Merchant Integration Guide

PHP API
with Visa, MasterCard and American Express
Level 2/3
v1.1.6

Moneris SOLUTIONS

Revision Number	Date	Description
V1.1.0	February 23, 2007	-Document edited for coherence
V1.1.1	October 15, 2008	-Section 2. System and Skill Requirements -Added PCI and PA DSS compliance note
V1.1.2	July 8, 2011	-New download link updated in various locations http://www.eselectplus.ca/en/downloadable-content -Section 2. System and Skill Requirements -Edited PCI and PA DSS compliance note
V1.1.3	July 18, 2011	-Section 11. How do I test my Solution? -Added CA Root Certificate File.
V1.1.4	October 14, 2011	-Section 6. Transaction Types and Transaction Flow For MasterCard -Removed Note.
V1.1.5	December 20, 2011	-Added Section 10. Transaction Types and Transaction Flow For American Express -Added Section 11. Transaction Examples for American Express -Added Appendix D. Definition of American Express Level 2/3 Request Fields
V1.1.6	March 23, 2012	-Removed Appendix F. Development Response Codes -Corrected Visa/MasterCard Level 2/3 Transaction Type names.

Table of Contents

1. About this Documentation

This documentation contains the basic 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 is to be used by merchants that require the ability to pass Level 2 / 3 data for Visa and MasterCard.

2. System and Skill Requirements

In order to use the PHP API your system will need to have the following:
1. A WebServer with an SSL certificate
2. PHP 4 or later
3. Port 43924 open for bi directional communication
4. CURL- PHP- this can be downloaded from http://curl.haxx.se/download.html

As well, you will need to have the following knowledge and/or skill set:
1. PHP

Note:

It is important to note that 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". Failure to comply with PCI DSS and the Card Association Compliance Programs may result in a Merchant being subject to fines, fees or assessments and/or termination of processing services. Non-compliant solutions may prevent merchants boarding with Moneris Solutions.

As a Moneris Solutions 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.

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

For more information on how to get your application PCI-DSS compliant, please contact our Integration Specialists and visit http://www.eselectplus.ca/en/downloadable-content to download the PCI-DSS Implementation Guide.

3. What is the Process I will need to follow?

You will need to follow these steps.
1. Do the required development as outlined in this document
2. Test your solution in the test environment
3. Activate your store
4. Make the necessary changes to move your solution from the development environment into production as outlined in this document

4. Transaction Types and Flow for Non-Level 2 / 3 supported card types

eSelect plus supports a wide variety of Level1 transactions through the API. Below is a list of transaction supported by the API; other terms used for the transaction type are indicated in brackets.

Purchase – (sale) The purchase transaction verifies funds on the customer's card, removes the funds and readies them for deposit into the merchant account.

Preauth – (authorisation / preauthorisation) 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 preauth so that they may be settled in the merchant account a capture must be performed.

Capture – (Completion/Preauth Completion) Once a Preauth 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.

Void – (Correction, Purchase Correction) Purchase and Captures 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.

Refund – (Credit) A refund can be performed against a purchase or a capture to refund any part, or all of the transaction.

Batch Close – (End of Day/Settlement) When a batch close is performed it takes the monies from all purchase, capture, void and refund transactions so they will be deposited the following business day. For funds to be deposited the following business day the batch must close before 11pm EST.

* A void 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.

5. Transaction Examples for Non-Level 2 / 3 supported card types

Included below is the sample code that can be found in the “Examples” folder of the API download.

Purchase (basic)

In the purchase example we require several variables. store_id, api_token, order_id, amount, pan(credit card number), expiry date, and crypt type. Please see appendices for variable definitions.

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestPurchase.php store7 45728773 45109

require "../mpgClasses.php";

$storeid='store1';
$apitoken='yesguy';
$orderid='aknflkasjdnasdf';

## step 1) create transaction hash ###
$txnArray=array(type=>'purchase',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());

?>
```

Preauth (basic)

The preauth is virtually identical to the purchase with the exception of the transaction type. It is 'preauth' instead of 'purchase'. Like the purchase example preauths require several variables. store_id, api_token, order_id, amount, pan (credit card number), expiry date, and crypt type. Please see appendices for variable definitions.

