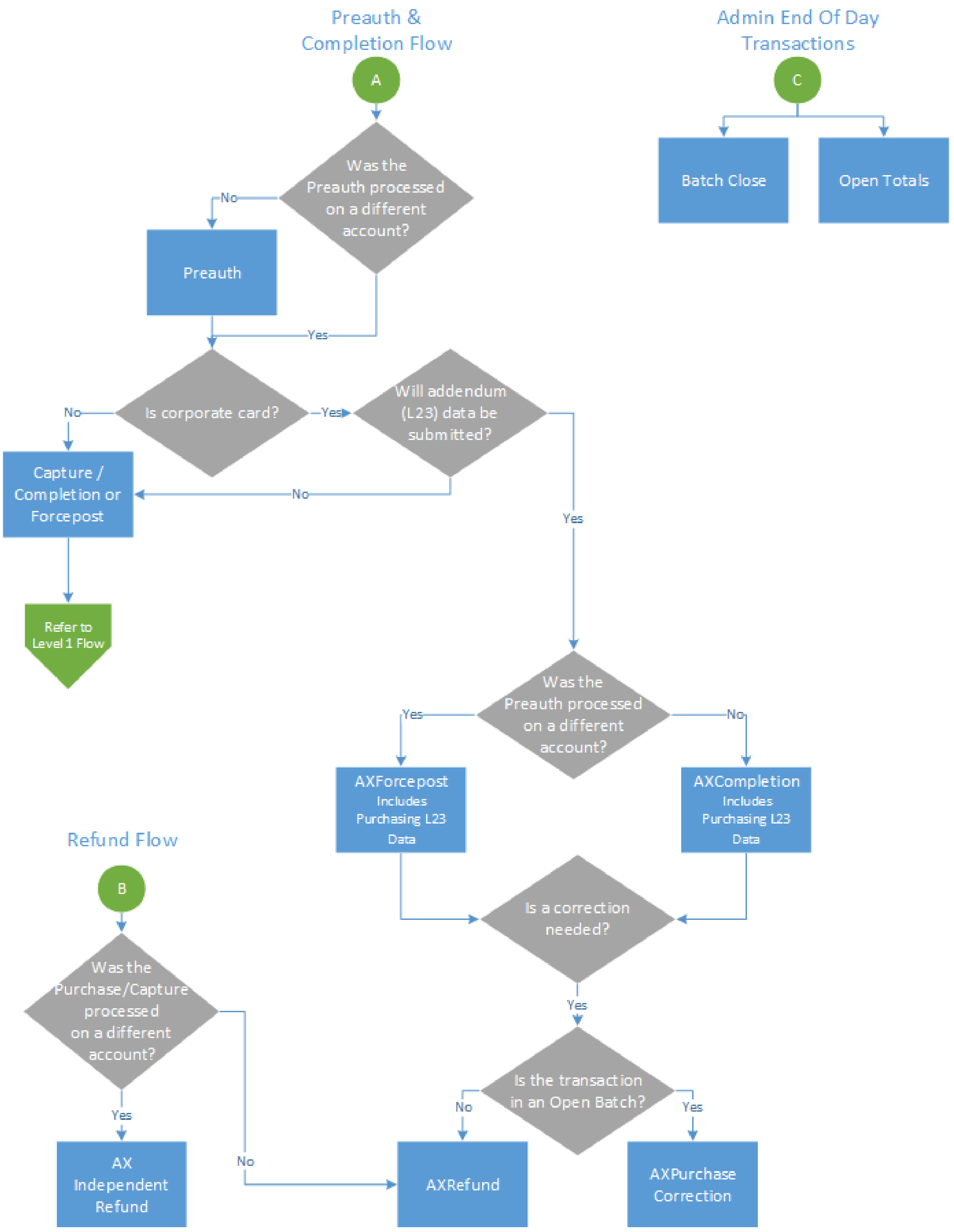
AX Refund – (Credit)

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?
No
Capture / Completion or Forcepost
Refer to Level 1 Flow
Yes
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
AXRefund
AXPurchase Correction
Admin End Of Day Transactions
C
Batch Close
Open Totals
Refund Flow
B
Was the Purchase/Capture processed on a different account?
Yes
No
AX Independent Refund

7.4.3 Level 2/3 Data Objects in Amex

- 7.4.3.1 About the Level 2/3 Data Objects for Amex
- 7.4.3.2 Defining the AxLevel23 Object
    - Table 1 Object
    - Table 2 Object
    - Table 3 Object

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

```
$mpgAxLevel23 = new mpgAxLevel23();
```

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	`$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);`	Refer below for further breakdown and definition of table1

Req*	Value	Limits	Set Method	Description
Y	Table2	Object	`$mpgAxLevel23->setTable2($axItLoop);`	Refer below for further breakdown and definition of table2
Y	Table3	Object	`$mpgAxLevel23->setTable3($taxTbl3);`	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

```
$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);
```

Table 1 AxLevel23 object - Table 1 object fields

Req*	Value	Limits	Set Method	Description
C	Purchase Order Number	22-character alphanumeric	`'big04'=>$big04`	The cardholder supplied Purchase Order Number, which is entered by the merchant at the point-of-sale This entry is used in the Statement/Reporting process and may include accounting information specific to the client **NOTE:** This element is mandatory, if the merchant's customer provides a Purchase Order Number.
N	Release Number	30-character alphanumeric	`'big05'=>$big05`	A number that identifies a release

Req*	Value	Limits	Set Method	Description
				against a Purchase Order previously placed by the parties involved in the transaction
N	Invoice Number	8-character alphanumeric	`'big10'=>$big10`	Contains the Amex invoice/reference number
N	N1Loop	Object	`'n1Loop'=>$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

```
$axN1Loop = new axN1Loop();
```

```
$axN1Loop->setN1Loop($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 alphanumeric	`n101`	Supported values: R6 - Requester (required) BG - Buying Group (optional) SF - Ship From (optional) ST - Ship To (optional)

<table>
<tr><th>Req*</th><th>Value</th><th>Limits</th><th>Variable or Set Method</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>40 - Receiver (optional)</td></tr>
<tr><td>Y</td><td>Name</td><td>40-character alphanumeric</td><td><code>n102</code></td><td><table><tr><th>n101 code</th><th>n102 meaning</th></tr><tr><td>R6</td><td>Requester Name</td></tr><tr><td>BG</td><td>Buying Group Name</td></tr><tr><td>SF</td><td>Ship From Name</td></tr><tr><td>ST</td><td>Ship To Name</td></tr><tr><td>40</td><td>Receiver Name</td></tr></table></td></tr>
<tr><td>N</td><td>Address</td><td>40-character alphanumeric</td><td><code>n301</code></td><td>Address</td></tr>
<tr><td>N</td><td>City</td><td>30-character alphanumeric</td><td><code>n401</code></td><td>City</td></tr>
<tr><td>N</td><td>State or Province</td><td>2-character alphanumeric</td><td><code>n402</code></td><td>State or province</td></tr>
<tr><td>N</td><td>Postal Code</td><td>15-character alphanumeric</td><td><code>n403</code></td><td>Postal Code</td></tr>
<tr><td>N</td><td>AxRef</td><td>Object</td><td><code>$axRef1 = new axRef ();</code></td><td>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</td></tr>
</table>

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

Table 1 - Setting the AxRef Object

Setting AXRef object

```
$axRef1 = new axRef();

$ref01 = array("4C", "CR"); //Reference ID Qualifier

$ref02 = array("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

<table>
<tr><th>Req*</th><th>Value</th><th>Limits</th><th>Variable</th><th>Description</th></tr>
<tr><td>Y</td><td>Reference Identification Qualifier</td><td>2-character alphanumeric</td><td>ref01</td><td>This element may contain the following qualifiers for the corresponding occurrences of the N1Loop:
<table>
<tr><th>n101 value</th><th>ref01 denotation</th></tr>
<tr><td>R6</td><td>Supported values:
4C - Shipment Destination Code (mandatory)
CR - Customer Reference Number (conditional)</td></tr>
<tr><td>BG</td><td>n/a</td></tr>
<tr><td>SF</td><td>n/a</td></tr>
<tr><td>ST</td><td>n/a</td></tr>
<tr><td>40</td><td>n/a</td></tr>
</table></td></tr>
<tr><td>Y</td><td>Reference Identification</td><td>15-character alphanumeric</td><td>ref02</td><td>This field must be populated for each ref01 provided</td></tr>
</table>

<table>
<tr><th>Req*</th><th>Value</th><th>Limits</th><th>Variable</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>
<table>
<tr><th>ref01 value</th><th>ref02 denotation</th></tr>
<tr><td>4C (n101 value = R6)</td><td>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.</td></tr>
<tr><td>CR (n101 value = R6):</td><td>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.</td></tr>
</table>
</td></tr>
</table>

*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

```
$mpgAxLevel23->setTable2($axItLoop);
```

Table 1 AxLevel23 object - Table 2 object fields

Req*	Value	Limits	Set Method	Description
N	It1loop	Object	`'axIt1Loop'=>$axIt1Loop`	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

```
$axItLoop = new axIt1Loop();

$axItLoop->setIt1Loop($it102[0], $it103[0], $it104[0], $it105[0], $it106s[0],
$txi[0], $pam05[0], $pid05[0]);

$axItLoop->setIt1Loop($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

Req*	Value	Limits	Variable	Description
				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	Unit Price	15-character decimal	it104	Line item cost per unit Must contain 2 decimal places Minimum amount is 0.00 and maximum is 999999.99
N	Basis or Unit Price Code	2-character alphanumeric	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 further breakdown and definition of object details.
N	AxTxi	object	txi	Refer below for further breakdown

Req*	Value	Limits	Variable	Description
				and definition of 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

```
$it10618 = array("MG", "MG", "MG", "MG", "MG"); //Product/Service ID qualifier
```

```
$it10719 = array("DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT");
//Product/Service ID (corresponds to it10618)

$it106s = array();

$it106s[0] = new axIt106s($it10618[0], $it10719[0]);

$it106s[1] = new axIt106s($it10618[1], $it10719[1]);

$it106s[2] = new axIt106s($it10618[2], $it10719[2]);

$it106s[3] = new axIt106s($it10618[3], $it10719[3]);

$it106s[4] = new axIt106s($it10618[4], $it10719[4]);
```

Table 1 AxLevel23 object - Table 2 object - AxIt106s object fields

<table>
<tr><th>Req*</th><th>Value</th><th>Limits</th><th>Set Method</th><th>Description</th></tr>
<tr><td>N</td><td>Product/Service ID Qualifier</td><td>2-character alpha-numeric</td><td>'it10618'=>$it10618</td><td>Supported values:
MG - Manufacturer's Part Number
VC - Supplier Catalog Number
SK - Supplier Stock Keeping Unit Number
UP - Universal Product Code
VP – Vendor Part Number
PO – Purchase Order Number
AN – Client Defined Asset Code</td></tr>
<tr><td>N</td><td>Product/Service ID</td><td><table><tr><th>it10618</th><th>it10719 - size/type</th></tr><tr><td>VC</td><td>20-character alphanumeric</td></tr><tr><td>PO</td><td>22-character alphanumeric</td></tr><tr><td>Other</td><td>30-character alphanumeric</td></tr></table></td><td>'it10719'=>$it10719</td><td>Product/Service ID corresponds to the preceding qualifier defined by it10618
The maximum length depends on the qualifier defined in it10618</td></tr>
</table>

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

Table 2 - Setting the AxTxi Object

Table 2 AxiTxi object definition

```
$txi01_GST = array("GS", "GS", "GS", "GS", "GS"); //Tax type code
```

```
$txi02_GST = array("0.70", "1.75", "1.00", "0.80","0.00"); //Monetary amount

$txi03_GST = array("5.0", "5.0", "5.0", "5.0","5.0"); //Percent

$txi06_GST = array("", "", "", "",""); //Tax exempt code

$txi01_PST = array("PG", "PG", "PG","PG","PG"); //Tax type code

$txi02_PST = array("0.80", "2.00", "1.00", "0.80","0.00"); //Monetary amount

$txi03_PST = array("7.0", "7.0", "7.0", "7.0","7.0"); //Percent

$txi06_PST = array("", "", "", "",""); //Tax exempt code

$txi = array(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, then txi02, txi03 or txi06 must be populated Valid codes include

Req*	Value	Limits	Variable	Description
				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) If taxes are not

Req*	Value	Limits	Variable	Description
				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 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

```
$mpgAxLevel23->setTable3($taxTbl3);
```

Table 1 AxLevel23 object - Table 3 object fields

Req*	Value	Limits	Set Method	Description
C	AxTxi	Object	`'taxTbl3'=>$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 (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

```
$taxTbl3 = new axTxi();

$taxTbl3->setTxi("GS", "4.25","5.0",""); //sum of GST taxes

$taxTbl3->setTxi("PG", "4.60","7.0",""); //sum of PST taxes

$taxTbl3->setTxi("TX", "8.85","13.0",""); //sum of all taxes

$mpgAxLevel23->setTable3($taxTbl3);
```

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 Valid codes include the following: CT – County/Tax (optional) CA – City Tax (optional) EV – Environmental Tax (optional)

Req*	Value	Limits	Variable	Description
				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) 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

Req*	Value	Limits	Variable	Description
				(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 B – Material Taxable, Labor Exempt C – Not Taxable F – Exempt (Goods / Services Tax) G – Exempt (Provincial Sales Tax)

Req*	Value	Limits	Variable	Description
				L – Exempt Local Service 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

```
$txnArray = array(‘type’=>’axcompletion', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for AX Completion

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

AX Completion transaction object values

Table 1 AX Completion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Completion amount	String	(missing or bad snippet)	`'comp_amount'=>$comp_amount`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Level 2/3 Data	Object	n/a	`$mpgTxn->setLevel23Data ($mpgAxLevel23);`

Sample AX Completion

```
<?php
require "../../mpgClasses.php";
/***************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************ Transactional Variables ****************************/
$type='axcompletion';
$order_id='ord-210916-12:06:38';
$comp_amount='62.37';
$txn_number = '18924-0_11';
$crypt = '7';
//Create AxLevel23 Object
$mpgAxLevel23 = new mpgAxLevel23();
//Create Table 1 with details
$n101 = "R6"; //Entity ID Code
$n102 = "Retailing Inc. International"; //Name
$n301 = "919 Oriole Rd."; //Address Line 1
$n401 = "Toronto"; //City
$n402 = "On"; //State or Province
$n403 = "H1T6W3"; //Postal Code
$ref01 = array("4C", "CR"); //Reference ID Qualifier
$ref02 = array("M5T3A5", "16802309004"); //Reference ID
$big04 = "PO7758545"; //Purchase Order Number
$big05 = "RN0049858"; //Release Number
$big10 = "INV99870E"; //Invoice Number
$axRef1 = new axRef();
$axRef1->setRef($ref01[0], $ref02[0]);
$axRef1->setRef($ref01[1], $ref02[1]);
$axN1Loop = new axN1Loop();
$axN1Loop->setN1Loop($n101, $n102, $n301, $n401, $n402, $n403, $axRef1);
$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);
//Create Table 2 with details
//the sum of the extended amount field (pam05) must equal the level 1 amount field

$it102 = array("1", "1", "1", "1", "1"); //Line item quantity invoiced
$it103 = array("EA", "EA", "EA", "EA", "EA"); //Line item unit or basis of measurement code
$it104 = array("10.00", "25.00", "8.62", "10.00", "-10.00"); //Line item unit price
$it105 = array("", "", "", "", ""); //Line item basis of unit price code

$it10618 = array("MG", "MG", "MG", "MG", "MG"); //Product/Service ID qualifier
$it10719 = array("DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"); //Product/Service ID
(corresponds to it10618)

$txi01_GST = array("GS", "GS", "GS", "GS", "GS"); //Tax type code
$txi02_GST = array("0.70", "1.75", "1.00", "0.80","0.00"); //Monetary amount
$txi03_GST = array("", "", "", "",""); //Percent
$txi06_GST = array("", "", "", "",""); //Tax exempt code

$txi01_PST = array("PG", "PG", "PG","PG","PG"); //Tax type code
$txi02_PST = array("0.80", "2.00", "1.00", "0.80","0.00"); //Monetary amount
$txi03_PST = array("", "", "", "",""); //Percent
$txi06_PST = array("", "", "", "",""); //Tax exempt code
$pam05 = array("11.50", "28.75", "10.62", "11.50", "-10.00"); //Extended line-item amount
$pid05 = array("Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"); //Line item
description
$it106s = array(new axIt106s(), new axIt106s(), new axIt106s(), new axIt106s(), new axIt106s());
$it106s[0]->setIt10618($it10618[0]);
```

Sample AX Completion

```
$it106s[0]->setIt10719($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]);
$txi = array(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]);
$axItLoop = new axIt1Loop();
$axItLoop->setIt1Loop($it102[0], $it103[0], $it104[0], $it105[0], $it106s[0], $txi[0], $pam05[0],
$pid05[0]);
$axItLoop->setIt1Loop($it102[1], $it103[1], $it104[1], $it105[1], $it106s[1], $txi[1], $pam05[1],
$pid05[1]);
$axItLoop->setIt1Loop($it102[2], $it103[2], $it104[2], $it105[2], $it106s[2], $txi[2], $pam05[2],
$pid05[2]);
$axItLoop->setIt1Loop($it102[3], $it103[3], $it104[3], $it105[3], $it106s[3], $txi[3], $pam05[3],
$pid05[3]);
$axItLoop->setIt1Loop($it102[4], $it103[4], $it104[4], $it105[4], $it106s[4], $txi[4], $pam05[4],
$pid05[4]);
$mpgAxLevel23->setTable2($axItLoop);
//Create Table 3 with details
$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
$mpgAxLevel23->setTable3($taxTbl3);
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'comp_amount'=>$comp_amount,
'txn_number'=> $txn_number,
'crypt_type'=>$crypt
);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgAxLevel23);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response *************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
```

Sample AX Completion

```
    print("\nTransAmount = " . $mpgResponse->getTransAmount());
    print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
    print("\nReceiptId = " . $mpgResponse->getReceiptId());
    print("\nTransType = " . $mpgResponse->getTransType());
    print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
    print("\nResponseCode = " . $mpgResponse->getResponseCode());
    print("\nISO = " . $mpgResponse->getISO());
    print("\nMessage = " . $mpgResponse->getMessage());
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    ?>
```

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

```
$txnArray = array(‘type’=>’axforcepost', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for AX Force Post transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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 alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character alphanumeric	`'pan'=>$pan`

Value	Type	Limits	Set Method
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
Authorization code	String	8-character alpha-numeric	`'auth_code'=>$auth_code`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Level 2/3 Data	Object	n/a	`$mpgTxn->setLevel23Data ($mpgAxLevel23);`

Table 2 AX Force Post transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`

Sample AX Force Post

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='axforcepost';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='62.37';
$pan='373269005095005';
$expiry_date='2012';
$auth_code='123456';
$crypt = '7';
//Create AxLevel23 Object
$mpgAxLevel23 = new mpgAxLevel23();
//Create Table 1 with details
$n101 = "R6"; //Entity ID Code
$n102 = "Retailing Inc. International"; //Name
$n301 = "919 Oriole Rd."; //Address Line 1
$n401 = "Toronto"; //City
$n402 = "On"; //State or Province
$n403 = "H1T6W3"; //Postal Code
$ref01 = array("4C", "CR"); //Reference ID Qualifier
$ref02 = array("M5T3A5", "16802309004"); //Reference ID
$big04 = "PO7758545"; //Purchase Order Number
```

Sample AX Force Post

```
$big05 = "RN0049858"; //Release Number
$big10 = "INV99870E"; //Invoice Number
$axRef1 = new axRef();
$axRef1->setRef($ref01[0], $ref02[0]);
$axRef1->setRef($ref01[1], $ref02[1]);
$axN1Loop = new axN1Loop();
$axN1Loop->setN1Loop($n101, $n102, $n301, $n401, $n402, $n403, $axRef1);
$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);
//Create Table 2 with details
//the sum of the extended amount field (pam05) must equal the level 1 amount field