```
<?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 45728773 45109
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];

## step 1) create transaction hash ###
$txnArray=array(type=>'preauth',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

Capture

The capture transaction is used to secure the funds locked by a preauth transaction. When sending a capture request you will need two pieces of information from the original preauth – the order_id and the txn_number from the returned response.

```
<?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 TestCompletion.php store1 45728773 45109 76452

require "../mpgClasses.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

$compamount='1.01';

## step 1) create transaction array ###
$txnArray=array(type=>'completion',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         comp_amount=>$compamount,
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());

?>
```

Void

The 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 transactions that can be voided are captures and purchases. To send a void the order_id and txn_number from the capture or purchase are required.

```
<?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 TestPurchaseCorrection.php store1 45728773 45109 76452
##

require "../mpgClasses.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

## step 1) create transaction hash ###
$txnArray=array(type=>'purchasecorrection',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         crypt_type=>'7'
        );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());


?>
```

Refund

The 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 or purchase. To send a refund you will require the order_id and txn_number from the original capture or purchase.

```
<?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 45728773 45109 76452
##

require "../mpgClasses.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];


## step 1) create transaction array ###
$txnArray=array(type=>'refund',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         amount=>'1.01',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());

?>
```

Independent Refund

The 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 eSelect plus gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a purchase or a preauth.

```
<?php

##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestIndependentRefund.php store1 45728773 45109
##

require "../mpgClasses.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];

## step 1) create transaction array ###
$txnArray=array(type=>'ind_refund',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());

?>
```

Batch Close

At the end of every day (11pm EST) the batch needs to be closed in order to have the funds settled the next business day. ***By default eSelectplus will close your batch automatically for you daily whenever there are funds in the open batch.*** Some merchants prefer to control batch close, and disable the automatic functionality. For these merchants we have provided the ability to close your batch through the API. When a batch is closed the response will include the transaction count and amount for each type of transaction for each type of card. To disable automatic close you will need to call the technical support line.

```
<?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 45728773 66004444

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$ecr_number=$argv[3];

## 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);

## step 3) create mpgHttpsPost object which does an https post ##
$mpgHttpPost=new mpgHttpsPost($storeid,$apitoken,$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]);
  }

?>
```

6. Transaction Types and Transaction Flow for MasterCard

When support for Level2/3 transactions is enabled for MasterCard all Level 1 and Level 2/3 MasterCard transactions must be sent using the following transaction set. This set includes a suite of financial transactions as well as a transaction that allows for the passing of Level 2/3 data. Batch Close, Open Totals and Preauth are identical to the non-MasterCard transactions outlined in the Sections 4 & 5. When the response contains CorporateCard equal to true then you can submit a Level2/3 data transaction. If Corporate Card is false then the card does not support Level 2/3 data.

*** Note: If you do not wish to send any Level 2/3 data then you may submit MasterCard transactions using the transaction set outlined in Section 4 & 5 (Non-level 2/3 transactions).*

Preauth – (authorisation / preauthorisation) 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 preauth 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 preauth as a preauth is not settled.

MCCompletion – (Capture/Preauth Completion) Once a Preauth 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 must be performed. MCCompletion will return a result indicating if it is a corporate credit card.

MCPurchaseCorrection – (Void, Correction) Purchase and Captures 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. MCPurchaseCorrection will return a result indicating if it is a corporate credit card.

MCRefund – (Credit) A refund can be performed against a purchase or a capture to refund any part, or all of the transaction. MCRefund will return a result indicating if it is a corporate credit card.

MCIndependentRefund – (Credit) A 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 eSelect plus. MCIndependentRefund will return a result indicating if it is a corporate credit card.

MCLevel23 – (Level 2/3 Data) The MCLevel23 will contain all the required and optional data fields for Level 2/3 data. MCLevel23 data can be sent when the card has been identified in the transaction request as being a corporate card

* An MCPurchaseCorrection 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.

7. Transaction Examples for MasterCard

Included below is the sample code that can be found in the “Examples” folder of the API download.

Preauth (basic)

Preauth require several variables. store_id, api_token, order_id, amount, pan (credit card number), expiry date, and crypt type. Please see appendices for variable definitions.

```
<?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 45728773 45109

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];

## step 1) create transaction hash ###
$txnArray=array(type=>'preauth',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

MCCompletion

The MasterCard Completion transaction is used to secure the funds locked by a preauth transaction. When sending a capture request you will need two pieces of information from the original preauth – the order_id and the TxnNumber from the returned response.

```
<?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 TestMCCompletion.php store1 45728773 45109 76452
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

$compamount='1.01';

## step 1) create transaction array ###
$txnArray=array(type=>'mccompletion',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         comp_amount=>$compamount,
         crypt_type=>'7'
           );


## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

MCPurchaseCorrection

The MasterCard (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 TxnNumber from the capture or purchase are required.

```
<?php

##
## This program takes 4 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. txn number
##
## Example php -q TestMCPurchaseCorrection.php storeid apitoken orderid txnnumber
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

## step 1) create transaction array ###
$txnArray=array(type=>'mcpurchasecorrection',
         order_id=>$orderid,
         txn_number=>$txnnumber,
         crypt_type=>'7'
           );


## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

MCRefund

The MasterCard 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 or purchase. To send a refund you will require the order_id and txn_number from the original capture or purchase.

```
<?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 TestMCRefund.php store1 45728773 45109 76452
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];


## step 1) create transaction array ###
$txnArray=array(type=>'mcrefund',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         amount=>'1.01',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

MCIndependentRefund

The MasterCard 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 eSelect plus gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a preauth.

```
<?php

## This program takes 6 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. amount
## 5. pan
## 6. expdate
## Example php -q TestMCIndependentRefund.php level23 moymoy 200404130332 24.01 4242424242424242 0505

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];

## step 1) create transaction array ###
$txnArray=array(type=>'mcind_refund',
         order_id=>$orderid,
         amount=>$amount,
         pan=>$pan,
         expdate=>$expdate,
         crypt_type=>'7'
           );
## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

MCLevel23

The MCLevel23 transaction request passes the Level 2 and 3 data for processing. The MCLevel23 request must be preceded by a financial transaction (completion, refund . . .) and the Corporate Card flag must be set to "true" in the response. The MCLevel23 request will need to contain the order_id of the financial transaction as well and the txn_number. Please see the appendices for a description of the Level 2 and Level 3 fields.

```
<?php

## This program takes 5 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. trans number
## 5. MC level2/3 transaction data file name
## Example php -q TestMCLevel23.php moneris hurgle 3333 eeeee mclevel23.txt

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$mclevel23filename=$argv[5];
$mclevel23String=implode('',file($mclevel23filename));

## step 1) create transaction array ###
$txnArray=array(type=>'mclevel23',
         txn_number=>$txnnumber,
         order_id=>$orderid,
           );

## step 2) create a transaction  object passing the array created in
## step 1.

$mpgTxn = new mpgTransaction($txnArray);
print ($mclevel23String);
## step 2.1) set MC Level2/3 transaction string check the appendix for the format
$mpgTxn->setMcLevel23String($mclevel23String);

## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());
?>
```

8. Transaction Types and Transaction Flow for Visa

When support for Level2/3 transactions is enabled for Visa all Level 1 and Level 2/3 Visa transactions should be sent using the following transaction set. This set includes a suite of financial transactions as well as two transactions that allow for the passing of Level 2/3 data. Batch Close, Open Totals and Preauth are identical to the non-Visa transactions outlined in the section above. When the response contains CorporateCard equal to true then you can submit a Level2/3 data transaction. If Corporate Card is false then the card does not support Level 2/3 data. The response will also include a field called MessageId – this field must be sent in the Level 2/3 data transaction.

*** Note: If you do not wish to send any Level 2/3 data then you may submit Visa transactions using the transaction set outlined in Section 4 & 5 (Non-Visa and non-MC transactions). Purchase Corrections would be sent using the methods outlined in Section 4 & 5.*

Preauth – (authorisation / preauthorisation) 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 preauth so that they may be settled in the merchant account a capture must be performed.

VSCompletion – (Capture/Preauth Completion) Once a Preauth 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 VSCompletion a Preauth must be performed. VSCompletion will return a result indicating if it is a corporate credit card.

VSRefund – (Credit) A refund can be performed against a purchase or a capture to refund any part, or all of the transaction. VSRefund will return a result indicating if it is a corporate credit card.

VSIndependentRefund – (Credit) A 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 eSelect plus. VSIndependentRefund will return a result indicating if it is a corporate credit card.

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

* A VSPurchaseCorrection 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.

9. Transaction Examples for Visa

Included below is the sample code that can be found in the “Examples” folder of the API download.

Preauth (basic)

Preauths require several variables. store_id, api_token, order_id, amount, pan(credit card number), expiry date, and crypt type. Please see appendices for variable definitions.

```
<?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 45728773 45109

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];

## step 1) create transaction hash ###
$txnArray=array(type=>'preauth',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

VSCompletion

The Visa Completion transaction is used to secure the funds locked by a preauth transaction. When sending a capture request you will need two pieces of information from the original preauth – the order_id and the TxnNumber from the returned response.

```
<?php

## This program takes 4/7 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. trans number
## 5. order level gst
## 6. merchant gst number
## 7. customer relationship identification
##
## Example php -q TestVSCompletion.php level23 moymoy 200404160405 189533-7-0
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$paras=count($argv);
$ol_gst="";
$ol_gstno="";
$cri="";
if ( $paras > 5 )
{
  $ol_gst=$argv[5];
}
if ( $paras > 6 )
{
  $ol_gstno=$argv[6];
}
if ( $paras > 7 )
{
  $cri=$argv[7];
}

$compamount='1.01';

## step 1) create transaction array ###
$txnArray=array(type=>'vscompletion',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         comp_amount=>$compamount,
         crypt_type=>'7',
         order_level_gst=>$ol_gst,
         merchant_gst_no=>$ol_gstno,
         cri=>$cri
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

VSRefund

The Visa 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 or purchase. To send a refund you will require the order_id and txn_number from the original capture or purchase.

```
<?php

##
## This program takes 4/7 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. trans number
## 5. order level gst
## 6. merchant gst number
## 7. customer relationship identification
##
## Example php -q TestMCRefund.php store1 45728773 45109 76452
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$paras=count($argv);
$ol_gst="";
$ol_gstno="";
$cri="";

if ( $paras > 5 )
{
  $ol_gst=$argv[5];
}

if ( $paras > 6 )
{
  $ol_gstno=$argv[6];
}


if ( $paras > 7 )
{
  $cri=$argv[7];
}

## step 1) create transaction array ###
$txnArray=array(type=>'vsrefund',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         amount=>'1.01',
         crypt_type=>'7',
         order_level_gst=>$ol_gst,
         merchant_gst_no=>$ol_gstno,
         cri=>$cri
           );

## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

VSIndependentRefund

The Visa 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 eSelect plus gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a preauth.

```
<?php

##
## This program takes 6/9 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. amount
## 5. pan
## 6. expdate
## 7. order level gst
## 8. merchant gst number
## 9. customer relationship identification
##
## Example php -q TestVSIndependentRefund.php level23 moymoy 200404160444 3.01 5550004242424242 0505
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];
$ol_gst="";
$ol_gstno="";
$cri="";
if ( $paras > 7 )
{
  $ol_gst=$argv[7];
}
if ( $paras > 8 )
{
  $ol_gstno=$argv[8];
}
if ( $paras > 9 )
{
  $cri=$argv[9];
}

## step 1) create transaction array ###
$txnArray=array(type=>'vsind_refund',
         order_id=>$orderid,
         amount=>$amount,
         pan=>$pan,
         expdate=>$expdate,
         crypt_type=>'7',
         order_level_gst=>$ol_gst,
         merchant_gst_no=>$ol_gstno,
         cri=>$cri
           );


## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

VSLevel23

Upon sending a VSCompletion, VSRefund, VSPurchaseCorrection and successfully receiving a message_id in the response the Level 2/3 data can be submitted. Below is a sample of setting the fields. For a full description of all fields (required and optional) please see the appendices.