$it102 = array("1", "1", "1", "1", "1"); //Line item quantity invoiced
$it103 = array("EA", "EA", "EA", "EA", "EA"); //Line item unit or basis of measurement code
$it104 = array("10.00", "25.00", "8.62", "10.00", "-10.00"); //Line item unit price
$it105 = array("", "", "", "", ""); //Line item basis of unit price code

$it10618 = array("MG", "MG", "MG", "MG", "MG"); //Product/Service ID qualifier
$it10719 = array("DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"); //Product/Service ID
(corresponds to it10618)

$txi01_GST = array("GS", "GS", "GS", "GS", "GS"); //Tax type code
$txi02_GST = array("0.70", "1.75", "1.00", "0.80","0.00"); //Monetary amount
$txi03_GST = array("", "", "", "",""); //Percent
$txi06_GST = array("", "", "", "",""); //Tax exempt code

$txi01_PST = array("PG", "PG", "PG","PG","PG"); //Tax type code
$txi02_PST = array("0.80", "2.00", "1.00", "0.80","0.00"); //Monetary amount
$txi03_PST = array("", "", "", "",""); //Percent
$txi06_PST = array("", "", "", "",""); //Tax exempt code
$pam05 = array("11.50", "28.75", "10.62", "11.50", "-10.00"); //Extended line-item amount
$pid05 = array("Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"); //Line item
description
$it106s = array();
$it106s[0] = new axIt106s($it10618[0], $it10719[0]);
$it106s[1] = new axIt106s($it10618[1], $it10719[1]);
$it106s[2] = new axIt106s($it10618[2], $it10719[2]);
$it106s[3] = new axIt106s($it10618[3], $it10719[3]);
$it106s[4] = new axIt106s($it10618[4], $it10719[4]);
$txi = array(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]);
$axItLoop = new axIt1Loop();
$axItLoop->setIt1Loop($it102[0], $it103[0], $it104[0], $it105[0], $it106s[0], $txi[0], $pam05[0],
$pid05[0]);
$axItLoop->setIt1Loop($it102[1], $it103[1], $it104[1], $it105[1], $it106s[1], $txi[1], $pam05[1],
$pid05[1]);
$axItLoop->setIt1Loop($it102[2], $it103[2], $it104[2], $it105[2], $it106s[2], $txi[2], $pam05[2],
$pid05[2]);
$axItLoop->setIt1Loop($it102[3], $it103[3], $it104[3], $it105[3], $it106s[3], $txi[3], $pam05[3],
$pid05[3]);
//$axItLoop->setIt1Loop($it102[4], $it103[4], $it104[4], $it105[4], $it106s[4], $txi[4], $pam05
```

Sample AX Force Post

```
[4], $pid05[4]);
$mpgAxLevel23->setTable2($axItLoop);
//Create Table 3 with details
$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
$mpgAxLevel23->setTable3($taxTbl3);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'auth_code'=>$auth_code,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgAxLevel23);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’axpurchasecorrection', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for AX Purchase Correction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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 alphanumeric	`'order_id'=>$order_id`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

AX Purchase Correction

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='axpurchasecorrection';
$order_id='ord-210916-12:12:18';
$txn_number = '66011731632016265121219276-0_11';
$crypt_type = '7';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=> $txn_number,
'crypt_type'=>$crypt_type
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
```

AX Purchase Correction

```
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************** Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’axrefund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for AX Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Level 2/3 Data	Object	n/a	`$mpgTxn->setLevel23Data ($mpgAxLevel23);`

Sample AX Refund

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='axrefund';
$order_id='ord-210916-12:06:38';
$amount='62.37';
$txn_number = '18924-1_11';
$crypt = '7';
//Create AxLevel23 Object
$mpgAxLevel23 = new mpgAxLevel23();
//Create Table 1 with details
$n101 = "R6"; //Entity ID Code
$n102 = "Retailing Inc. International"; //Name
$n301 = "919 Oriole Rd."; //Address Line 1
$n401 = "Toronto"; //City
$n402 = "On"; //State or Province
$n403 = "H1T6W3"; //Postal Code
$ref01 = array("4C", "CR"); //Reference ID Qualifier
$ref02 = array("M5T3A5", "16802309004"); //Reference ID
$big04 = "PO7758545"; //Purchase Order Number
$big05 = "RN0049858"; //Release Number
$big10 = "INV99870E"; //Invoice Number
$axRef1 = new axRef();
$axRef1->setRef($ref01[0], $ref02[0]);
$axRef1->setRef($ref01[1], $ref02[1]);
$axN1Loop = new axN1Loop();
$axN1Loop->setN1Loop($n101, $n102, $n301, $n401, $n402, $n403, $axRef1);
```

Sample AX Refund

```
$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);
//Create Table 2 with details
//the sum of the extended amount field (pam05) must equal the level 1 amount field

$it102 = array("1", "1", "1", "1", "1"); //Line item quantity invoiced
$it103 = array("EA", "EA", "EA", "EA", "EA"); //Line item unit or basis of measurement code
$it104 = array("10.00", "25.00", "8.62", "10.00", "-10.00"); //Line item unit price
$it105 = array("", "", "", "", ""); //Line item basis of unit price code

$it10618 = array("MG", "MG", "MG", "MG", "MG"); //Product/Service ID qualifier
$it10719 = array("DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"); //Product/Service ID
(corresponds to it10618)

$txi01_GST = array("GS", "GS", "GS", "GS", "GS"); //Tax type code
$txi02_GST = array("0.70", "1.75", "1.00", "0.80","0.00"); //Monetary amount
$txi03_GST = array("", "", "", "",""); //Percent
$txi06_GST = array("", "", "", "",""); //Tax exempt code

$txi01_PST = array("PG", "PG", "PG","PG","PG"); //Tax type code
$txi02_PST = array("0.80", "2.00", "1.00", "0.80","0.00"); //Monetary amount
$txi03_PST = array("", "", "", "",""); //Percent
$txi06_PST = array("", "", "", "",""); //Tax exempt code
$pam05 = array("11.50", "28.75", "10.62", "11.50", "-10.00"); //Extended line-item amount
$pid05 = array("Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"); //Line item
description
$it106s = array();
$it106s[0] = new axIt106s($it10618[0], $it10719[0]);
$it106s[1] = new axIt106s($it10618[1], $it10719[1]);
$it106s[2] = new axIt106s($it10618[2], $it10719[2]);
$it106s[3] = new axIt106s($it10618[3], $it10719[3]);
$it106s[4] = new axIt106s($it10618[4], $it10719[4]);
$txi = array(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]);
$axItLoop = new axIt1Loop();
$axItLoop->setIt1Loop($it102[0], $it103[0], $it104[0], $it105[0], $it106s[0], $txi[0], $pam05[0],
$pid05[0]);
$axItLoop->setIt1Loop($it102[1], $it103[1], $it104[1], $it105[1], $it106s[1], $txi[1], $pam05[1],
$pid05[1]);
$axItLoop->setIt1Loop($it102[2], $it103[2], $it104[2], $it105[2], $it106s[2], $txi[2], $pam05[2],
$pid05[2]);
$axItLoop->setIt1Loop($it102[3], $it103[3], $it104[3], $it105[3], $it106s[3], $txi[3], $pam05[3],
$pid05[3]);
//$axItLoop->setIt1Loop($it102[4], $it103[4], $it104[4], $it105[4], $it106s[4], $txi[4], $pam05
[4], $pid05[4]);
$mpgAxLevel23->setTable2($axItLoop);
//Create Table 3 with details
$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
```

Sample AX Refund

```
$mpgAxLevel23->setTable3($taxTbl3);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'txn_number'=> $txn_number,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgAxLevel23);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

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

```
$txnArray = array(‘type’=>’axind_refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for AX Independent Refund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character alpha-numeric	`'pan'=>$pan`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 2 AX Independent Refund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`

Sample AX Independent Refund

```
<?php
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='moneris';
$api_token='hurgle';
//$status = 'false';
/************************* Transactional Variables ****************************/
$type='axind_refund';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='62.37';
$pan='373269005095005';
$expiry_date='2012';
$crypt = '7';
//Create AxLevel23 Object
$mpgAxLevel23 = new mpgAxLevel23();
//Create Table 1 with details
$n101 = "R6"; //Entity ID Code
$n102 = "Retailing Inc. International"; //Name
```

Sample AX Independent Refund

```
$n301 = "919 Oriole Rd."; //Address Line 1
$n401 = "Toronto"; //City
$n402 = "On"; //State or Province
$n403 = "H1T6W3"; //Postal Code
$ref01 = array("4C", "CR"); //Reference ID Qualifier
$ref02 = array("M5T3A5", "16802309004"); //Reference ID
$big04 = "PO7758545"; //Purchase Order Number
$big05 = "RN0049858"; //Release Number
$big10 = "INV99870E"; //Invoice Number
$axRef1 = new axRef();
$axRef1->setRef($ref01[0], $ref02[0]);
$axRef1->setRef($ref01[1], $ref02[1]);
$axN1Loop = new axN1Loop();
$axN1Loop->setN1Loop($n101, $n102, $n301, $n401, $n402, $n403, $axRef1);
$mpgAxLevel23->setTable1($big04, $big05, $big10, $axN1Loop);
//Create Table 2 with details
//the sum of the extended amount field (pam05) must equal the level 1 amount field

$it102 = array("1", "1", "1", "1", "1"); //Line item quantity invoiced
$it103 = array("EA", "EA", "EA", "EA", "EA"); //Line item unit or basis of measurement code
$it104 = array("10.00", "25.00", "8.62", "10.00", "-10.00"); //Line item unit price
$it105 = array("", "", "", "", ""); //Line item basis of unit price code

$it10618 = array("MG", "MG", "MG", "MG", "MG"); //Product/Service ID qualifier
$it10719 = array("DJFR4", "JFJ49", "FEF33", "FEE43", "DISCOUNT"); //Product/Service ID
(corresponds to it10618)

$txi01_GST = array("GS", "GS", "GS", "GS", "GS"); //Tax type code
$txi02_GST = array("0.70", "1.75", "1.00", "0.80","0.00"); //Monetary amount
$txi03_GST = array("", "", "", "",""); //Percent
$txi06_GST = array("", "", "", "",""); //Tax exempt code

$txi01_PST = array("PG", "PG", "PG","PG","PG"); //Tax type code
$txi02_PST = array("0.80", "2.00", "1.00", "0.80","0.00"); //Monetary amount
$txi03_PST = array("", "", "", "",""); //Percent
$txi06_PST = array("", "", "", "",""); //Tax exempt code
$pam05 = array("11.50", "28.75", "10.62", "11.50", "-10.00"); //Extended line-item amount
$pid05 = array("Stapler", "Lamp", "Bottled Water", "Fountain Pen", "DISCOUNT"); //Line item
description
$it106s = array();
$it106s[0] = new axIt106s($it10618[0], $it10719[0]);
$it106s[1] = new axIt106s($it10618[1], $it10719[1]);
$it106s[2] = new axIt106s($it10618[2], $it10719[2]);
$it106s[3] = new axIt106s($it10618[3], $it10719[3]);
$it106s[4] = new axIt106s($it10618[4], $it10719[4]);
$txi = array(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]);
$axItLoop = new axIt1Loop();
$axItLoop->setIt1Loop($it102[0], $it103[0], $it104[0], $it105[0], $it106s[0], $txi[0], $pam05[0],
$pid05[0]);
```

Sample AX Independent Refund

```
$axItLoop->setIt1Loop($it102[1], $it103[1], $it104[1], $it105[1], $it106s[1], $txi[1], $pam05[1],
$pid05[1]);
$axItLoop->setIt1Loop($it102[2], $it103[2], $it104[2], $it105[2], $it106s[2], $txi[2], $pam05[2],
$pid05[2]);
$axItLoop->setIt1Loop($it102[3], $it103[3], $it104[3], $it105[3], $it106s[3], $txi[3], $pam05[3],
$pid05[3]);
//$axItLoop->setIt1Loop($it102[4], $it103[4], $it104[4], $it105[4], $it106s[4], $txi[4], $pam05
[4], $pid05[4]);
$mpgAxLevel23->setTable2($axItLoop);
//Create Table 3 with details
$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
$mpgAxLevel23->setTable3($taxTbl3);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setLevel23Data($mpgAxLevel23);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

8 MPI

- 8.1 About MPI Transactions
- 8.2 3-D Secure Implementations (VbV, MCSC, SafeKey)
- 8.3 Activating 3-D Secure Functionality
- 8.4 Activating Amex SafeKey
- 8.5 Transaction Flow for MPI
- 8.6 MPI Transactions

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

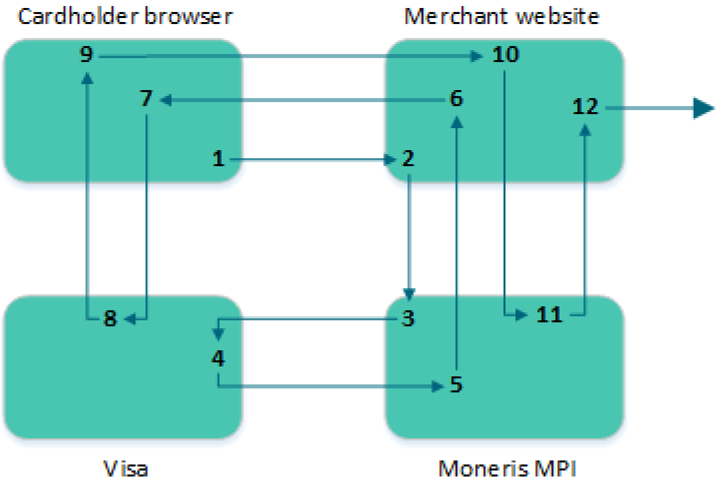


Figure 3: Transaction flow diagram

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 250).
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 17.5 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 10 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 71: 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 72: 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 73: 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 74: 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 74: 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

```
$txnArray = array(‘type’=>’mpitxn', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MpiTxn transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

MpiTxn transaction values

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

Table 75: MpiTxn transaction object mandatory values

Value	Type	Limits	Set method
XID	String	20-character alpha-numeric	'xid'=>$xid
Credit card number	String	20-character numeric	'pan'=>$pan
Expiry date	String	4-character alpha-numeric (YYMM format)	'expdate'=>$expiry_date
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits	'amount'=>$amount

Table 75: MpiTxn transaction object mandatory values (continued)

Value	Type	Limits	Set method
		(cents) after the decimal point **EXAMPLE:** 1234567.89	
MD	String	1024-character alpha-numeric	`MD=>$MD`
Merchant URL	String	N/A	`merchantUrl=>$merchantUrl`
Accept	String	N/A	`accept=>$accept`
User Agent	String	N/A	`userAgent=>$userAgent`

Sample MpiTxn Request

```
<?php
$store_id ="moneris";
$api_token="hurgle";
$merchUrl="https://YOUR_MPI_RESPONSE_URL";
include("../../mpgClasses.php");

$xid =sprintf("%'920d", rand());
$pan = "4242424242424242";
$expiry = "1811";
$purchase_amount = "1.00";

$HTTP_ACCEPT = getenv("HTTP_ACCEPT");
$HTTP_USER_AGENT = getenv("HTTP_USER_AGENT");

//these are form variable gotten after cardholder hits buy button on merchant site
//(purchase_amount,pan,expiry)
$txnArray=array('type'=>'txn',
'xid'=>$xid,
'amount'=>$purchase_amount,
'pan'=>$pan,
'expdate'=>$expiry,
'MD'=> "xid=" . $xid //MD is merchant data that can be passed along
."&amp;pan=" . $pan
."&amp;expiry=".$expiry
."&amp;amount=" .$purchase_amount,
'merchantUrl'=>$merchUrl,
'accept'=>$HTTP_ACCEPT,
'userAgent'=>$HTTP_USER_AGENT
);

$mpgTxn = new mpgTransaction($txnArray);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
```

Sample MpiTxn Request

```
$mpgRequest->setTestMode(true); //false or comment out this line for production
transactions
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
$mpgResponse=$mpgHttpPost->getMpgResponse();

if($mpgResponse->getMpiMessage() == 'Y')
{
$vbvInLineForm = $mpgResponse->getMpiInLineForm();
print "$vbvInLineForm\n";
}
else {
if ($mpgResponse->getMpiMessage() == 'U') {
// merchant assumes liability for charge back (usu. corporate cards)
$crypt_type='7';
}
else {
// merchant is not liable for chargeback (attempt was made)
$crypt_type='6';
}
//Perform regular transaction with $crypt_type='7'
}
?>
```

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.

8.6.3 Vault MPI Transaction – ResMpiTxn

Vault MPI Transaction transaction object definition

```
$txnArray = array(‘type’=>’res_mpitxn', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault MPI Transaction transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault MPI Transaction transaction values

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

Table 76: Vault MPI Transaction transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`'data_key'=>$data_key`
XID	String	20-character alpha-numeric	`'xid'=>$xid`
Amount	String	10-character decimal Up to 7 digits (dollars) + decimal point (.) + 2 digits (cents) after the decimal point **EXAMPLE:** 1234567.89	`'amount'=>$amount`
MD	String	1024-character alpha-numeric	`MD=>$MD`
Merchant URL	String	n/a	`merchantUrl=>$merchantUrl`
Accept	String	n/a	`accept=>$accept`
User Agent	String	n/a	`userAgent=>$userAgent`
Expiry date	String	4-character alpha-numeric (YYMM format)	`'expdate'=>$expiry_date`

Sample Vault MPI Transaction

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$data_key='ot-DYm9m3m00lCgN2b1Kk6mEb7np';
```

Sample Vault MPI Transaction

```
$amount='1.00';
$xid = sprintf("%'920d", rand());
$MD = $xid."mycardinfo".$amount;
$merchantUrl = "www.mystoreurl.com";
$accept = "true";
$userAgent = "Mozilla";
$expdate = "1712"; //For Temp Tokens only
/************************ Transaction Array **********************************/
$txnArray =array(type=>'res_mpitxn',
data_key=>$data_key,
//expdate=>$expdate,
amount=>$amount,
xid=>$xid,
MD=>$MD,
merchantUrl=>$merchantUrl,
accept=>$accept,
userAgent=>$userAgent
);
/************************ Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nMpiSuccess = " . $mpgResponse->getMpiSuccess());
if($mpgResponse->getMpiSuccess() == "true")
{
print($mpgResponse->getMpiInLineForm());
}
else
{
print("\nMpiMessage = " . $mpgResponse->getMpiMessage());
}
?>
```

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 441).

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

```
$txnArray = array(‘type’=>’acs', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for MPI ACS Request transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

MPI ACS Request transaction values

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

Table 77: 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	`'amount'=>$amount`
MD	String	1024-character alpha-numeric	`MD=>$MD`
PARes	String	n/a	`'PaRes'=>$PaRes`

Sample MPI ACS Request

```
<?php
require "../../mpgClasses.php";
/******************************* Request Variables ********************************/
$store_id = "moneris";
$api_token = "hurgle";
/****************************** Transactional Variables ***************************/
$type='acs';
$PaRes = "PaRes String";
$MD = "mycardinfo";
/*************************** Transaction Associative Array ***********************/
$txnArray=array(
'type'=>$type,
'PaRes'=>$PaRes,
'MD'=>$MD,
);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
```

Sample MPI ACS Request

```
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************* Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nMpiMessage = " . $mpgResponse->getMpiMessage());
print("\nMpiSuccess = " . $mpgResponse->getMpiSuccess());
if (strcmp($mpgResponse->getMpiSuccess(),"true") == 0)
{
print("\nCAVV = " . $mpgResponse->getMpiCavv());
print("\nECI = " . $mpgResponse->getMpiEci());
}
?>
```