```
<?php

## This program takes 5 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. trans number
## 5. gxml file name
##
## Example php -q TestVSLevel23.php level23 moymoy TestL230419_0012 189586-25-0 gxml.txt
require "../l23Classes.php";
$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$gxmlfilename=$argv[5];
$gxmlString=implode('',file($gxmlfilename));

## step 1) create transaction array ###
$txnArray=array(type=>'vslevel23',
         txn_number=>$txnnumber,
         order_id=>$orderid,
           );

## step 2) create a transaction  object passing the array created in
## step 1.

$mpgTxn = new mpgTransaction($txnArray);

## step 2.1) set gxml string check the appendix
$mpgTxn->setGxmlString($gxmlString);

## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());
?>
```

10. Transaction Types and Transaction Flow for American Express

When support for Level2/3 transactions is enabled for American Express; all Level 2/3 American Express transactions (Corporate Card = true) should be sent using the following transaction set. All non Level2/3 transactions should be sent using the transaction set outlined in Section 4 & 5. The American Express Level 2/3 transaction set includes a suite of financial transactions as well as specific transactions that allow for the passing of Level 2/3 data. Batch Close, Open Totals, Purchase Correction and Preauth are identical to the non-Level 2/3 American Express transactions outlined in the section above. When the response contains CorporateCard equal to true then you can submit a Level2/3 data transaction. If CorporateCard is false then the card does not support Level 2/3 data. The response will also include a field called MessageId – this field must be sent in the Level 2/3 data transaction.

*** Note: If you do not wish to send any Level 2/3 data then you must submit American Express transactions using the transaction set outlined in Section 4 & 5 (Non-Level 2/3 transactions).*

Preauth – (authorisation / preauthorisation) 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 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.

AXCompletion – (Capture/Preauth Completion) Once a Preauth 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.

AXPurchaseCorrection – (Void, Correction) Purchases and Captures 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.

AXRefund – (Credit) A refund can be performed against a purchase or a capture to refund any part, or all of the transaction.

AXIndependentRefund – (Credit) A 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 eSELECTplus.

* An AXPurchaseCorrection 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.

11. Transaction Examples for American Express

Included below is the sample code that can be found in the “Examples” folder of the API download.

Preauth (basic)

Preauths require several variables. store_id, api_token, order_id, amount, pan(credit card number), expiry date, and crypt type. Please see appendices for variable definitions.

```
<?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 45728773 45109
##
require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];

## step 1) create transaction hash ###
$txnArray=array(type=>'preauth',
         order_id=>$orderid,
         amount=>'1.01',
         pan=>'4242424242424242',
         expdate=>'0303',
         crypt_type=>'7'
           );
## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());


?>
```

AXCompletion

The American Express Completion transaction is used to secure the funds locked by a preauth transaction. When sending a capture request you will need two pieces of information from the original preauth – the order_id and the TxnNumber from the returned response.

```
<?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 TestMCCompletion.php store1 45728773 45109 76452
##
require "./mpgClasses.php";

$storeid="moneris";
$apitoken="yesguy";
$orderid="mvt4416883223";
$txnnumber="7181-647-0";
$compamount="1.00";
$axlevel23String=implode('',file($argv[0]));

## step 1) create transaction array ###
$txnArray=array(type=>'axcompletion',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         comp_amount=>$compamount,
           );
## step 2) create a transaction  object passing the hash created in
## step 1.

$mpgTxn = new mpgTransaction($txnArray);

## step 2.1) set AX Level2/3 transaction string
$mpgTxn->setAxLevel23String($axlevel23String);

## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

AXPurchaseCorrection

The American Express 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 TxnNumber from the capture or purchase are required.

```
<?php

##
## This program takes 4 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## 4. txn number
##
## Example php -q TestMCPurchaseCorrection.php storeid apitoken orderid txnnumber
##

require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

## step 1) create transaction array ###
$txnArray=array(type=>'axpurchasecorrection',
         order_id=>$orderid,
         txn_number=>$txnnumber,
         crypt_type=>'7'
           );


## 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);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

AXRefund

The American Express 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 or purchase. To send a refund you will require the order_id and txn_number from the original capture or purchase.