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 – cavv_purchase

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 10 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

```
$txnArray = array(‘type’=>’cavv_purchase', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Purchase with 3-D Secure transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Cavv Purchase transaction values

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

Table 78: Purchase with 3-D Secure transaction object mandatory values

Value	Type	Limits	Set method
Order ID	*String*	50-character alphanumeric	`'order_id'=>$order_id`
Amount	*String*	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	*String*	20-character alphanumeric	`'pan'=>$pan`

Table 78: Purchase with 3-D Secure transaction object mandatory values

Value	Type	Limits	Set method
Expiry date	*String*	4-character alphanumeric (YYMM format)	`'expdate'=>$expiry_date`
CAVV **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	*String*	50-character alphanumeric	`cavv=>$cavv`
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	`'crypt_type'=>$crypt`

Table 1 INTERAC® e-Commerce Fields – Required for Apple Pay and Google Pay Only

Variable Name	Type	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	alphabetical	
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	

Table 2 Purchase with 3-D Secure transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	*String*	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	*String*	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Card Match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String*	50-character alphanumeric	`'cm_id' => $transaction_id`
Customer information	*Object*	N/A	`$mpgTxn->setCustInfo($mpgCustInfo);`

Value	Type	Limits	Set Method
AVS	*Object*	N/A	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD	*Object*	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
Convenience fee **NOTE:** Not applicable when processing Apple Pay transactions.	*Object*	N/A	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`

Value	Type	Limits	Set Method
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	`$mpgTxn->setRecur($mpgRecur);`
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 alpha-numeric	`'wallet_indicator'=>$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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	*Object*	N/A	

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Recurring Billing Info Object Request Fields

Variable and Field 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 units that must pass between recurring billings

Variable and Field Name	Type and Limits	Description
Start Date `start_date`	*String* YYYY/MM/DD	Date of the first future recurring billing transaction This value **must** be a date in the future If an additional charge is to be made immediately, the value of Start Now must be set to true
Start Now `start_now`	*String* true/false	If a single charge is to be made against the card immediately, set this value to true; the amount to be billed immediately may differ from the amount billed on a regular basis thereafter If the billing is to start in the future, set this value to false When set to false, use Card Verification prior to sending the Purchase with Recur and Credential on File objects
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	Amount of the recurring transaction This is the amount that 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 Period to define the billing frequency Possible values are: day week

Variable and Field Name	Type and Limits	Description
		month eom (end of month)

Sample Purchase with 3-D Secure - cavv_purchase

```
<?php
## Example php -q TestPurchase-VBV.php "moneris" store
require "../../mpgClasses.php";
/******************************* Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/****************************** Transactional Variables ***************************/
$type='cavv_purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="1511";
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$crypt_type = '7';
$wallet_indicator = "APP";
$dynamic_descriptor='123456';
/*************************** Transaction Associative Array ************************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'cavv'=>$cavv,
'crypt_type'=>$crypt_type, //mandatory for AMEX only
//'wallet_indicator'=>$wallet_indicator, //set only for wallet transactions. e.g. APPLE PAY
//'network'=> "Interac", //set only for Interac e-commerce
//'data_type'=> "3DSecure", //set only for Interac e-commerce
'dynamic_descriptor'=>$dynamic_descriptor
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************* Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
```

Sample Purchase with 3-D Secure - cavv_purchase

```
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

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

```
<?php
## Example php -q TestPurchase-VBV.php "moneris" store
require "../../mpgClasses.php";
/******************************* Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/****************************** Transactional Variables ***************************/
$type='cavv_purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="1511";
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$crypt_type = '7';
$wallet_indicator = "APP";
$dynamic_descriptor='123456';
/********************************** Recur Variables ****************************/
$recurUnit = 'month'; //eom - end of month
$startDate = '2018/02/06';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';
/**************************** Transaction Associative Array ************************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
```

Purchase with 3-D Secure and Recurring Billing

```
'expdate'=>$expiry_date,
'cavv'=>$cavv,
'crypt_type'=>$crypt_type, //mandatory for AMEX only
//'wallet_indicator'=>$wallet_indicator, //set only for wallet transactions. e.g. APPLE PAY
//'network'=> "Interac", //set only for Interac e-commerce
//'data_type'=> "3DSecure", //set only for Interac e-commerce
'dynamic_descriptor'=>$dynamic_descriptor
);
/*********************** Recur Associative Array **********************/
$recurArray = array('recur_unit'=>$recurUnit, // (day | week | month)
'start_date'=>$startDate, //yyyy/mm/dd
'num_recurs'=>$numRecurs,
'start_now'=>$startNow,
'period' => $recurInterval,
'recur_amount'=> $recurAmount
);
$mpgRecur = new mpgRecur($recurArray);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Recur Object *********************************/
$mpgTxn->setRecur($mpgRecur);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("R");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************* Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

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 10 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

```
$txnArray = array(‘type’=>’cavv_preauth', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Pre-Authorization with 3-D Secure transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Pre-Authorization with 3-D Secure transaction values

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

Table 79: Pre-Authorization with 3-D Secure object mandatory values

Value	Type	Limits	Set method
Order ID	*String*	50-character alphanumeric	`'order_id'=>$order_id`
Amount	*String*	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	*String*	20-character numeric	`'pan'=>$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	`cavv=>$cavv`
Expiry date	*String*	4-character numeric	`'expdate'=>$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	`'crypt_type'=>$crypt`

Table 1 INTERAC® e-Commerce Fields – Required for Apple Pay and Google Pay Only

Variable Name	Type	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	alphabetical	
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	

Table 2 Pre-Authorization with 3-D Secure object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	*String*	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	*String*	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Card Match ID
NOTE: Applies to Offlinx™ only; must be unique value for each transaction
	String	50-character alphanumeric	`cavvPreauth`
			`'cm_id' => $transaction_id`

AVS	*Object*	N/A	`$mpgTxn->setAvsInfo($mpgAvsInfo);`

Value	Type	Limits	Set method
CVD	*Object*	N/A	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`
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 alpha-numeric	`'wallet_indicator'=>$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. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	*Object*	N/A	

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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	`$cof->setIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`$cof->setPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions 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 Definitions of Request Fields – Credential on File

Sample Pre-Authorization with 3-D Secure – cavvPreauth

```
<?php
## Example php -q TestPurchase-VBV.php "moneris" store
require "../../mpgClasses.php";
/******************************* Request Variables *********************************/
$store_id='store5';
$api_token='yesguy';
/****************************** Transactional Variables ****************************/
$type='cavv_preauth';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="0812";
```

Sample Pre-Authorization with 3-D Secure – cavvPreauth

```
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$crypt_type = '7';
$wallet_indicator = "APP";
$dynamic_descriptor='123456';
/*************************** Transaction Associative Array ************************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'cavv'=>$cavv,
'crypt_type'=>$crypt_type, //mandatory for AMEX only
//'wallet_indicator'=>$wallet_indicator, //set only for wallet transactions. e.g. APPLE PAY
'dynamic_descriptor'=>$dynamic_descriptor
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************ Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

8.6.7 Cavv Result Codes for Verified by Visa

Table 80: CAVV result codes for VbV

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	Please check that you are following the VbV process correctly and passing the correct data in our

Table 80: CAVV result codes for VbV (continued)

Code	Message	Significance
	only)	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)
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

```
$txnArray = array(‘type’=>’res_cavv_purchase_cc', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Cavv Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Cavv Purchase transaction details

Table 81: Vault Cavv Purchase transaction object mandatory values

Value	Type	Limits	Set method
Data Key	String	25-character alphanumeric	`'data_key'=>$data_key`
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`

Table 81: Vault Cavv Purchase transaction object mandatory values

Value	Type	Limits	Set method
Cardholder Authentication Verification Value (CAVV)	String	50-character alphanumeric	`cavv=>$cavv`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 82: Vault Cavv Purchase transaction object optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Expiry date	String	4-character alphanumeric (YYMM format)	`'expdate'=>$expiry_date`

Sample Vault Cavv Purchase

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables ******************************/
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
$orderid='res-purchase-'.date("dmy-G:i:s");
$amount='1.00';
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA';
$custid='cust'; //if sent will be submitted, otherwise cust_id from profile will be used
$expdate = '1902'; //YYMM - used only for temp token
$crypt_type = '6'; //value obtained from MpiACS transaction
/************************ Transaction Array **********************************/
$txnArray =array('type'=>'res_cavv_purchase_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'cavv'=>$cavv,
```

Sample Vault Cavv Purchase

```
//'expdate'=>$expdate, //mandatory for temp tokens only
//'crypt_type'=>$crypt_type, //set for AMEX SafeKey only
'dynamic_descriptor'=>'12346'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

8.6.9 Vault Cavv Pre-Authorization

Vault Cavv Pre-Authorization transaction object definition

```
$txnArray = array(‘type’=>’res_cavv_preauth_cc', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Vault Cavv Pre-Authorization

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Vault Cavv Pre-Authorization transaction details

Table 83: Vault Cavv Pre-Authorization object mandatory values

Value	Type	Limits	Set method
Data Key	String	25-character alphanumeric	`'data_key'=>$data_key`
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
CAVV	String	50-character alphanumeric	`cavv=>$cavv`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 84: Vault Cavv Pre-Authorization object optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Expiry date	String	4-character numeric	`'expdate'=>$expiry_date`

Sample Vault Cavv Pre-Authorization

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='store5';
$api_token='yesguy';
/************************ Transaction Variables *******************************/
$data_key='t8RCndWBNFNt4Dx32CCnl2tlz';
$orderid='res-preauth-'.date("dmy-G:i:s");
$amount='1.00';
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA';
```

Sample Vault Cavv Pre-Authorization

```
$custid='cust'; //if sent will be submitted, otherwise cust_id from profile will be used
$expdate = '1902'; //YYMM - used only for temp token
$crypt_type = '6'; //value obtained from MpiACS transaction
/************************ Transaction Array **********************************/
$txnArray =array('type'=>'res_cavv_preauth_cc',
'data_key'=>$data_key,
'order_id'=>$orderid,
'cust_id'=>$custid,
'amount'=>$amount,
'cavv'=>$cavv,
//'expdate'=>$expdate, //mandatory for temp tokens only
//'crypt_type'=>$crypt_type, //set for AMEX SafeKey only
'dynamic_descriptor'=>'12346'
);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
//----------------- ResolveData ------------------------------
print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
?>
```

9 e-Fraud Tools

- 9.1 Address Verification Service
- 9.2 Card Validation Digits (CVD)
- 9.3 Transaction Risk Management Tool

9.1 Address Verification Service

- 9.1.1 About Address Verification Service (AVS)
- 9.1.2 AVS Info Object
- 1 AVS Information Object
- 9.1.3 AVS Response Codes
- 9.1.4 AVS Sample Code

9.1.1 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.

9.1.2 AVS Info Object

AVSInfo object definition

```
$avsTemplate = array(
'avs_street_number'=>$avs_street_number,
'avs_street_name' =>$avs_street_name,
'avs_zipcode' => $avs_zipcode,
'avs_hostname'=>$avs_hostname,
'avs_browser' =>$avs_browser,
'avs_shiptocountry' => $avs_shiptocountry,
'avs_merchprodsku' => $avs_merchprodsku,
'avs_custip'=>$avs_custip,
'avs_custphone' => $avs_custphone
);
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
```

Transaction object set method

```
$mpgTxn->setAvsInfo($mpgAvsInfo);
```

Table 1 AVS Info Object – Required Fields

Variable and Field 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	`'avs_street_number'=>'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	`'avs_street_name' =>'Payton Street'`	Cardholder street name
AVS zip/postal code	*String*	`'avs_zipcode'`	Cardholder zip/postal

Variable and Field Name	Type and Limits	Set Method	Description
	9-character alphanumeric	`=>'M1M1M1'`	code

9.1.3 AVS Response Codes

Below is a full list of possible AVS response codes. These can be returned when you call the `$mpgResponse->getAvsResultCode()` method .

Table 85: AVS result codes

Value	Visa	MasterCard/Discover	Amex/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.

Table 85: AVS result codes (continued)

Value	Visa	MasterCard/Discover	Amex/JCB
G	Address information not verified for international transaction. Any of the following may be true: • Issuer is not an AVS participant. • AVS data was present in the request, but issuer did not return an AVS result. • Visa performs AVS on behalf of the issuer and there was no address record on file for this account.	N/A	N/A
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.
N/A	N/A	Customer name and zip/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. Also used when acquirer requests AVS but sends no AVS data.	Neither address nor postal code matches.	Billing address and postal code do not match.
O	N/A	N/A	Customer name and billing address match
P	Postal code matches. Acquirer sent both postal code and street address, but street address not verified due to incompatible formats.	N/A	N/A
R	Retry: System unavailable or timed out. Issuer ordinarily performs AVS, but was unavailable. The code R is used by Visa when issuers are unavailable. Issuers should refrain from using this code.	Retry. System unable to process.	Retry. System unavailable.

Table 85: AVS result codes (continued)

Value	Visa	MasterCard/Discover	Amex/JCB
S	N/A	AVS currently not supported.	AVS currently not supported.
T	N/A	Nine-digit zip/postal code matches, address does not match.	N/A
U	Address not verified for domestic transaction. One of the following is true: • Issuer is not an AVS participant • AVS data was present in the request, but issuer did not return an AVS result • 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 US Addresses, nine-digit zip/postal code matches, address does not. For addresses outside the US, zip/postal code matches, address does not.	Customer name, billing address, and zip/postal code are all correct.
X	N/A	For US addresses, nine-digit zip/postal code and address match. For addresses outside the US,zip/postal code and address match.	N/A
Y	Street address and zip/postal code match.	For US addresses, five-digit zip/postal code and address match.	Billing address and zip/postal code match.
Z	Zip/postal code matches, but street address either does not match or street address was not included in request.	For U.S. addresses, five-digit zip code matches, address does not match.	Postal code matches, billing address does not match.

9.1.4 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

```
$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';
$avs_email = 'test@host.com';
$avs_hostname = "www.testhost.com';
$avs_browser = 'Mozilla';
$avs_shiptocountry = 'Canada';
$avs_merchprodsku = '123456';
$avs_custip = '192.168.0.1';
$avs_custphone = '5556667777';
$avsTemplate = array(
'avs_street_number'=>$avs_street_number,
'avs_street_name' =>$avs_street_name,
'avs_zipcode' => $avs_zipcode,
'avs_hostname'=>$avs_hostname,
'avs_browser' =>$avs_browser,
'avs_shiptocountry' => $avs_shiptocountry,
'avs_merchprodsku' => $avs_merchprodsku,
'avs_custip'=>$avs_custip,
'avs_custphone' => $avs_custphone
);
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
$txnArray=array(
'type'=>'purchase',
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt
);
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
```

9.2 Card Validation Digits (CVD)

- 9.2.1 About Card Validation Digits (CVD)
- 9.2.3 CVD Information Object
- 9.2.4 CVD Result Codes
- 9.2.5 Sample Purchase with CVD Info Object

9.2.1 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.

9.2.2 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

9.2.3 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.

CVD Information object definition

```
<!-- start CVD -->

<!ELEMENT cvd_info (cvd_indicator, cvd_value)>

<!ELEMENT cvd_indicator (#PCDATA)>

<!ELEMENT cvd_value (#PCDATA)>
```

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);
```

CVD Info object request fields – Required

Variable Name	Type and Limits	Description
CVD indicator `<cvd_indicator>`	*String* 1-character numeric	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 printed on the card, but is illegible 9 – Cardholder states that the card has no

Variable Name	Type and Limits	Description
		CVD
CVD value `<cvd_value>`	*String* 4-character numeric	CVD value printed on card **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.

Table 1 CVD Info Object – Required Fields

Variable and Field Name	Type and Limits	Set Method	Description
CVD indicator	*String* 1-character numeric	`'cvd_indicator' =>'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 numeric	`'cvd_value' =>'123'`	CVD value located on credit card **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.

9.2.4 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

9.2.5 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

```
$cvdTemplate = array(
'cvd_indicator' => '1',
'cvd_value' => '123'
);
$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
$txnArray=array(
'type'=>'purchase',
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt
);
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCvdInfo($mpgCvdInfo);
```

9.3 Transaction Risk Management Tool

- 9.3.1 About the Transaction Risk Management Tool
- 9.3.2 Introduction to Queries
- 9.3.3 Session Query
- 9.3.4 Attribute Query
- 9.3.6 Inserting the Profiling Tags Into Your Website
- 9.3.6 Inserting the Profiling Tags Into Your Website

The Transaction Risk Management Tool (TRMT) is available to **Canadian integrations** only.

9.3.1 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).

9.3.2 Introduction to Queries

There are two types of transactions associated with the Transaction Risk Management Tool (TRMT):

- Session Query (page 292)
- Attribute Query (page 299)

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.

9.3.3 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

```
$riskTxn = new riskTransaction($txnArray);
```

HttpsPostRequest object for Session Query transaction

```
$riskHttpsPost =new riskHttpsPost($store_id,$api_token,$riskRequest);
```

Session Query transaction values

Table 86: Session Query transaction object mandatory values

<table>
<tr><th rowspan="2">Value</th><th>Type</th><th>Limits</th><th>Set method</th></tr>
<tr><th colspan="3">Description</th></tr>
<tr><td rowspan="2">Session ID</td><td>String</td><td>9-character decimal
Permitted characters: [a-z], [A-Z], 0-9, _, -</td><td>'session_id'=>$session_id</td></tr>
<tr><td colspan="3">Web server session identifier generated when device profiling was initiated.</td></tr>
<tr><td rowspan="2">Service type</td><td>String</td><td>9-character decimal</td><td>'service_type'=>$service_type</td></tr>
<tr><td colspan="3">Which output fields are returned.
session -- returns IP and device related attributes.</td></tr>
<tr><td rowspan="2">Event type</td><td>String</td><td>payment</td><td>'event_type'=>$event_type</td></tr>
<tr><td colspan="3">Defines the type of transaction or event for reporting purposes.
payment - Purchasing of goods/services.</td></tr>
<tr><td rowspan="2">Credit card number (PAN)</td><td>String</td><td>20-character numeric
No spaces or dashes</td><td>'pan'=>$pan</td></tr>
<tr><td colspan="3">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.</td></tr>
<tr><td rowspan="2">Account address street 1</td><td>String</td><td>32-character alphanumeric</td><td>'account_address_street1'=>$account_address_street1</td></tr>
<tr><td colspan="3">First portion of the street address component of the billing address.</td></tr>
</table>

Table 86: Session Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Account Address street 2	String	32-character alphanumeric	'account_address_street2'=>$account_address_street2
	Second portion of the street address component of the billing address.		
Account address city	String	50-character alphanumeric	'account_address_city'=>$account_address_city
	The city component of the billing address.		
Account address state/-province	String	64-character alphanumeric	'account_address_state'=>$account_address_state
	The state/province component of the billing address.		
Account address country	String	2-character alphanumeric	'account_address_country'=>$account_address_country
	ISO2 country code of the billing addresses.		
Account address ZIP/-postal code	String	8-character alphanumeric	'account_address_zip'=>$account_address_zip
	ZIP/postal code of the billing address.		
Shipping address street 1	String	32-character alphanumeric	'shipping_address_street1'=>$shipping_address_street1
	First portion of the street address component of the shipping address.		
Shipping address street 2	String	32-character alphanumeric	'shipping_address_street2'=>$shipping_address_street2
	Second portion of the street address component of the shipping address.		
Shipping address city	String	50-character alphanumeric	'shipping_address_city'=>$shipping_address_city
	City component of the shipping address.		
Shipping address state/-province	String	64-character alphanumeric	'shipping_address_state'=>$shipping_address_state
	The state/province component of the shipping address.		

Table 86: Session Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Shipping address country	String	2-character alphanumeric	'shipping_address_country'=>$shipping_address_country
	ISO2 country code of the account address country.		
Shipping address ZIP	String	8-character alphanumeric	'shipping_address_zip'=>$shipping_address_zip
	The ZIP/postal code component of the shipping address.		
Local attribute 1-5	String	255-character alphanumeric	
	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	
	The numeric currency amount.		
Transaction currency	String	10-character numeric	
	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 87: Session Query transaction object optional values

Value	Type	Limits	Set method
	Description		
Account login	String	255-character alphanumeric	'account_login'=>$account_login
	The Account Login name.		
Password hash	String	40-character alphanumeric	'password_hash' =>$password_hash
	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 87: Session Query transaction object optional values (continued)

Value	Type	Limits	Set method	Description
Account number	String	255-character alphanumeric	'account_number' => $account_number	The account number for the account.
Account name	String	255-character alphanumeric	'account_name' => $account_name	Account name (or concatenation of first and last name of account holder).
Account email	String	100-character alphanumeric	'account_email'=>$account_email	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
Address ZIP	String	8-character alphanumeric		The ZIP/postal code of the account address.

Table 87: Session Query transaction object optional values (continued)

Value	Type	Limits	Set method
	Description		
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

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables ****************************/
$store_id='moneris';
$api_token='hurgle';
/********************* Transactional Variables *************************/
$type='session_query';
```

Sample Session Query - CA

```
$order_id='risktest-'.date("dmy-G:i:s");
$session_id='abc123';
$service_type='session';
//$event_type='login';
/************************** SessionAccountInfo Variables ******************************/
$policy = '';
$device_id = '4EC40DE5-0770-4fa0-BE53-981C067C598D';
$account_login = '13195417-8CA0-46cd-960D-14C158E4DBB2';
$password_hash = '489c830f10f7c601d30599a0deaf66e64d2aa50a';
$account_number = '3E17A905-AC8A-4c8d-A417-3DADA2A55220';
$account_name = '4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD';
$account_email = '3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com';
$account_telephone = '5556667777';
$pan = '4242424242424242';
$account_address_street1 = '3300 Bloor St W';
$account_address_street2 = '4th Flr West Tower';
$account_address_city = 'Toronto';
$account_address_state ='Ontario';
$account_address_country = 'CA';
$account_address_zip = 'M8X2X2';
$shipping_address_street1 = '3300 Bloor St W';
$shipping_address_street2 = '4th Flr West Tower';
$shipping_address_city = 'Toronto';
$shipping_address_state = 'Ontario';
$shipping_address_country = 'CA';
$shipping_address_zip = 'M8X2X2';
$local_attrib_1 = 'a';
$local_attrib_2 = 'b';
$local_attrib_3 = 'c';
$local_attrib_4 = 'd';
$local_attrib_5 = 'e';
$online_tld = 'Facebook';
$online_id_handle = 'Moneris';
$transaction_amount = '1.00';
$transaction_currency = '124';
/********************** SessionAccountInfo Associative Array **************************/
$sessionAccountInfoTemplate = array
(
'account_login'=>$account_login,
'password_hash' =>$password_hash,
'account_number' => $account_number,
'account_name' => $account_name,
'account_email'=>$account_email,
'pan' =>$pan
);
/************************** SessionAccountInfo Object *********************************/
$mpgSessionAccountInfo = new mpgSessionAccountInfo ($sessionAccountInfoTemplate);
/***************** Transactional Associative Array *********************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'session_id'=>$session_id,
'service_type'=>$service_type
);
/********************** Transaction Object *****************************/
$riskTxn = new riskTransaction($txnArray);
/************************ Set SessionAccountInfo ******************************/
$riskTxn->setSessionAccountInfo($mpgSessionAccountInfo);
/************************ Request Object *******************************/
```

Sample Session Query - CA

```
$riskRequest = new riskRequest($riskTxn);
$riskRequest->setTestMode(true);
/*********************** HTTPS Post Object ****************************/
$riskHttpsPost =new riskHttpsPost($store_id,$api_token,$riskRequest);
/***************************** Response ******************************/
$riskResponse=$riskHttpsPost->getRiskResponse();
//print("\nResponse = " . $riskResponse);
print("\nResponseCode = " . $riskResponse->getResponseCode());
print("\nMessage = " . $riskResponse->getMessage());
$results = $riskResponse->getResults();
foreach($results as $key => $value)
{
print("\n".$key ." = ". $value);
}
$rules = $riskResponse->getRules();
//print_r($rules);
foreach ($rules as $i)
{
foreach ($i as $key => $value)
{
echo "\n$key = $value";
}
}
?>
```

9.3.3.1 Session Query Transaction Flow

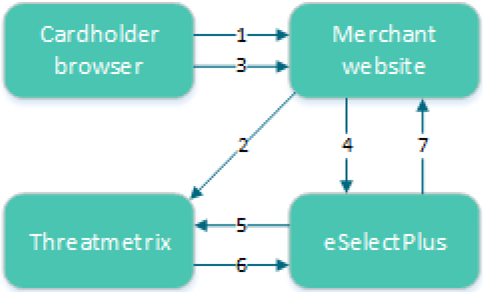


Figure 4: Session Query transaction flow

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 transaction.

9.3.4 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

```
$riskTxn = new riskTransaction($txnArray);
```

HttpsPostRequest object for AttributeQuery transaction

```
$riskHttpsPost =new riskHttpsPost($store_id,$api_token,$riskRequest);
```

Attribute Query transaction values

Table 88: Attribute Query transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Service type	String	N/A	`'service_type'=>$service_type`
	Which output fields are returned. session -- returns IP and device related attributes.		
Device ID	String	36-character alphanumeric	`'device_id'=>$device_id`
	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	`'pan'=>$pan`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		
IP address	String	64-character alphanumeric	`'ip_address'=>$ip_address`
	True IP address. Results will be returned as true_ip_geo, true_ip_score and so on.		

Table 88: Attribute Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
IP forwarded	String	64-character alphanumeric	`'ip_forwarded'=>$ip_forwarded`
	The IP address of the proxy. If the IPAddress is supplied, results will be returned as proxy_ip_geo and proxy_ip_score. If the IP Address is not supplied, this IP address will be treated as the true IP address and results will be returned as true_ip_geo, true_ip_score and so on		
Account address street 1	String	32-character alphanumeric	`'account_address_street1'=>$account_address_street1`
	First portion of the street address component of the billing address.		
Account Address Street 2	String	32-character alphanumeric	`'account_address_street2'=>$account_address_street2`
	Second portion of the street address component of the billing address.		
Account address city	String	50-character alphanumeric	`'account_address_city'=>$account_address_city`
	The city component of the billing address.		
Account address state/province	String	64-character alphanumeric	`'account_address_state'=>$account_address_state`
	The state component of the billing address.		
Account address country	String	2-character alphanumeric	`'account_address_country'=>$account_address_country`
	ISO2 country code of the billing addresses.		
Account address zip/postal code	String	8-character alphanumeric	`'account_address_zip'=>$account_address_zip`
	Zip/postal code of the billing address.		
Shipping address street 1	String	32-character alphanumeric	`'shipping_address_street1'=>$shipping_address_street1`
	Account address country		
Shipping Address Street 2	String	32-character alphanumeric	`'shipping_address_street2'=>$shipping_address_street2`
	Second portion of the street address component of the shipping address.		

Table 88: Attribute Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Shipping Address City	String	50-character alphanumeric	'shipping_address_city'=>$shipping_address_city
	City component of the shipping address.		
Shipping Address State/Province	String	64-character alphanumeric	'shipping_address_state'=>$shipping_address_state
	State/Province component of the shipping address.		
Shipping Address Country	String	2-character alphanumeric	'shipping_address_country'=>$shipping_address_country
	ISO2 country code of the account address country.		
Shipping Address zip/postal code	String	8-character alphanumeric	'shipping_address_zip'=>$shipping_address_zip
	The zip/postal code component of the shipping address.		

Sample Attribute Query

```
<?php
require "../../mpgClasses.php";
/************************ Request Variables ***************************/
$store_id='moneris';
$api_token='hurgle';
/********************* Transactional Variables ************************/
$type='session_query';
$order_id='risktest-'.date("dmy-G:i:s");
$session_id='abc123';
$service_type='session';
//$event_type='login';
/************************** SessionAccountInfo Variables *****************************/
$policy = '';
$device_id = '4EC40DE5-0770-4fa0-BE53-981C067C598D';
$account_login = '13195417-8CA0-46cd-960D-14C158E4DBB2';
$password_hash = '489c830f10f7c601d30599a0deaf66e64d2aa50a';
$account_number = '3E17A905-AC8A-4c8d-A417-3DADA2A55220';
$account_name = '4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD';
$account_email = '3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com';
$account_telephone = '5556667777';
$pan = '4242424242424242';
$account_address_street1 = '3300 Bloor St W';
$account_address_street2 = '4th Flr West Tower';
$account_address_city = 'Toronto';
$account_address_state ='Ontario';
$account_address_country = 'CA';
$account_address_zip = 'M8X2X2';
$shipping_address_street1 = '3300 Bloor St W';
$shipping_address_street2 = '4th Flr West Tower';
$shipping_address_city = 'Toronto';
```

Sample Attribute Query

```
$shipping_address_state = 'Ontario';
$shipping_address_country = 'CA';
$shipping_address_zip = 'M8X2X2';
$local_attrib_1 = 'a';
$local_attrib_2 = 'b';
$local_attrib_3 = 'c';
$local_attrib_4 = 'd';
$local_attrib_5 = 'e';
$online_tld = 'Facebook';
$online_id_handle = 'Moneris';
$transaction_amount = '1.00';
$transaction_currency = '124';
/********************** SessionAccountInfo Associative Array *************************/
$sessionAccountInfoTemplate = array
(
'account_login'=>$account_login,
'password_hash' =>$password_hash,
'account_number' => $acount_number,
'account_name' => $account_name,
'account_email'=>$account_email,
'pan' =>$pan
);
/************************** SessionAccountInfo Object ********************************/
$mpgSessionAccountInfo = new mpgSessionAccountInfo ($sessionAccountInfoTemplate);
/***************** Transactional Associative Array ********************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'session_id'=>$session_id,
'service_type'=>$service_type
);
/********************** Transaction Object ****************************/
$riskTxn = new riskTransaction($txnArray);
/************************ Set SessionAccountInfo *****************************/
$riskTxn->setSessionAccountInfo($mpgSessionAccountInfo);
/************************ Request Object ******************************/
$riskRequest = new riskRequest($riskTxn);
$riskRequest->setTestMode(true);
/*********************** HTTPS Post Object ****************************/
$riskHttpsPost =new riskHttpsPost($store_id,$api_token,$riskRequest);
/***************************** Response ******************************/
$riskResponse=$riskHttpsPost->getRiskResponse();
//print("\nResponse = " . $riskResponse);
print("\nResponseCode = " . $riskResponse->getResponseCode());
print("\nMessage = " . $riskResponse->getMessage());
$results = $riskResponse->getResults();
foreach($results as $key => $value)
{
print("\n".$key ." = ". $value);
}
$rules = $riskResponse->getRules();
//print_r($rules);
foreach ($rules as $i)
{
foreach ($i as $key => $value)
{
echo "\n$key = $value";
}
}
?>
```

9.3.4.1 Attribute Query Transaction Flow

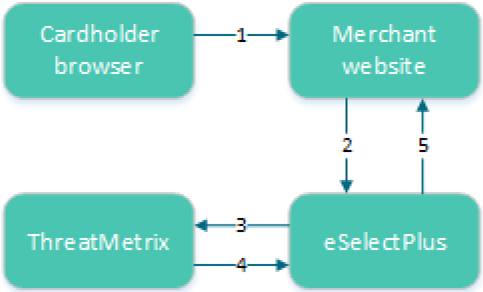


Figure 5: Attribute query transaction flow

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.

9.3.5 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.

9.3.5.1 TRMT Response Fields

Table 89: Receipt object response values for TRMT

Value	Type	Limits	Get method
	Definition		
Response Code	String	3-character alphanumeric	`$mpgResponse->getResponseCode();`
	001 – Success 981 – Data error 982 – Duplicate Order ID 983 – Invalid Transaction 984 – Previously asserted 985 – Invalid activity description 986- Invalid impact description 987 – Invalid Confidence description 988 - Cannot find Previous		
Message	String	N/A	`$mpgResponse->getMessage();`
	Response message		
Event type	String	N/A	
	Type of transaction or event returned in the response.		
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.		

Table 89: Receipt object response values for TRMT (continued)

Value	Type	Limits	Get method
	Definition		
Request ID	String	N/A	
	Unique number and will always be returned with the return request.		
Request result	String	N/A	
	See 9.3.5.1 (page 304).		
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 90: Response code descriptions

Value	Definition
001	Success
981	Data error
982	Duplicate order ID
983	Invalid transaction
984	Previously asserted
985	Invalid activity description

Value	Definition
986	Invalid impact description
987	Invalid confidence description
988	Cannot find previous

Table 91: 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
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	Definition
	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

9.3.5.2 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 92 defines the risk scores ranges.

Table 92: 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.

9.3.5.3 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 93 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 93: 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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
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.	
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

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
	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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
SatelliteProxyISP	PX013	Satellite Proxy
	This transaction is coming from a source that is using a satellite proxy.	
GEO		
DeviceCountriesNotAllowed	GE001	True GEO in Countries Not Allowed blacklist
	This device is connecting from a high-risk geographic location.	
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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
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.	
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.	

Table 93: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
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	

9.3.5.4 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>
<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>
```

Sample Risk Response - Session Query

```
<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>
<RuleCode>DV004</RuleCode>
<RuleMessageEn>No Device ID</RuleMessageEn>
<RuleMessageFr>null</RuleMessageFr>
</Rule>
</receipt>
</response>
```

9.3.6 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>
```

9.4 Encorporating All Available Fraud Tools

- 9.4.1 Implementation Options for TRMT
- 9.4.2 Implementation Checklist
- 9.4.3 Making a Decision

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.

9.4.1 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.

9.4.2 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 94: 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 303) 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.

9.4.3 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.

10 Apple Pay In-App and on the Web Integration

- 10.1 About Apple Pay In-App and on the Web Integration10.1 About Apple Pay In-App and on the Web Integration
- 10.2 About API Integration of Apple Pay
- 10.3 Apple Pay In-App Process Flows10.3 Apple Pay In-App Process Flows
- 10.4 Cavv Purchase – Apple Pay 10.4 Cavv Purchase – Apple Pay
- 10.5 Cavv Pre-Authorization – Apple Pay

10.1 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.

10.2 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.

10.2.1 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:

- 10.4 Cavv Purchase – Apple Pay 10.4 Cavv Purchase – Apple Pay
- 10.5 Cavv Pre-Authorization – Apple Pay

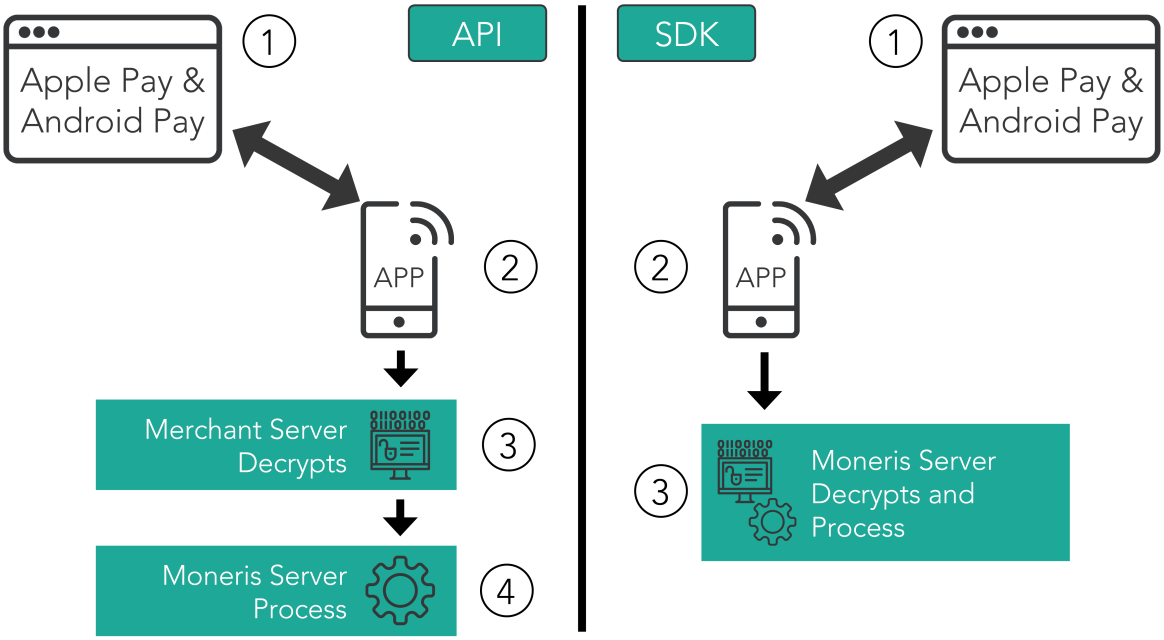
NOTE: INTERAC® e-Commerce functionality is currently available using the Cavv Purchase transaction type only.

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 33)
- Pre-Authorization Completion (page 23)
- Purchase Correction (page 31)

10.3 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 323)Cavv Purchase – Apple Pay (page 323) or Cavv Pre-Authorization – Apple Pay (page 327)Cavv Pre-Authorization – Apple Pay (page 327) 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.

10.4 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 10 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

```
$txnArray = array(‘type’=>’cavv_purchase', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Cavv Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Cavv Purchase transaction values

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

Table 95: Cavv Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character alphanumeric	`'pan'=>$pan`
Expiry date	String	4-character alphanumeric YYMM format	`'expdate'=>$expiry_date`
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	`cavv=>$cavv`
E-commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 1 CavvPurchase transaction object optional values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Card Match ID **NOTE:** Applies to Offlinx™ only; must be unique value for each transaction	*String*	50-character alpha-numeric	`'cm_id' => $transaction_id`
Customer information	Object	N/A	`$mpgTxn->setCustInfo($mpgCustInfo);`
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	
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	

Sample Cavv Purchase for Apple Pay

```
<?php
## Example php -q TestPurchase-VBV.php "moneris" store
require "../../mpgClasses.php";
/******************************* Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/****************************** Transactional Variables ***************************/
$type='cavv_purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="1511";
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$crypt_type = '7';
$wallet_indicator = "APP";
$dynamic_descriptor='123456';
/*************************** Transaction Associative Array ************************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'cavv'=>$cavv,
'crypt_type'=>$crypt_type, //mandatory for AMEX only
//'wallet_indicator'=>$wallet_indicator, //set only for wallet transactions. e.g. APPLE PAY
//'network'=> "Interac", //set only for Interac e-commerce
//'data_type'=> "3DSecure", //set only for Interac e-commerce
'dynamic_descriptor'=>$dynamic_descriptor
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************* Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
```

Sample Cavv Purchase for Apple Pay

```
    print("\nAuthCode = " . $mpgResponse->getAuthCode());
    print("\nComplete = " . $mpgResponse->getComplete());
    print("\nTransDate = " . $mpgResponse->getTransDate());
    print("\nTransTime = " . $mpgResponse->getTransTime());
    print("\nTicket = " . $mpgResponse->getTicket());
    print("\nTimedOut = " . $mpgResponse->getTimedOut());
    print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
    print("\nIssuerId = " . $mpgResponse->getIssuerId());
    ?>
```

10.5 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 10 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

```
$txnArray = array(‘type’=>’cavv_preauth', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Cavv Pre-Authorization transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Cavv Pre-Authorization transaction values

Table 96: Cavv Pre-Authorization object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	'order_id'=>$order_id
Amount	String	(missing or bad snippet)	'amount'=>$amount
Credit card number	String	20-character numeric	'pan'=>$pan
Cardholder Authentication Verification Value (CAVV) **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, CAVV field contains the decrypted cryptogram. For more, see Appendix A Definitions of Request Fields.	String	50-character alphanumeric	cavv=>$cavv
Expiry date	String	4-character numeric	'expdate'=>$expiry_date
E-commerce indicator **NOTE:** For Apple Pay Cavv Purchase and Cavv Pre-Authorization transactions, the E-commerce indicator is a mandatory field containing the value received from the decrypted payload or a default value of 5. If you get a 2-character value (e.g.,. 05 or 07) from the payload, remove the initial 0 and just send us the 2nd character. For more, see Appendix A Definitions of Request Fields.	String	1-character alphanumeric	'crypt_type'=>$crypt

Table 1 Cavv Pre-Authorization object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Card Match ID **NOTE:** Applies to Off-linx™ only; must be unique value for each transaction	*String*	50-character alpha-numeric	`'cm_id' => $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	
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	

Sample Cavv Pre-Authorization for Apple Pay

```
<?php
```

Sample Cavv Pre-Authorization for Apple Pay

```
## Example php -q TestPurchase-VBV.php "moneris" store
require "../../mpgClasses.php";
/******************************* Request Variables *********************************/
$store_id='store5';
$api_token='yesguy';
/****************************** Transactional Variables ****************************/
$type='cavv_preauth';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="0812";
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$crypt_type = '7';
$wallet_indicator = "APP";
$dynamic_descriptor='123456';
/*************************** Transaction Associative Array *************************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'cavv'=>$cavv,
'crypt_type'=>$crypt_type, //mandatory for AMEX only
//'wallet_indicator'=>$wallet_indicator, //set only for wallet transactions. e.g. APPLE PAY
'dynamic_descriptor'=>$dynamic_descriptor
//,'cm_id' => '8nAK8712sGaAkls56' //set only for usage with Offlinx - Unique max 50 alphanumeric
characters transaction id generated by merchant
);
/****************************** Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("U");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/******************************* Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object ********************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************************ Response ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
```

Sample Cavv Pre-Authorization for Apple Pay

```
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

11 Offlinx™

- What Is a Pixel Tag?
- Offlinx™ and API Transactions

11.1 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

11.2 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 – cavv_purchase
- Pre-Authorization with 3-D Secure – cavvPreauth
- Cavv Purchase – Apple Pay
- Cavv Pre-Authorization – Apple Pay

12 Convenience Fee

- 12.1 About Convenience Fee
- 12.2 Purchase with Convenience Fee
- 12.3 Convenience Fee Purchase w/ Customer Information
- 12.4 Convenience Fee Purchase with VbV, MCSC and Amex SafeKey

12.1 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.

12.2 Purchase with Convenience Fee

NOTE: Convenience Fee Purchase with Customer Information is also supported.

Convenience Fee Purchase transaction object definition

```
$txnArray = array(‘type’=>’purchase', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Convenience Fee Purchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

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	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric YYMM format	`'expdate'=>$expiry_date`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`
Convenience fee amount	String	9-character decimal	`$convFeeTemplate = array (convenience_fee=>$convfee_amount);`

Table 2 Convenience Fee Purchase transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
AVS information	Object		`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD information	Object		`$mpgTxn->setCvdInfo ($mpgCvdInfo);`

Sample Purchase with Convenience Fee

```
<?php
/* Moneris Gateway Canada Convenience Fee Account Required this transaction*/
```

Sample Purchase with Convenience Fee

```
require "../../mpgClasses.php";
/************************ Request Variables ***********************************/
$store_id='monca00392';
$api_token='qYdISUhHiOdfTr1CLNpN';
//$status = 'false';
/************************ Transaction Variables *******************************/
$orderid='ord-'.date("dmy-G:i:s");
$amount='10.00';
$pan='4242424242424242';
$expiry_date='1812';
$dynamic_descriptor='test';
/************************ Transaction Array ***********************************/
$txnArray=array(type=>'purchase',
order_id=>$orderid,
cust_id=>'cust',
amount=>$amount,
pan=>$pan,
expdate=>$expiry_date,
crypt_type=>'7',
dynamic_descriptor=>$dynamic_descriptor
);
/********************** ConvFee Associative Array **************************/
$convFeeTemplate = array(
convenience_fee=>'1.00'
);
/************************** ConvFee Object *********************************/
$mpgConvFee = new mpgConvFeeInfo($convFeeTemplate);
/************************ Transaction Object ********************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set ConvFee ******************************/
$mpgTxn->setConvFeeInfo($mpgConvFee);
/************************ Request Object ***********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"CA" for sending transaction to Canadian environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ mpgHttpsPost Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************ Response Object ***********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nISO = " . $mpgResponse->getISO());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
print("\nCfSuccess = " . $mpgResponse->getCfSuccess());
print("\nCfStatus = " . $mpgResponse->getCfStatus());
```

Sample Purchase with Convenience Fee

```
    print("\nFeeAmount = " . $mpgResponse->getFeeAmount());
    print("\nFeeRate = " . $mpgResponse->getFeeRate());
    print("\nFeeType = " . $mpgResponse->getFeeType());
    //print("\nStatusCode = " . $mpgResponse->getStatusCode());
    //print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
    ?>
```

12.3 Convenience Fee Purchase w/ Customer Information

Convenience Fee Purchase with Customer information transaction object definition

```
$txnArray = array(‘type’=>’purchase', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Convenience Fee Purchase with Customer Info transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Convenience Fee Purchase with Customer information transaction object values

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

Table 1 Convenience Fee Purchase w/ Customer Info transaction object mandatory values

Value	Type	Limits	Set Method
Convenience Fee	Object	n/a	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`

Table 1 Convenience Fee Purchase w/ Customer Info transaction object mandatory values (continued)

Value	Type	Limits	Set Method
Expiry date	String	4-character numeric YYMM format	`'expdate'=>$expiry_date`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Convenience fee amount	String	9-character decimal	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`

Table 2 Convenience Fee Purchase w/ Customer Info transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Customer information	Object	n/a	`$mpgTxn->setCustInfo ($mpgCustInfo);`
AVS information	Object	n/a	`$mpgTxn->setAvsInfo ($mpgAvsInfo);`
CVD information	Object	n/a	`$mpgTxn->setCvdInfo ($mpgCvdInfo);`

Sample Convenience Fee Purchase with Customer Information

```
<?php
## Example php -q TestPurchase-CustInfo.php
require "../../mpgClasses.php";
/************************ Request Variables ***************************/
$store_id='monca00392';
$api_token='qYdISUhHiOdfTr1CLNpN';
/********************* Transactional Variables ************************/
$type='purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='my cust id';
$amount='114.28';
$pan='4242424242424242';
```

Sample Convenience Fee Purchase with Customer Information

```
$expiry_date='0812'; //December 2008
$crypt='7';
/******************* Customer Information Variables *********************/
$first_name = 'Cedric';
$last_name = 'Benson';
$company_name = 'Chicago Bears';
$address = '334 Michigan Ave';
$city = 'Chicago';
$province = 'Illinois';
$postal_code = 'M1M1M1';
$country = 'United States';
$phone_number = '453-989-9876';
$fax = '453-989-9877';
$tax1 = '1.01';
$tax2 = '1.02';
$tax3 = '1.03';
$shipping_cost = '9.95';
$email ='Joe@widgets.com';
$instructions ="Make it fast";
/*********************** Line Item Variables ***************************/
$item_name = array();
$item_quantity = array();
$item_product_code = array();
$item_extended_amount = array();
$item_name[0] = 'Guy Lafleur Retro Jersey';
$item_quantity[0] = '1';
$item_product_code[0] = 'JRSCDA344';
$item_extended_amount[0] = '129.99';
$item_name[1] = 'Patrick Roy Signed Koho Stick';
$item_quantity[1] = '1';
$item_product_code[1] = 'JPREEA344';
$item_extended_amount[1] = '59.99';
/******************** Customer Information Object **********************/
$mpgCustInfo = new mpgCustInfo();
/********************** Set Customer Information ***********************/
$billing = array(
'first_name' => $first_name,
'last_name' => $last_name,
'company_name' => $company_name,
'address' => $address,
'city' => $city,
'province' => $province,
'postal_code' => $postal_code,
'country' => $country,
'phone_number' => $phone_number,
'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setBilling($billing);
$shipping = array(
'first_name' => $first_name,
'last_name' => $last_name,
'company_name' => $company_name,
'address' => $address,
'city' => $city,
'province' => $province,
'postal_code' => $postal_code,
```

Sample Convenience Fee Purchase with Customer Information

```
'country' => $country,
'phone_number' => $phone_number,
'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setShipping($shipping);
$mpgCustInfo->setEmail($email);
$mpgCustInfo->setInstructions($instructions);
/*********************** Set Line Item Information *********************/
$item[0] = array(
'name'=>$item_name[0],
'quantity'=>$item_quantity[0],
'product_code'=>$item_product_code[0],
'extended_amount'=>$item_extended_amount[0]
);
$item[1] = array(
'name'=>$item_name[1],
'quantity'=>$item_quantity[1],
'product_code'=>$item_product_code[1],
'extended_amount'=>$item_extended_amount[1]
);
$mpgCustInfo->setItems($item[0]);
$mpgCustInfo->setItems($item[1]);
/********************** ConvFee Associative Array **************************/
$convFeeTemplate = array(
'convenience_fee'=>'2.00'
);
/************************** ConvFee Object *********************************/
$mpgConvFee = new mpgConvFeeInfo($convFeeTemplate);
/***************** Transactional Associative Array *********************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt
);
/********************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************** Set Customer Information *************************/
$mpgTxn->setCustInfo($mpgCustInfo);
/************************ Set ConvFee ******************************/
$mpgTxn->setConvFeeInfo($mpgConvFee);
/************************* Request Object ******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ HTTPS Post Object ****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/****************8********** Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
```

Sample Convenience Fee Purchase with Customer Information

```
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

12.4 Convenience Fee Purchase with VbV, MCSC and Amex SafeKey

Convenience Fee Purchase with VbV/MCSC/SafeKey transaction object definition

```
$txnArray = array(‘type’=>’cavv_purchase', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for Convenience Fee Purchase w/ VbV/MCSC/SafeKey transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Convenience Fee Purchase with VbV/MCSC/SafeKey transaction object values

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

Table 1 Convenience Fee Purchase with VbV, MCSC, SafeKey - Required Fields

Value	Type	Limits	Set Method
Convenience Fee	Object	N/A	`$mpgConvFee = new mpgConvFeeInfo ($convFeeTemplate);`
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snip-pet)	`'amount'=>$amount`
Credit card number	String	20-character numeric	`'pan'=>$pan`
Expiry date	String	4-character numeric YYMM format	`'expdate'=>$expiry_date`

Value	Type	Limits	Set Method
E-Commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`
Cardholder Authentication Verification Value (CAVV)	String	50-character alphanumeric	`cavv=>$cavv`
Convenience fee amount	String	9-character decimal	`$convFeeTemplate = array(convenience_fee=>$convfee_amount);`

Table 2 Convenience Fee Purchase with VbV, MCSC, SafeKey - Optional Values

Value	Type	Limits	Set Method
Status Check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`
E-commerce indicator	String	1-character numeric	`'crypt_type'=>$crypt`
Customer Information	Object	N/A	`$mpgTxn->setCustInfo($mpgCustInfo);`
AVS Information	Object	N/A	`$mpgTxn->setAvsInfo($mpgAvsInfo);`
CVD Information	Object	N/A	`$mpgTxn->setCvdInfo($mpgCvdInfo);`

Sample Purchase with VbV/MCSC/SafeKey

```
<?php
require "../../mpgClasses.php";
```

Sample Purchase with VbV/MCSC/SafeKey

```
/******************************* Request Variables ********************************/
$store_id='monca00392';
$api_token='qYdISUhHiOdfTr1CLNpN';
//$status = 'false';
/****************************** Transactional Variables ***************************/
$type='cavv_purchase';
$order_id="ord-".date("dmy-G:i:s");
$cust_id='customer1';
$amount='1.00';
$pan='4242424242424242';
$expiry_date='0912';
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA';
//$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$commcard_invoice='Invoice 5757FRJ8';
$commcard_tax_amount='1.00';
$crypt_type = '7';
/*************************** Transaction Associative Array ************************/
$txnArray=array(
type=>$type,
order_id=>$order_id,
cust_id=>$cust_id,
amount=>$amount,
pan=>$pan,
expdate=>$expiry_date,
cavv=>$cavv,
commcard_invoice=>$commcard_invoice,
commcard_tax_amount=>$commcard_tax_amount,
crypt_type=>$crypt_type, //mandatory for AMEX only
dynamic_descriptor=>'test'
);
/********************** ConvFee Associative Array *************************/
$convFeeTemplate = array(
convenience_fee=>'1.00'
);
/************************** ConvFee Object ********************************/
$mpgConvFee = new mpgConvFeeInfo($convFeeTemplate);
/****************************** Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set ConvFee *****************************/
$mpgTxn->setConvFeeInfo($mpgConvFee);
/******************************* Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"CA" for sending transaction to Canadian environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/****************************** HTTPS Post Object *******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
//Status check example
//$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);
/************************************* Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
```

Sample Purchase with VbV/MCSC/SafeKey

```
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
print("\nCavvResultCode = " . $mpgResponse->getCavvResultCode());
print("\nCfSuccess = " . $mpgResponse->getCfSuccess());
print("\nCfStatus = " . $mpgResponse->getCfStatus());
print("\nFeeAmount = " . $mpgResponse->getFeeAmount());
print("\nFeeRate = " . $mpgResponse->getFeeRate());
print("\nFeeType = " . $mpgResponse->getFeeType());
//print("\nStatusCode = " . $mpgResponse->getStatusCode());
//print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

13 Recurring Billing

- 13.1 About Recurring Billing
- 13.2 Purchase with Recurring Billing
- 1 Purchase with Recurring Billing
- 13.3 Recurring Billing Update
- 1 Recurring Billing Update
- 13.4 Recurring Billing Response Fields and Codes
- 13.5 Credential on File and Recurring Billing

13.1 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.

13.2 Purchase with Recurring Billing

Recurring Billing Info Object Definition

```
$recurArray = array(

'recur_unit'=>$recurUnit, // (day | week | month)

'start_date'=>$startDate, //yyyy/mm/dd

'num_recurs'=>$numRecurs,

'start_now'=>$startNow,

'period' => $recurInterval,

'recur_amount'=> $recurAmount

);

$mpgRecur = new mpgRecur($recurArray);
```

Transaction object set method

```
$mpgTxn->setRecur($mpgRecur);
```

Recurring Billing Info Object Request Fields

Variable and Field 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 units that must pass between recurring billings
Start Date `start_date`	*String* YYYY/MM/DD	Date of the first future recurring billing transaction This value **must** be a date in the future If an additional charge is to be made immediately, the value of Start Now must be set to true
Start Now `start_now`	*String* true/false	If a single charge is to be made against the card immediately, set this value to true; the amount to be billed immediately may differ from the amount billed on a regular basis thereafter If the billing is to start in the future, set this value to false When set to false, use Card Verification prior to sending the Purchase with Recur and Credential on File objects
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	Amount of the recurring transaction This is the amount that will be billed on the Start Date and then billed repeatedly based on the interval defined by Period and Recur Unit

Variable and Field Name	Type and Limits	Description
	EXAMPLE: 1234567.89	
Recur Unit `recur_unit`	*String* day, week, month or eom	Unit to be used as a basis for the interval Works in conjunction with Period to define the billing frequency Possible values are: day week month eom (end of month)

Sample Purchase with Recurring Billing

```
<?php
##
## Example php -q TestPurchase-Recur.php store3 yesguy unique_order_id
##
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id = 'store5';
$api_token = 'yesguy';
/********************************* Recur Variables *****************************/
$recurUnit = 'eom';
$startDate = '2018/11/30';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';
/************************* Transactional Variables *****************************/
$orderId = 'ord-'.date("dmy-G:i:s");
$custId = 'student_number';
$creditCard = '5454545454545454';
$nowAmount = '10.00';
$expiryDate = '0912';
$cryptType = '7';
/*********************** Recur Associative Array ***********************/
$recurArray = array('recur_unit'=>$recurUnit, // (day | week | month)
'start_date'=>$startDate, //yyyy/mm/dd
'num_recurs'=>$numRecurs,
'start_now'=>$startNow,
'period' => $recurInterval,
'recur_amount'=> $recurAmount
);
$mpgRecur = new mpgRecur($recurArray);
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>'purchase',
```

Sample Purchase with Recurring Billing

```
'order_id'=>$orderId,
'cust_id'=>$custId,
'amount'=>$nowAmount,
'pan'=>$creditCard,
'expdate'=>$expiryDate,
'crypt_type'=>$cryptType
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Recur Object *********************************/
$mpgTxn->setRecur($mpgRecur);
/******************* Credential on File **********************************/
$cof = new CofInfo();
$cof->setPaymentIndicator("R");
$cof->setPaymentInformation("2");
$cof->setIssuerId("168451306048014");
$mpgTxn->setCofInfo($cof);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print ("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nRecurSuccess = " . $mpgResponse->getRecurSuccess());
print("\nIssuerId = " . $mpgResponse->getIssuerId());
?>
```

13.3 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

```
$txnArray=array('type'=>'recur_update',... );
```

HttpsPostRequest object for Recurring Billing Update transaction

```
$mpgTxn = new mpgTransaction($txnArray);

$mpgRequest = new mpgRequest($mpgTxn);
```

Recurring Billing Update transaction values

Table 1 Recurring Billing Update – Basic Required Fields

Variable and Field Name	Type and Limits	Set Method
Order ID `order_id`	*String* 50-character alphanumeric	`'cust_id'=>$cust_id`

Table 2 Recurring Billing Update – Basic Optional Fields

Variable and Field Name	Type and Limits	Set Method
Customer ID `cust_id`	*String* 50-character alphanumeric	`'cust_id'=>$cust_id`
Credit card number `pan`	*String* 20-character alphanumeric	`'pan'=>$pan`
Expiry date `expdate`	*String* YYMM	`'expdate'=>$expiry_date`

Table 3 Recurring Billing Update – Recurring Billing Required Fields

Variable and Field 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 **EXAMPLE-:** 1234567.-89	`'recur_amount'=>$recur_amount`	Changes the amount that is billed recurrently The change takes effect on the next charge
Add number of recurs `add_num`	*String* numeric, 1-999	`'add_num_recurs' => $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	`'total_num_recurs' => $total_num`	**Replaces** the current (remaining) number of recurring transactions
Hold recurring billing `hold`	*String* true/false	`'hold' => $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	`'terminate' => $terminate`	Terminates recurring billing **NOTE:** After it has

Variable and Field Name	Type and Limits	Set Method	Description
			been terminated, a recurring transaction **cannot** be reactivated; a new purchase transaction with recurring billing must be submitted.

Sample Recurring Billing Update

```
<?php
##
## Example php -q TestRecurUpdate.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store5';
$api_token='yesguy';
/************************* Transactional Variables *****************************/
$type='recur_update';
$cust_id='my cust id';
$order_id='ord-110515-10:45:21';
$recur_amount='1.00';
$pan='4242424242424242';
$expiry_date='1811';
$add_num='';
$total_num='7';
$hold = 'false';
$terminate = 'false';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'recur_amount'=>$recur_amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'add_num_recurs' => $add_num,
'total_num_recurs' => $total_num,
'hold' => $hold,
'terminate' => $terminate
);
/******************* Credential on File ***********************************/
$cof = new CofInfo();
$cof->setIssuerId("168451306048014");
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCofInfo($cof);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response *************************************/
```

Sample Recurring Billing Update

```
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nRecurUpdateSuccess = " . $mpgResponse->getRecurUpdateSuccess());
print("\nNextRecurDate = " . $mpgResponse->getNextRecurDate());
print("\nRecurEndDate = " . $mpgResponse->getRecurEndDate());
?>
```

13.4 Recurring Billing Response Fields and Codes

Table 97 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

`$mpgResponse=$mpgHttpPost->getMpgResponse();`

Table 97: Recurring Billing response fields

Value	Type	Limits	Get method
	Description		
Transaction object with Recurring Billing response fields			
Response code	String	3-character numeric	`$mpgResponse->getResponseCode()`
	See Table 98: for a description of possible response codes.		
Recur success	String	TBD	`$mpgResponse->getRecurSuccess()`
	Indicates whether the transaction successfully registered		
Recur update object response fields			
Recur update success	String	`true/false`	`$mpgResponse->getRecurUpdateSuccess()`
	Indicates whether the transaction successfully updated.		
Next recur date	String	yyyy-mm-dd format	`$mpgResponse->getNextRecurDate()`
	Indicates when the transaction will be billed again.		
Recur end date	String	yyyy-mm-dd format	`$mpgResponse->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 98: 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

13.5   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.

14 Customer Information

- 14.1 Using the Customer Information Object
- 14.2 Customer Information Sample Code

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.

14.1 Using the Customer Information Object

- 14.1.1 Customer Info Object – Miscellaneous Properties
- 14.1.2 Customer Info Object – Billing/Shipping Information
- 14.1.3 Customer Info Object – Item Information

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

```
$mpgCustInfo = new mpgCustInfo();
```

Transaction object set method

```
$mpgTxn->setCustInfo($mpgCustInfo);
```

14.1.1 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 99: CustInfo object miscellaneous properties

Value	Type	Limits	Set method
Email Address	String	60-character alphanumeric	`$mpgCustInfo->setEmail($email);`
Instructions	String	100-character alphanumeric	`$mpgCustInfo->setInstructions ($note);`

14.1.2 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 100: 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"

Table 100: Billing and shipping information values (continued)

Value	Limit	Hash table key
City	30	"city"
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"

14.1.2.1 Set Methods for Billing and Shipping Info

The billing information and the shipping information for a given CustInfo object are written by using the `$mpgCustInfo->setBilling($billing);` and `$mpgCustInfo->setShipping($shipping);` methods respectively:

```
$billing = array(

'first_name' => $first_name,

'last_name' => $last_name,

'company_name' => $company_name,

'address' => $address,

'city' => $city,

'province' => $province,

'postal_code' => $postal_code,

'country' => $country,

'phone_number' => $phone_number,

'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setBilling($billing);
$shipping = array(
'first_name' => $first_name,
'last_name' => $last_name,
'company_name' => $company_name,
'address' => $address,
'city' => $city,
'province' => $province,
'postal_code' => $postal_code,
'country' => $country,
'phone_number' => $phone_number,
'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setShipping($shipping);
```

Both of these methods have the same set of mandatory arguments. They are described in the Billing and shipping information values table in 14.1.2.1 Set Methods for Billing and Shipping Info.

For sample code, see 14.2 Customer Information Sample Code.

14.1.2.2 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 14.1.2 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 14.2 Customer Information Sample Code.

14.1.3 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 101: 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

14.1.3.1 Set Methods for Item Information

All the item information found in the Item information values table in 14.1.3 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 14.2 Customer Information Sample Code.

14.1.3.2 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 14.1.3 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 14.2 Customer Information Sample Code.

14.2 Customer Information Sample Code

Below is an example of a Basic Purchase with Customer Information transaction.

Note that the two items ordered are represented by four arrays, and the billing and shipping details are the same.

Sample Purchase with Customer Information

```
## Example php -q TestPurchase-CustInfo.php
require "../../mpgClasses.php";
/************************ Request Variables ***************************/
$store_id='store5';
$api_token='yesguy';
/********************* Transactional Variables ************************/
$type='purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='my cust id';
$amount='1.00';
$pan='4242424242424242';
$expiry_date='0812'; //December 2008
$crypt='7';
/******************* Customer Information Variables ********************/
$first_name = 'Cedric';
$last_name = 'Benson';
$company_name = 'Chicago Bears';
$address = '334 Michigan Ave';
$city = 'Chicago';
$province = 'Illinois';
$postal_code = 'M1M1M1';
$country = 'United States';
$phone_number = '453-989-9876';
$fax = '453-989-9877';
$tax1 = '1.01';
$tax2 = '1.02';
$tax3 = '1.03';
$shipping_cost = '9.95';
$email ='Joe@widgets.com';
$instructions ="Make it fast";
```

Sample Purchase with Customer Information

```
/*********************** Line Item Variables ***************************/
$item_name[0] = 'Guy Lafleur Retro Jersey';
$item_quantity[0] = '1';
$item_product_code[0] = 'JRSCDA344';
$item_extended_amount[0] = '129.99';
$item_name[1] = 'Patrick Roy Signed Koho Stick';
$item_quantity[1] = '1';
$item_product_code[1] = 'JPREEA344';
$item_extended_amount[1] = '59.99';
/******************** Customer Information Object **********************/
$mpgCustInfo = new mpgCustInfo();
/********************** Set Customer Information ***********************/
$billing = array(
'first_name' => $first_name,
'last_name' => $last_name,
'company_name' => $company_name,
'address' => $address,
'city' => $city,
'province' => $province,
'postal_code' => $postal_code,
'country' => $country,
'phone_number' => $phone_number,
'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setBilling($billing);
$shipping = array(
'first_name' => $first_name,
'last_name' => $last_name,
'company_name' => $company_name,
'address' => $address,
'city' => $city,
'province' => $province,
'postal_code' => $postal_code,
'country' => $country,
'phone_number' => $phone_number,
'fax' => $fax,
'tax1' => $tax1,
'tax2' => $tax2,
'tax3' => $tax3,
'shipping_cost' => $shipping_cost
);
$mpgCustInfo->setShipping($shipping);
$mpgCustInfo->setEmail($email);
$mpgCustInfo->setInstructions($instructions);
/*********************** Set Line Item Information **********************/
$item[0] = array(
'name'=>$item_name[0],
'quantity'=>$item_quantity[0],
'product_code'=>$item_product_code[0],
'extended_amount'=>$item_extended_amount[0]
);
$item[1] = array(
'name'=>$item_name[1],
'quantity'=>$item_quantity[1],
'product_code'=>$item_product_code[1],
```

Sample Purchase with Customer Information

```
'extended_amount'=>$item_extended_amount[1]
);
$mpgCustInfo->setItems($item[0]);
$mpgCustInfo->setItems($item[1]);
/***************** Transactional Associative Array ********************/
$txnArray=array(
'type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt
);
/********************** Transaction Object ****************************/
$mpgTxn = new mpgTransaction($txnArray);
/******************** Set Customer Information ************************/
$mpgTxn->setCustInfo($mpgCustInfo);
/************************* Request Object *****************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/************************ HTTPS Post Object ***************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/*************************** Response *********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

15 Status Check

- 15.1 About Status Check
- 15.2 Using Status Check Response Fields
- 15.3 Sample Purchase with Status Check

15.1 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.

15.2 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.

15.3 Sample Purchase with Status Check

Sample Purchase transaction with Status Check

```
<?php
require "../../mpgClasses.php";
$store_id='store5';
$api_token='yesguy';
$status_check = 'true';

$txnArray=array('type'=>'purchase',
    'order_id'=>'order',
    'cust_id'=>'cust',
    'amount'=>'1.00',
    'pan'=>'4242424242424242',
    'expdate'=>'2202',
    'crypt_type'=>'1',
    'dynamic_descriptor'=>''
);

$mpgTxn = new mpgTransaction($txnArray);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA");
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);

$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nStatusCode = " . $mpgResponse->getStatusCode());
print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

16 Visa Checkout

- 16.1 About Visa Checkout
- 16.2 Transaction Types - Visa Checkout
- 16.3 Integrating Visa Checkout Lightbox
- 16.4 Transaction Flow for Visa Checkout
- 16.5 Visa Checkout Purchase
- 16.6 Visa Checkout Pre-Authorization
- 16.7 Visa Checkout Completion
- 16.8 Visa Checkout Purchase Correction
- 16.9 Visa Checkout Refund
- 16.10 Visa Checkout Information

16.1 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.

16.2 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.

16.3 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

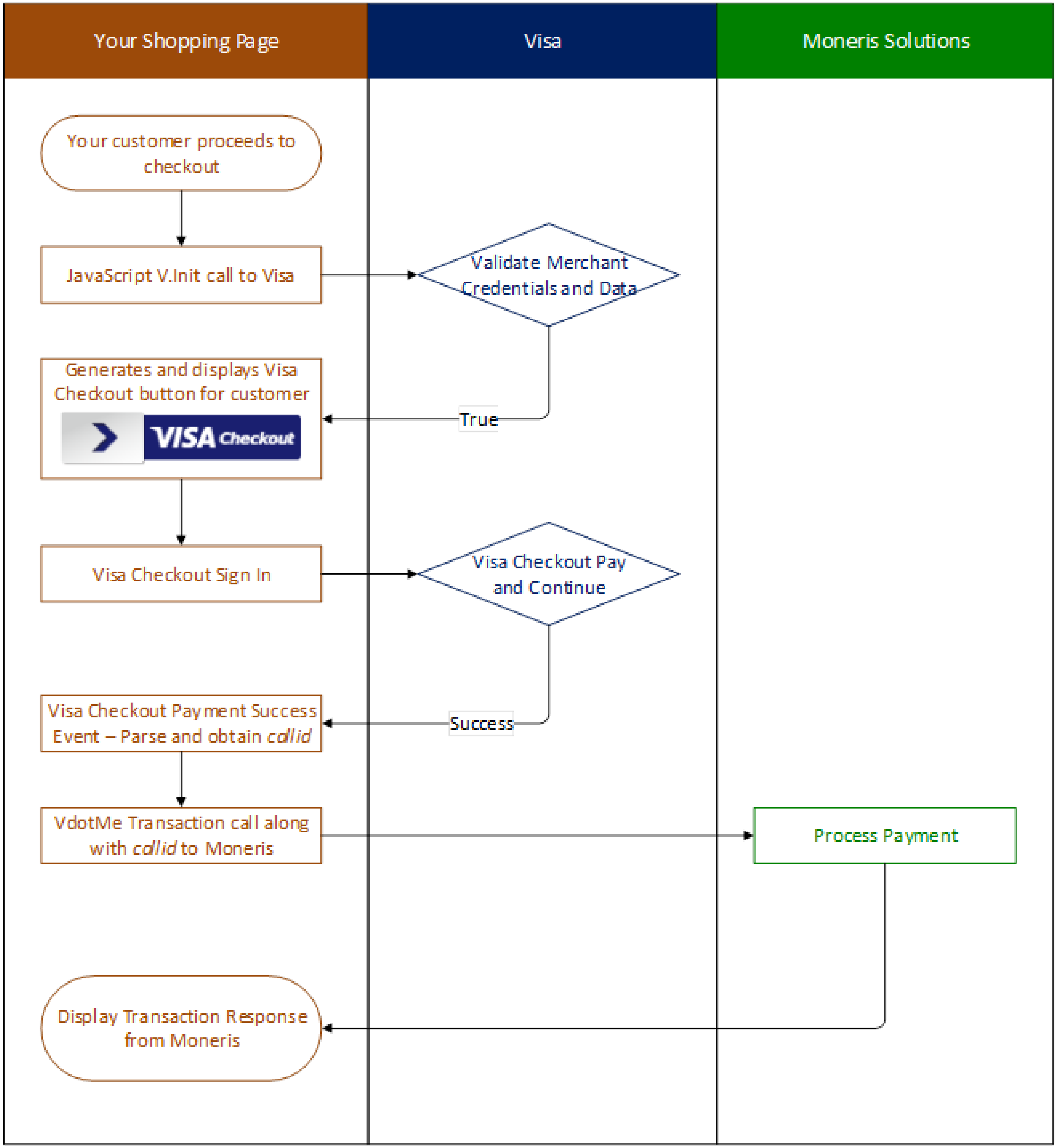
2. If you get a payment success event from the resulting Visa Lightbox JavaScript, you will have to parse and obtain the `callid` from their JSON response. The additional information is obtained using `VdotMeInfo`.

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.

16.4 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

16.5 Visa Checkout Purchase

VdotMePurchase transaction object definition

```
$txnArray = array(‘type’=>’vdotme_purchase', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest for VdotMePurchase transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMePurchase transaction object values

Table 1 VdotMePurchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Call ID	String	20-character numeric	`'callid'=>$callid`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 2 VdotMePurchase transaction object optional values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample VdotMePurchase

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
```

Sample VdotMePurchase

```
/**************************** Request Variables *******************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='vdotme_purchase';
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$callid = '2040321768994339501';
$crypt='7';
$dynamic_descriptor='123';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'callid'=>$callid,
'crypt_type'=>$crypt,
'cust_id'=>$cust_id,
'dynamic_descriptor'=>$dynamic_descriptor
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

16.6 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

```
$txnArray = array(‘type’=>’vdotme_preauth', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VdotMePreAuth transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMePreAuth transaction object values

Table 1 VdotMePreAuth transaction object mandatory values

Value	Type	Limits	Set Method
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Call ID	String	20-character numeric	`'callid'=>$callid`
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 2 VdotMePreAuth transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample VdotMePreAuth

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='vdotme_preauth';
```

Sample VdotMePreAuth

```
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$callid = '7019571968382473715';
$crypt='7';
$dynamic_descriptor='123';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'amount'=>$amount,
'callid'=>$callid,
'crypt_type'=>$crypt,
'cust_id'=>$cust_id,
'dynamic_descriptor'=>$dynamic_descriptor
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

16.7 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

```
$txnArray = array('type'=>'vdotme_completion', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VdotMeCompletion transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMeCompletion transaction object values

Table 1 VdotMeCompletion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`
Completion amount	String	(missing or bad snippet)	`'comp_amount'=>$comp_amount`
E-commerce indicator	String	1-character alphanumeric	`'crypt_type'=>$crypt`

Table 2 VdotMeCompletion transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alphanumeric	`'dynamic_descriptor'=>$dynamic_descriptor`

Sample VdotMeCompletion

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables ***************************/
$type='vdotme_completion';
$cust_id='cust id';
$order_id='ord-110515-16:01:19';
$comp_amount='0.10';
$txn_number = '721358-0_10';
$crypt='7';
```

Sample VdotMeCompletion

```
$dynamic_descriptor='123';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'comp_amount'=>$comp_amount,
'txn_number'=>$txn_number,
'crypt_type'=>$crypt,
'cust_id'=>$cust_id,
'dynamic_descriptor'=>$dynamic_descriptor
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

16.8 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

```
$txnArray = array(‘type’=>’vdotme_purchasecorrection', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VdotMePurchaseCorrection transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMePurchaseCorrection transaction object values

Table 1 VdotMePurchaseCorrection transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`'order_id'=>$order_id`
Transaction number	String	255-character alpha-numeric	`'txn_number'=>$txn_number`

Table 2 VdotMePurchaseCorrection transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Status check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample VdotMePurchaseCorrection

```php
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables ********************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables *****************************/
$type='vdotme_purchasecorrection';
$cust_id='cust id';
$order_id='ord-110515-15:58:00';
$txn_number = '721355-0_10';
$crypt='7';
/*********************** Transactional Associative Array ***********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=>$txn_number,
'crypt_type'=>$crypt,
'cust_id'=>$cust_id,
);
/**************************** Transaction Object ******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object ********************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

Sample VdotMePurchaseCorrection

```
/******************************* Response ************************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

16.9 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

```
$txnArray = array('type'=>'vdotme_refund', …);
```

```
$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VdotMeRefund transaction

```
$mpgRequest = new mpgRequest($mpgTxn);
```

```
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMeRefund transaction object values

Table 1 VdotMeRefund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	`'order_id'=>$order_id`
Amount	String	(missing or bad snippet)	`'amount'=>$amount`
Transaction number	String	255-character alphanumeric	`'txn_number'=>$txn_number`

Value	Type	Limits	Set Method
E-commerce indicator	String	1-character alpha-numeric	`'crypt_type'=>$crypt`

Table 2 VdotMeRefund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`'cust_id'=>$cust_id`
Dynamic descriptor	String	20-character alpha-numeric	`'dynamic_descriptor'=>$dynamic_descriptor`
Status check	Boolean	true/false	`$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);`

Sample VdotMeRefund

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$type='vdotme_refund';
$cust_id='cust id';
$order_id='ord-110515-16:01:19';
$txn_number = '721359-1_10';
$amount = '0.05';
$crypt='7';
$dynamic_descriptor='123';
/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'txn_number'=>$txn_number,
'amount'=>$amount,
'crypt_type'=>$crypt,
'cust_id'=>$cust_id,
'dynamic_descriptor'=>$dynamic_descriptor
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
```

Sample VdotMeRefund

```
/***************************** HTTPS Post Object *****************************/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/****************************** Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

16.10 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

```
$txnArray = array(‘type’=>’vdotme_getpaymentinfo', …);

$mpgTxn = new mpgTransaction($txnArray);
```

HttpsPostRequest object for VdotMeInfo transaction

```
$mpgRequest = new mpgRequest($mpgTxn);

$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
```

VdotMeInfo transaction object values

Table 1 VdotMeInfo transaction object mandatory values

Value	Type	Limits	Set Method
Call ID	String	20-character numeric	`'callid'=>$callid`

Sample VdotMeInfo

```
<?php
##
## Example php -q TestPurchase.php store1
```

Sample VdotMeInfo

```
##
require "../../mpgClasses.php";
/**************************** Request Variables *******************************/
$store_id='store2';
$api_token='yesguy';
/************************* Transactional Variables ****************************/
$callid='8620484083629792701';
/*********************** Transactional Associative Array **********************/
$txnArray=array(type=>'vdotme_getpaymentinfo',
'callid'=>$callid
);
/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Request Object *******************************/
$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to US environment
$mpgRequest->setTestMode(true); //false or comment out this line for production transactions
/***************************** HTTPS Post Object *****************************/
/* Status Check Example
$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
*/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/******************************* Response ************************************/
$vdotmeinfo=$mpgHttpPost->getMpgResponse();
print("\nResponse Code: " . $vdotmeinfo->getResponseCode());
print("\nResponse Message: " . $vdotmeinfo->getMessage());
print("\nCurrency Code: " . $vdotmeinfo->getCurrencyCode());
print("\nPayment Totals: " . $vdotmeinfo->getPaymentTotal());
print("\nUser First Name: " . $vdotmeinfo->getUserFirstName());
print("\nUser Last Name: " . $vdotmeinfo->getUserLastName());
print("\nUsername: " . $vdotmeinfo->getUserName());
print("\nUser Email: " . $vdotmeinfo->getUserEmail());
print("\nEncrypted User ID: " . $vdotmeinfo->getEncUserId());
print("\nCreation Time Stamp: " . $vdotmeinfo->getCreationTimeStamp());
print("\nName on Card: " . $vdotmeinfo->getNameOnCard());
print("\nExpiration Month: " . $vdotmeinfo->getExpirationDateMonth());
print("\nExpiration Year: " . $vdotmeinfo->getExpirationDateYear());
print("\nLast 4 Digits: " . $vdotmeinfo->getLastFourDigits());
print("\nBin Number (6 Digits): " . $vdotmeinfo->getBinSixDigits());
print("\nCard Brand: " . $vdotmeinfo->getCardBrand());
print("\nCard Type: " . $vdotmeinfo->getVDotMeCardType());
print("\nBilling Person Name: " . $vdotmeinfo->getBillingPersonName());
print("\nBilling Address Line 1: " . $vdotmeinfo->getBillingAddressLine1());
print("\nBilling City: " . $vdotmeinfo->getBillingCity());
print("\nBilling State/Province Code: " . $vdotmeinfo->getBillingStateProvinceCode());
print("\nBilling Postal Code: " . $vdotmeinfo->getBillingPostalCode());
print("\nBilling Country Code: " . $vdotmeinfo->getBillingCountryCode());
print("\nBilling Phone: " . $vdotmeinfo->getBillingPhone());
print("\nBilling ID: " . $vdotmeinfo->getBillingId());
print("\nBilling Verification Status: " . $vdotmeinfo->getBillingVerificationStatus());
print("\nPartial Shipping Country Code: " . $vdotmeinfo->getPartialShippingCountryCode());
print("\nPartial Shipping Postal Code: " . $vdotmeinfo->getPartialShippingPostalCode());
print("\nShipping Person Name: " . $vdotmeinfo->getShippingPersonName());
print("\nShipping Address Line 1: " . $vdotmeinfo->getShippingAddressLine1());
print("\nShipping City: " . $vdotmeinfo->getShippingCity());
print("\nShipping State/Province Code: " . $vdotmeinfo->getShippingStateProvinceCode());
print("\nShipping Postal Code: " . $vdotmeinfo->getShippingPostalCode());
print("\nShipping Country Code: " . $vdotmeinfo->getShippingCountryCode());
print("\nShipping Phone: " . $vdotmeinfo->getShippingPhone());
```

Sample VdotMeInfo

```
print("\nShipping Default: " . $vdotmeinfo->getShippingDefault());
print("\nShipping ID: " . $vdotmeinfo->getShippingId());
print("\nShipping Verification Status: " . $vdotmeinfo->getShippingVerificationStatus());
print("\nisExpired: " . $vdotmeinfo->getIsExpired());
print("\nBase Image File Name: " . $vdotmeinfo->getBaseImageFileName());
print("\nHeight: " . $vdotmeinfo->getHeight());
print("\nWidth: " . $vdotmeinfo->getWidth());
print("\nIssuer Bid: " . $vdotmeinfo->getIssuerBid());
print("\nRisk Advice: " . $vdotmeinfo->getRiskAdvice());
print("\nRisk Score: " . $vdotmeinfo->getRiskScore());
print("\nAVS Response Code: " . $vdotmeinfo->getAvsResponseCode());
print("\nCVV Response Code: " . $vdotmeinfo->getCvvResponseCode());
?>
```

17 Testing a Solution

- 17.1 About the Merchant Resource Center
- 17.2 Logging In to the QA Merchant Resource Center
- 17.3 Test Credentials for Merchant Resource Center
- 17.4 Getting a Unique Test Store ID and API Token
- 17.5 Processing a Transaction
- 17.6 Testing INTERAC® Online Payment Solutions
- 17.7 Testing MPI Solutions
- 17.8 Testing Visa Checkout
- 1 Test Cards
- 17.10 Simulator Host

17.1 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.

17.2 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 379)

17.3 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:

```
$store_id='monca00392';

$api_token='qYdISUhHiOdfTr1CLNpN';

$mpgRequest->setProcCountryCode("CA");

$mpgRequest->setTestMode(true);
```

Table 102: 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 381)

17.4 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 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 379).

17.5 Processing a Transaction

- 1.1 Overview
- 1.2 HttpsPostRequest Object
- 1.3 Receipt Object

17.5.1 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 17.5

   Section 17.5 (page 383) 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 PHP 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.

```
<?php
##
## Example php -q TestPurchase.php store1
##
require "../mpgClasses.php";
```

Include all necessary classes.

```
$type='purchase';
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='4242424242424242';
$expiry_date='1111';
$crypt='7';
```

Define all mandatory values for the transaction object properties.

```
$store_id='store5';
$api_token='yesguy';
```

Define all mandatory values for the connection object properties.

```
$txnArray=array('type'=>$type,
'order_id'=>$order_id,
'cust_id'=>$cust_id,
'amount'=>$amount,
'pan'=>$pan,
'expdate'=>$expiry_date,
'crypt_type'=>$crypt,
'dynamic_descriptor'=>$dynamic_descriptor
);

$mpgTxn = new mpgTransaction($txnArray);

$mpgRequest = new mpgRequest($mpgTxn);
$mpgRequest->setProcCountryCode("CA"); //"US" for sending transaction to
US environment
$mpgRequest->setTestMode(true); //false or comment out this line for
production transactions
```

Instantiate the transaction object and assign values to properties.

```
/* Status Check Example
$mpgHttpPost =new mpgHttpsPostStatus($store_id,$api_token,$status_
check,$mpgRequest);
*/
$mpgHttpPost =new mpgHttpsPost($store_id,$api_token,$mpgRequest);



$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nStatusCode = " . $mpgResponse->getStatusCode());
print("\nStatusMessage = " . $mpgResponse->getStatusMessage());
?>
```

Instantiate connection object and assign values to properties, including the transaction object you just created.

Instantiate the Receipt object and use its get methods to retrieve the desired response data.

17.5.2 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 103: HttpsPostRequest object mandatory values

Value	Type	Limits	Set method
	Description		
Processing country code	String	2-character alphabetic	$mpgRequest->setProcCountryCode ("CA");
	CA for Canada, US for USA.		
Test mode	Boolean	true/false	$mpgRequest->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	$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
	Unique identifier provided by Moneris upon merchant account set up. See 17.1 About the Merchant Resource Center for test environment details.		
API Token	String	20-character alphanumeric	$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
	Unique alphanumeric string assigned upon merchant account activation. To locate your production API token, refer to the Merchant Resource Center Admin Store Settings. See 17.3 Test Credentials for Merchant Resource Center for test environment details.		
Transaction	Object	Not applicable	$mpgRequest = new mpgRequest ($mpgTxn);
	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	$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
	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.		

17.5.3 Receipt Object

After you send a transaction using the HttpsPostRequest object's send method, you can instantiate a receipt object.

Receipt Object Definition

```
$mpgResponse=$mpgHttpPost->getMpgResponse();
```

For an in-depth explanation of Receipt object methods and properties, see Appendix B Definitions of Response Fields.

17.6 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 458) and Appendix F (page 462).

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.

17.7 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 104: 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.
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 105: 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.

17.8 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 365.
3. For test card numbers specifically for use when testing Visa Checkout, see "Test Cards for Visa Checkout" on the next page

17.8.1 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.

17.9 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:

Table 106: General test card numbers

Card Plan	Card Number
Mastercard	5454545454545454
Visa	4242424242424242
Amex	373599005095005
JCB	3566007770015365

Card Plan	Card Number
Diners	36462462742008
Track2	5258968987035454=06061015454001060101?
Discover	6510000000000182
UnionPay	6250944000000771

17.9.1 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

17.9.2 Test Cards for Visa Checkout

Table 1 Test Cards Numbers – Visa Checkout

Card Plan	Card Number
Visa	4005520201264821 (without card art)
Visa	4242424242424242 (with card art)
MasterCard	5500005555555559
American Express	340353278080900
Discover	6011003179988686

17.10 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.

18 Moving to Production

- 18.1 Activating a Production Store Account
- 18.2 Configuring a Store for Production
- 18.3 Receipt Requirements
- 1 Getting Help

18.1 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 18.1.
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 393)

> **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.

18.2 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 393).
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: `$mpgRequest->setTestMode(true);`	Set method for production: `$mpgRequest->setTestMode(false);`

2	String:
	`$store_id='store5';`
	Associated Set Method:
	`'store_id'=>$store_id`
		String for Production:
		`$store_id='monXXXXXXXX';`

3	String:
	`$api_token='yesguy';`
	Associated Set Method:
	`'api_token'=>$api_token`
		String for Production:
		`$api_token='XXXX';`

18.2.1 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/

18.2.1.1 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.

18.2.1.2 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 105 for additional client requirements.

18.3 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.

18.3.1 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 383.

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 107: Request fields

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Order ID	String	50-character alphanumeric	`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 107: 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	`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)	`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)	`expdate`
	Note: This is the reverse of the date displayed on the physical card, which is MMYY.		

Table 107: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
E-Commerce indicator	String	1-character alpha-numeric	`crypt_type`
	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) 8: Non-secure transaction (web- or email-based) 9: SET non-authenticated transaction **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 107: 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	`comp_amount`	Amount of a Completion transaction. This may not be equal to the amount value (described on page 397), which appeared in the original Pre-Authorization transaction.

Shipping Indicator[1]	String	1-character alpha-numeric	`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 107: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Transaction number	String	255-character alphanumeric	`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	`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	`ecr_number`
	Electronic cash register number, also referred to as TID or Terminal ID.		
MPI transaction values			
XID	String	20-character alphanumeric	`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	`MD`
	Information to be echoed back in the response.		
Merchant URL	String	Variable length	`merchantUrl`
	URL to which the MPI response is to be sent.		
Accept	String	Variable length	`accept`
	MIME types that the browser accepts		
User Agent	String	Variable length	`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 107: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Cardholder Authentication Verification Value (CAVV)	String	50-character alpha-numeric	`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	`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	`duration`
	Amount of time the temporary token should be available, up to 900 seconds.		

Table 107: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Data key format[1]	String	2-character alpha-numeric	`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	`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	`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	`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 107: Request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Device type	String	30-character alpha-numeric	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 108: Optional transaction values

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Customer ID	String	30-character alphanumeric	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	status_check
	See .		
Dynamic descriptor	String	20-character alphanumeric Combined with merchant's business name cannot exceed 25 characters.	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 108: Optional transaction values (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Wallet indicator[1]	String	3-character alphanumeric	`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	`phone`
	Phone number of the customer. Can be sent in when creating or updating a Vault profile.		
Email address	String	30-character alphanumeric	`email`
	Email address of the customer. Can be sent in when creating or updating a Vault profile.		
Additional notes	String	30-character alphanumeric	`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 14 Customer Information

[1] Available to Canadian integrations only.

For information about Address Verification Service (AVS) request fields see 9.1 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 Definitions of Request Fields – Credential on File

Variable Name	Type	Limits	Description
Issuer ID **NOTE:** This variable is required for all merchant-intiated 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, you must save the Issuer ID on your system to use in subsequent Credential on File transactions (applies to merchant-initiated transactions only)
Payment Indicator	String	1-character alphabetic	Indicates the intended or current use of the credentials Possible values for first transactions: C - unscheduled credential on file (first transaction only) R - recurring Possible values for subsequent transactions: R - recurring U - unscheduled merchant-initiated transaction Z - unscheduled cardholder-initiated transaction
Payment Information	String	1-character numeric	Describes whether the transaction is the first or subsequent in the series Possible values are: 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 and Field 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 units that must pass between recurring billings
Start Date `start_date`	*String* YYYY/MM/DD	Date of the first future recurring billing transaction This value **must** be a date in the future If an additional charge is to be made immediately, the value of Start Now must be set to true
Start Now `start_now`	*String* true/false	If a single charge is to be made against the card immediately, set this value to true; the amount to be billed immediately may differ from the amount billed on a regular basis thereafter If the billing is to start in the future, set this value to false When set to false, use Card Verification prior to sending the Purchase with Recur and Credential on File objects
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	Amount of the recurring transaction This is the amount that will be billed on the Start Date and then billed repeatedly based on the interval defined by Period and Recur Unit

Variable and Field Name	Type and Limits	Description
	EXAMPLE: 1234567.89	
Recur Unit `recur_unit`	*String* day, week, month or eom	Unit to be used as a basis for the interval Works in conjunction with Period to define the billing frequency Possible values are: day week month eom (end of month)

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	`'national_tax'=>$national_tax`	Must reflect the amount of National Tax (GST or HST) appearing on the invoice. Minimum - 0.01 Maximum - 999999.99. Must have 2 decimal places.
Y	Merchant VAT Registration/Single Business Reference Number	20-character alpha-numeric	`'merchant_vat_no'=>$merchant_vat_no`	Merchant's Tax Registration Number must be provided if tax is included on the invoice

Req*	Field Name	Limits	Set Method	Description
				NOTE: Must not be all spaces or all zeroes
C	Local Tax	12-character decimal	`'local_tax'=>$local_tax`	Must reflect the amount of Local Tax (PST or QST) appearing on the invoice If Local Tax included then must not be all spaces or all zeroes; Must be provided if Local Tax (PST or QST) applies Minimum = 0.01 Maximum = 999999.99 Must have 2 decimal places
C	Local Tax (PST or QST) Registration Number	15-character alphanumeric	`'local_tax_no'=>$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

Req*	Field Name	Limits	Set Method	Description
C	Customer VAT Registration Number	13-character alphanumeric	'customer_vat_no'=>$customer_vat_no	If the Customer's Tax Registration Number appears on the invoice to support tax exempt transactions it must be provided here
C	Customer Code/Customer Reference Identifier (CRI)	16-character alphanumeric	'cri'=>$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	'customer_code'=>$customer_code	Optional customer code field that will not be passed along to Visa, but will be included on Moneris reporting
N	Invoice Number	17-character alphanumeric	'invoice_number'=>$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

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 alphanumeric	Buyer/Receipient Name

Req	Variable Name	Field Name	Size/Type	Description
				*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 relationship 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 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	discount_treatment	1-character numeric	Indicates how the 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

Req	Variable Name	Field Name	Size/Type	Description
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 Code / Zip Code	ship_from_pos_code	10-character alphanumeric	The postal code or 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

Req	Variable Name	Field Name	Size/Type	Description
				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 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

Req	Variable Name	Field Name	Size/Type	Description
				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
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

Req	Variable Name	Field Name	Size/Type	Description
				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 every line item
Y	Item Description	item_description	35-character alphanumeric	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

Req	Variable Name	Field Name	Size/Type	Description
				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 **EXAMPLE:** 8% PST should be 8.0 maximum 99.99
Y	Discount Treatment	discount_treatmentL	1-character numeric	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

Req	Variable Name	Field Name	Size/Type	Description
				2 if Tax was calculated on Pre-Discount totals.
C	Discount Amount	discount_amtL	12-character decimal	Amount of discount, 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-	A control number, such as purchase order number, project

Req	Variable Name	Field Name	Size/Type	Description
			numeric	number, department allocation number or name that the purchaser supplied the merchant. Left-justified; may be spaces
N	UniqueInvoiceNumber	Unique Invoice Number	17-character alphanumeric	Unique number associated with the individual transaction provided by the merchant
N	CommodityCode	Commodity Code	15-character alphanumeric	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 alphanumeric	Contains a corporation's value added tax (VAT) number
N	CustomerVatNumber	Customer VAT Number	20-character alphanumeric	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
N	DestinationProvinceCode	Destination State / Province Code	3-character alphanumeric	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 alphanumeric	The country code where goods will be delivered. Left justified with trailing spaces. e.g., CAN - Canada

Req	Variable Name	Field Name	Size/Type	Description
N	ShipFromPosCode	Ship From Postal Code	10-character alphanumeric	The postal code or zip code from which items were shipped
N	ShipToPosCode	Destination Postal Code	10-character alphanumeric	The postal code or zip code where goods will be delivered
N	AuthorizedContactName	Authorized Contact Name	36-character alphanumeric	Name of an individual or company contacted for company authorized purchases
N	AuthorizedContactPhone	Authorized Contact Phone	17-character alphanumeric	Phone number of an individual or company contacted for company authorized purchases
N	AdditionalCardAcceptordata	Additional Card Acceptor Data	40-character alphanumeric	Information pertaining to the card acceptor
N	CardAcceptorType	Card Acceptor Type	8-character alphanumeric	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

Req	Variable Name	Field Name	Size/Type	Description
				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 business 3 - SBA certification as small disadvantaged business 4 - Other government or agency-recognized certification (such as Minority Supplier Development Council)

Req	Variable Name	Field Name	Size/Type	Description
				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 3 - Subcontinent Asian American 4 - Hispanic American 5 - Native American Indian 6 - Native Hawaiian 7 - Native Alaskan 8 - Caucasian 9 - Other

Req	Variable Name	Field Name	Size/Type	Description
				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 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

Req	Variable Name	Field Name	Size/Type	Description
				provided. R - Card acceptor refused to provide business racial/ethnic type
N	CardAcceptorTaxId	Card Acceptor Tax ID	20-character alphanumeric	US Federal tax ID number for value added tax (VAT) ID.
N	CardAcceptorReferenceNumber	Card Acceptor Reference Number	25-character alphanumeric	Code that facilitates card acceptor/corporation communication and record keeping
N	CardAcceptorVatNumber	Card Acceptor VAT Number	20-character alphanumeric	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 alphanumeric	A control number, such as purchase order number, project number, department allocation number or name that the pur-

Req	Variable Name	Field Name	Size/Type	Description
				chaser 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 alphanumeric	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, the productCode value has to be “Discount”
Y	ItemDescription	Item Description	35-character alphanumeric	Line Item description
Y	ItemQuantity	Item Quantity	12-character alpha-	Quantity of line

Req	Variable Name	Field Name	Size/Type	Description
			numeric	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 alphanumeric	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 Minimum amount is 0.00 and maximum is 999999.99
N	CommodityCode	Commodity Code	15-character alphanumeric	Code assigned to the merchant that best categorizes the item(s) being purchased

Req	Variable Name	Field Name	Size/Type	Description
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 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, max-

Req	Variable Name	Field Name	Size/Type	Description
				imum 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

<table>
<tr><th>Req</th><th>Variable Name</th><th>Field Name</th><th>Size/Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>accounting information specific to the client
Mandatory if the merchant's customer provides a Purchase Order Number</td></tr>
<tr><td>N</td><td>big05</td><td>Release Number</td><td>30-character alphanumeric</td><td>A number that identifies a release against a Purchase Order previously placed by the parties involved in the transaction</td></tr>
<tr><td>N</td><td>big10</td><td>Invoice Number</td><td>8-character alphanumeric</td><td>Contains the Amex invoice/reference number</td></tr>
<tr><td>Y</td><td>n101</td><td>Entity Identifier Code</td><td>2-character alphanumeric</td><td>Supported values:
‘R6’ - Requester (required)
‘BG’ - Buying Group (optional)
‘SF’ - Ship From (optional)
‘ST’ - Ship To (optional)
‘40’ - Receiver (optional)</td></tr>
<tr><td>Y</td><td>n102</td><td>Name</td><td>40-character alphanumeric</td><td><table>
<tr><th>n101 code</th><th>n102 meaning</th></tr>
<tr><td>R6</td><td>Requester Name</td></tr>
<tr><td>BG</td><td>Buying Group Name</td></tr>
<tr><td>SF</td><td>Ship From Name</td></tr>
<tr><td>ST</td><td>Ship To Name</td></tr>
<tr><td>40</td><td>Receiver Name</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Req</th><th>Variable Name</th><th>Field Name</th><th>Size/Type</th><th>Description</th></tr>
<tr><td>N</td><td>n301</td><td>Address</td><td>40-character alphanumeric</td><td>Address</td></tr>
<tr><td>N</td><td>n401</td><td>City</td><td>30-character alphanumeric</td><td>City</td></tr>
<tr><td>N</td><td>n402</td><td>State or Province</td><td>2-character alphanumeric</td><td>State or Province</td></tr>
<tr><td>N</td><td>n403</td><td>Postal Code</td><td>15-character alphanumeric</td><td>Postal Code</td></tr>
<tr><td>Y</td><td>ref01</td><td>Reference Identification Qualifier</td><td>2-character alphanumeric</td><td>This element may contain the following qualifiers for the corresponding occurrences of the N1Loop:
<table>
<tr><th>n101 value</th><th>ref01 denotation</th></tr>
<tr><td>R6</td><td>Supported values:
4C - Shipment Destination Code (mandatory)
CR - Customer Reference Number (conditional)</td></tr>
<tr><td>BG</td><td>n/a</td></tr>
<tr><td>SF</td><td>n/a</td></tr>
<tr><td>ST</td><td>n/a</td></tr>
<tr><td>40</td><td>n/a</td></tr>
</table></td></tr>
<tr><td>Y</td><td>ref02</td><td>Reference Identification</td><td>15-character alphanumeric</td><td>VR is the Vendor ID Number, other codes describe the following:</td></tr>
</table>

<table>
<tr><th>Req</th><th>Variable Name</th><th>Field Name</th><th>Size/Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td><table>
<tr><th>ref01 code</th><th>ref02 denotation</th></tr>
<tr><td>4C</td><td>Ship to Zip or Canadian Postal Code (required)</td></tr>
<tr><td>CR</td><td>Cardmember Reference Number (optional)</td></tr>
</table></td></tr>
</table>

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 maximum is 9999999999.
Y	it103	Unit or Basis for Measurement 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

<table>
<tr><th>Req</th><th>Variable Name</th><th>Field Name</th><th>Size/Type</th><th>Description</th></tr>
<tr><td>Y</td><td>it104</td><td>Unit Price</td><td>15-character decimal</td><td>Line item cost per unit
Must contain 2 decimal places
Minimum amount is 0.00 and maximum is 999999.99</td></tr>
<tr><td>N</td><td>it105</td><td>Basis or Unit Price Code</td><td>2-character alphanumeric</td><td>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</td></tr>
<tr><td>N</td><td>it10618</td><td>Product/Service ID Qualifier</td><td>2-character alphanumeric</td><td>Supported values:
'MG' - Manufacturer's Part Number
'VC' - Supplier Catalog Number
'SK' - Supplier Stock Keeping Unit Number
'UP' - Universal Product Code
'VP' – Vendor Part Number
'PO' – Purchase Order Number
'AN' – Client Defined Asset Code</td></tr>
<tr><td>N</td><td>it10719</td><td>Product/Service ID</td><td><table><tr><th>it10618</th><th>it10719 - size/type</th></tr><tr><td>VC</td><td>20-character alphanumeric</td></tr><tr><td>PO</td><td>22-character alphanumeric</td></tr></table></td><td>Product/Service ID corresponds to the preceding qualifier defined in it10618
The maximum length depends on</td></tr>
</table>

Req	Variable Name	Field Name	Size/Type	Description
			it10618 **it10719 - size/type** Other 30-character alphanumeric	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 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	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

Req	Variable Name	Field Name	Size/Type	Description
				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	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

Req	Variable Name	Field Name	Size/Type	Description
				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	pam05	Line Item Extended Amount	8-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 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) ‘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 applic-

Req	Variable Name	Field Name	Size/Type	Description
				able 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) 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

Req	Variable Name	Field Name	Size/Type	Description
				R – Recurring Exempt U – Usage Exempt

A.6 Definition of Request Fields – Offlinx™

Applies to Offlinx™ integration only

Variable and Field 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 109: Receipt object response values

<table>
<tr><th rowspan="2">Value</th><th>Type</th><th>Limits</th><th>Get Method</th></tr>
<tr><th colspan="3">Description</th></tr>
<tr><th colspan="4">General response fields</th></tr>
<tr><td rowspan="2">Card type</td><td>String</td><td>2-character alphabetic (min. 1)</td><td><code>$mpgResponse->getCardType();</code></td></tr>
<tr><td colspan="3">Represents the type of card in the transaction, e.g., Visa, Mastercard.
Possible values:
• V = Visa
• M = Mastercard
• AX = American Express
• DC = Diner's Card
• NO = Novus/Discover
• SE = Sears
• D = Debit
• C1 = JCB</td></tr>
<tr><td rowspan="2">Transaction amount</td><td>String</td><td>(missing or bad snippet)</td><td><code>$mpgResponse->getTransAmount();</code></td></tr>
<tr><td colspan="3">Transaction amount that was processed.</td></tr>
<tr><td rowspan="2">Transaction number</td><td>String</td><td>255-character alphanumeric</td><td><code>$mpgResponse->getTxnNumber();</code></td></tr>
<tr><td colspan="3">Gateway Transaction identifier often needed for follow-on transactions (such as Refund and Purchase Correction) to reference the originally processed transaction.</td></tr>
<tr><td rowspan="2">Receipt ID</td><td>String</td><td>50-character alphanumeric</td><td><code>$mpgResponse->getReceiptId();</code></td></tr>
<tr><td colspan="3">Order ID that was specified in the transaction request.</td></tr>
<tr><td rowspan="2">Transaction type</td><td>String</td><td>2-character alphanumeric</td><td><code>$mpgResponse->getTransType();</code></td></tr>
<tr><td colspan="3">• 0 = Purchase
• 1 = Pre-Authorization
• 2 = Completion
• 4 = Refund
• 11 = Void</td></tr>
</table>

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Reference number	String	18-character numeric	`$mpgResponse->getReferenceNum();`
	Terminal used to process the transaction as well as the shift, batch and sequence number. This data is typically used to reference transactions on the host systems, and must be displayed on any receipt presented to the customer. This information is to be stored by the merchant. Example: 660123450010690030 • 66012345: Terminal ID • 001: Shift number • 069: Batch number • 003: Transaction number within the batch.		
Response code	String	3-character numeric	`$mpgResponse->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	`$mpgResponse->getISO();`
	ISO response code		
Bank totals	Object		
	Response data returned in a Batch Close and Open Totals request. See "Definitions of Response Fields" on the previous page.		
Message	String	100-character alpha-numeric	`$mpgResponse->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	`$mpgResponse->getAuthCode();`
	Authorization code returned from the issuing institution.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Complete	String	true/false	`$mpgResponse->getComplete();`
	Transaction was sent to authorization host and a response was received		
Transaction date	String	Format: yyyy-mm-dd	`$mpgResponse->getTransDate();`
	Processing host date stamp		
Transaction time	String	Format: ##:##:##	`$mpgResponse->getTransTime();`
	Processing host time stamp		
Ticket	String	N/A	`$mpgResponse->getTicket();`
	Reserved field.		
Timed out	String	true/false	`$mpgResponse->getTimedOut();`
	Transaction failed due to a process timing out.		
Is Visa Debit	String	true/false	`$mpgResponse->getIsVisaDebit();`
	Indicates whether the card processed is a Visa Debit.		
Batch Close/Open Totals response fields			
Processed card types	String Array	N/A	
	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	
	Returns the terminal ID/ECR Number from the request.		
Purchase count	String	4-character numeric	`$mpgResponse->getPurchaseCount($ecr_number,$creditCards[$i]);`
	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.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Purchase amount	String	11-character alpha-numeric	`$mpgResponse->getPurchaseAmount ($ecr_number,$creditCards[$i]);`
	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	`$mpgResponse->getRefundAmount($ecr_number,$creditCards[$i]);`
	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 alpha-numeric	`$mpgResponse->getRefundAmount($ecr_number,$creditCards[$i]);`
	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	`$mpgResponse->getCorrectionCount ($ecr_number,$creditCards[$i]);`
	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 alpha-numeric	`$mpgResponse->getCorrectionAmount ($ecr_number,$creditCards[$i]);`
	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)			
Recurring billing success	String	true/false	`$mpgResponse->getRecurSuccess();`
	Indicates whether the recurring billing transaction has been successfully set up for future billing.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Recur update success	String	true/false	$mpgResponse->getRecurUpdateSuccess();
	Indicates recur update success.		
Next recur date	String	yyyy-mm-dd	$mpgResponse->getNextRecurDate();
	Indicates next recur billing date.		
Recur end date	String	yyyy-mm-dd	$mpgResponse->getRecurEndDate();
	Indicates final recur billing date.		
Status Check response fields (see)			
Status code	String	3-character alpha-numeric	$mpgResponse->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	$mpgResponse->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 9.1, page 281)			
AVS result code	String	1-character alpha-numeric	$mpgResponse->getAvsResultCode();
	Indicates the address verification result. For a full list of possible response codes refer to Section Appendix B.		
CVD response fields (see)			
CVD result code	String	2-character alpha-numeric	$mpgResponse->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)			

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Type	String	99-character alphanumeric	
	VERes, PARes or error defines what type of response you are receiving .		
Success	Boolean	true/false	`$mpgResponse->getMpiSuccess();`
	True if attempt was successful, false if attempt was unsuccessful.		
Message	String	100-character alphabetic	`$mpgResponse->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 alphanumeric	
	URL to which the PARes is returned		
MD	String	1024-character alphanumeric	
	Merchant-defined data that was echoed back		
ACS URL	String	255-character alphanumeric	
	URL that will be for the generated pop-up		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
MPI CAVV	String	28-character alpha-numeric	
	VbV/MCSC/American Express SafeKey authentication data		
MPI E-Commerce Indicator	String	1-character alpha-numeric	
CAVV result code	String	1-character alpha-numeric	`$mpgResponse->getCavvResultCode();`
	Indicates the Visa CAVV result. For more information, see 8.6.7 Cavv Result Codes for Verified by Visa. • 0 = CAVV authentication results invalid • 1 = CAVV failed validation; authentication • 2 = CAVV passed validation; authentication • 3 = CAVV passed validation; attempt • 4 = CAVV failed validation; attempt • 7 = CAVV failed validation; attempt (US issued cards only) • 8 = CAVV passed validation; attempt (US issued cards only) • The CAVV result code indicates the result of the CAVV validation.		
MPI inline form			`$mpgResponse->getMpiInLineForm();`
Vault response fields (see 4.1, page 50)			
Data key	String	28-character alpha-numeric	`$mpgResponse->getDataKey();`
	The data key response field is populated when you send a Vault Add Credit Card – ResAddCC (page 52), Vault Encrypted Add Credit Card – EncResAddCC (page 56), Vault Tokenize Credit Card – ResTokenizeCC (page 79), Vault Temporary Token Add – ResTempAdd (page 59) or Vault Add Token – ResAddToken (page 76) transaction. It is the profile identifier that all future financial Vault transactions will use to associate with the saved information.		
Vault payment type	String	cc	`$mpgResponse->getPaymentType();`
	Indicates the payment type associated with a Vault profile		
Expiring card's Payment type	String	cc	`$mpgResponse->getExpPaymentType();`
	Indicates the payment type associated with a Vault profile. Applicable to Vault Get Expiring transaction type.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Vault masked PAN	String	20-character numeric	`$mpgResponse->getResDataMaskedPan();`
	Returns the first 4 and/or last 4 of the card number saved in the profile.		
Expiring card's Masked PAN	String	20-character numeric	`$mpgResponse->getResDataMaskedPan();`
	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	`$mpgResponse->getResSuccess();`
	Indicates whether Vault transaction was successful.		
Vault customer ID	String	30-character alphanumeric	`$mpgResponse->getResDataCustId();`
	Returns the customer ID saved in the profile.		
Expiring card's customer ID	String	30-character alphanumeric	`$mpgResponse->getResDataCustId();`
	Returns the customer ID saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault phone number	String	30-character alphanumeric	`$mpgResponse->getResDataPhone();`
	Returns the phone number saved in the profile.		
Expiring card's phone number	String	30-character alphanumeric	`$mpgResponse->getResDataPhone();`
	Returns the phone number saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault email address	String	30-character alphanumeric	`$mpgResponse->getResDataEmail();`
	Returns the email address saved in the profile.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Expiring card's email address	String	30-character alpha-numeric	`$mpgResponse->getResDataEmail();`
	Returns the email address saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault note	String	30-character alpha-numeric	`$mpgResponse->getResDataNote();`
	Returns the note saved in the profile.		
Expiring card's note	String	30-character alpha-numeric	`$mpgResponse->getResDataNote();`
	Returns the note saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault expiry date	String	4-character numeric	`$mpgResponse->getResDataExpDate();`
	Returns the expiry date of the card number saved in the profile. YYMM format.		
Expiring card's expiry date	String	4-character numeric	`$mpgResponse->getResDataExpDate();`
	Returns the expiry date of the card number saved in the profile. YYMM format. Applicable to Vault Get Expiring transaction type.		
Vault E-commerce indicator	String	1-character numeric	`$mpgResponse->getResDataCryptType();`
	Returns the e-commerce indicator saved in the profile.		
Expiring card's E-commerce indicator	String	1-character numeric	`$mpgResponse->getResDataCryptType();`
	Returns the e-commerce indicator saved in the profile. Applicable to Vault Get Expiring transaction type.		
Vault AVS street number	String	19-character alpha-numeric	`$mpgResponse->getResDataAvsStreetNumber();`
	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.		

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Expiring card's AVS street number	String	19-character alphanumeric	`$mpgResponse->getResDataAvsStreetNumber();`
	Returns the AVS street number saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault AVS street name	String	19-character alphanumeric	`$mpgResponse->getResDataAvsStreetName();`
	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 alphanumeric	`$mpgResponse->getResDataAvsStreetName();`
	Returns the AVS street name saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault AVS ZIP code	String	9-character alphanumeric	`$mpgResponse->getResDataAvsZipcode();`
	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 alphanumeric	`$mpgResponse->getResDataAvsZipcode();`
	Returns the AVS zip/postal code saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer. Applicable to Vault Get Expiring transaction type.		
Vault credit card number	String	20-character numeric	`$mpgResponse->getResDataPan();`
	Returns the full credit card number saved in the Vault profile. Applicable to Vault Lookup Full transaction only.		
Corporate card	String	true/false	`$mpgResponse->getCorporateCard();`
	Indicates whether the card associated with the Vault profile is a corporate card.		
Encrypted Mag Swipe response fields (see 6, page 116)			

Table 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Masked credit card number	String	20-character alpha-numeric	`$mpgResponse->getMaskedPan();`
Convenience Fee response fields (see Appendix A, page 1)			
Convenience fee success	String	true/false	`$mpgResponse->getCfSuccess();`
	Indicates whether the Convenience Fee transaction processed successfully.		
Convenience fee status	String	2-character alpha-numeric	`$mpgResponse->getCfStatus();`
	Indicates the status of the merchant and convenience fee transactions. The CfStatus field provides details about the transaction behavior and should be referenced when contacting Moneris Customer Support. Possible values are: • 1 or 1F – Completed 1st purchase transaction • 2 or 2F – Completed 2nd purchase transaction • 3 – Completed void transaction • 4A or 4D – Completed refund transaction • 7 or 7F – Completed merchant independent refund transaction • 8 or 8F – Completed merchant refund transaction • 9 or 9F – Completed 1st void transaction • 10 or 10F – Completed 2nd void transaction • 11A or 11D – Completed refund transaction		
Convenience fee amount	String	9-character decimal	`$mpgResponse->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	`$mpgResponse->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 109: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Convenience fee type	String	AMT/PCT	`$mpgResponse->getFeeType();`
	The type of convenience fee that has been defined on the merchant's profile. Available options are: AMT – fixed amount PCT – percentage		

Table 110: 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 111: 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

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.

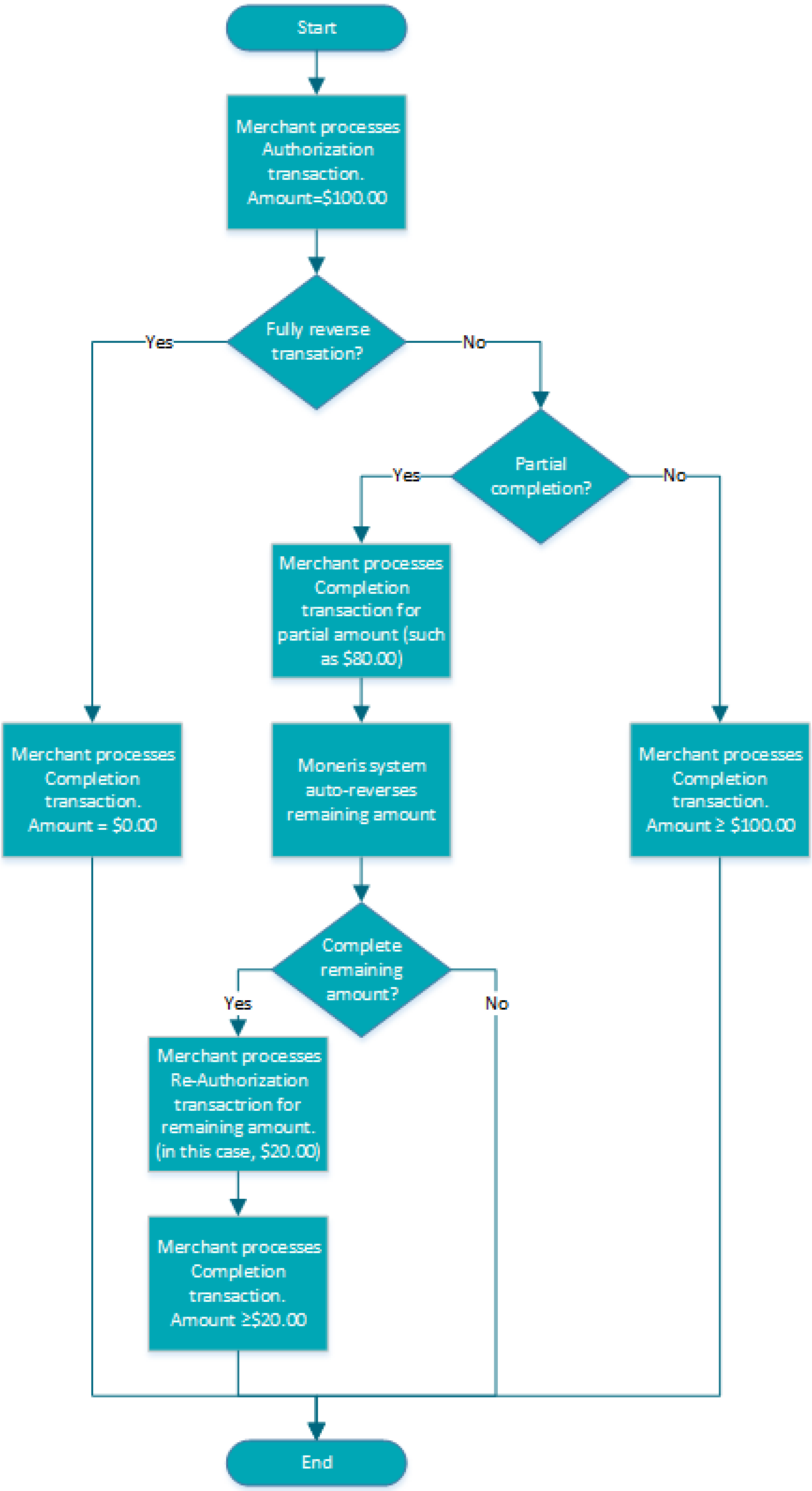
Malformed XML
Parse error.

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 category (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 112: Checklist for web display requirements

Done	Requirement
Checkout page	

Table 112: 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 112: 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 113: 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 information	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 information	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 114: 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 114: Checklist for front-end tests

Case #	Date Completed	Remarks
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

Merchant Requirements

Table 115: 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 115: 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*$^{\text{MD}}$>> (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" - $^{\text{MD}}$ 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 116: 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 117: 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 category (Circle one)	Government Education General	
Third-party service provider	Company name	
Service provider's merchant web application	Solution name	
	Version	

Merchant Requirements

Table 118: 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 118: 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 119: 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
- H.2 Test Cases
- H.3 Merchant front-end test case values

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 120: 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 120: 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 outcome	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 121: 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 121: 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 122: 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 122: 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 123: 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 124: 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 124: 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 125: 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 126: Test case 2—Funded URL

Redirection URL	Funded
ISSLANG	en

Table 126: 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 127: 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 128: 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 128: 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 129: 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 130: 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 130: 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

Copyright Notice

This document has been produced as a reference guide to assist Moneris client's hereafter referred to as merchants. Every effort has been made to the make the information in this reference guide as accurate as possible. The authors of Moneris Solutions shall have neither liability nor responsibility to any person or entity with respect to any loss or damage in connection with or arising from the information contained in this reference guide.

Trademarks

Printed in Canada.

10 9 8 7 6 5 4 3 2 1