```
<?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 TestMCCompletion.php store1 45728773 45109 76452
##
require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$amount=$argv[5];
$axlevel23String=implode('',file($argv[6]));

## step 1) create transaction array ###
$txnArray=array(type=>'axrefund',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         amount=>$amount,
           );
## step 2) create a transaction  object passing the hash created in
## step 1.

$mpgTxn = new mpgTransaction($txnArray);

## step 2.1) set AX Level2/3 transaction string
$mpgTxn->setAxLevel23String($axlevel23String);

## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

AXIndependantRefund

The American Express 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 eSelect plus gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a preauth.

```
<?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 TestMCCompletion.php store1 45728773 45109 76452
##
require "../l23Classes.php";

$storeid=$argv[1];
$apitoken=$argv[2];
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];
$axlevel23String=implode('',file($argv[7]));

## step 1) create transaction array ###
$txnArray=array(type=>'axind_refund',
         txn_number=>$txnnumber,
         order_id=>$orderid,
         amount=>$amount,
         pan=>$pan,
         expdate=>$expdate
           );
## step 2) create a transaction  object passing the hash created in
## step 1.

$mpgTxn = new mpgTransaction($txnArray);

## step 2.1) set AX Level2/3 transaction string
$mpgTxn->setAxLevel23String($axlevel23String);

## step 3) create a mpgRequest object passing the transaction object created
## in step 2
$mpgRequest = new mpgRequest($mpgTxn);

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($storeid,$apitoken,$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());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nMessageId = " . $mpgResponse->getMessageId());

?>
```

12. What Information will I get as a Response to My Transaction Request?

For each transaction you will receive a response message. For a full description of each field please refer to Appendix E. Definitions of Response Fields.

To determine whether a transaction is successful or not the field that must be checked is Response Code. See the table below to determine the transaction result.

Response Code	Result
0 – 49 (inclusive)	Approved
50 – 999 (inclusive)	Declined
Null	Incomplete

For a full list of response codes and the associated message please refer to the Response Code document available for download at http://www.eselectplus.ca/en/downloadable-content.

13. How Do I Test My Solution?

A testing environment is available for you to connect to while you are integrating your site to our payment gateway. The test environment is generally available 7x24; however since it is a development environment we cannot guarantee 100% availability. Also, please be aware that other merchants are using the testing environment so you may see transactions and user ids that you did not create. As a courtesy to others that are testing we ask that when you are processing refunds, changing passwords and/or trying other functions that you use only the transactions/users that you created.

When using the APIs in the test environment you will need to use test store, api_token and user ids. These are different than your production ids. The ids that you can use in the test environment are in the table below.

Test IDs			
store_id	api_token	Username	Password
level23	moymoy	DemoUser	password
moneris	hurgle	DemoUser	password

The development environment will approve and decline transactions based on the penny value of the amount. ,00 , .01 and .04 will approve, other values will decline. (i.e. 37.01 will approve and 37.10 will decline) . Please see **Error! Reference source not found.** for details on the values you must send to simulate the various responses. Transactions in the test environment should not exceed $1000.00. This limit does not exist in the production environment. For a list of all current test environment responses for various penny values, please see the Test Environment Penny Response table available at http://www.eselectplus.ca/en/downloadable-content.

When testing level 2/3 transactions, you can use any valid card number with any expiry date. The following test card numbers can be used as well with any expiry date.

Test Card Numbers	
Card Plan	Card Number
MasterCard	5454545454545454
Visa	4242424242424242
Amex	373599005095005 (Amex will approve on .37 and .70)
Diners	36462462742008

To access the Merchant Resource Center in the test environment go to https://esqa.moneris.com. And use the logins provided in the previous table.

The test environment has been designed to replicate our production environment as closely as possible. One major difference is that we are unable to send test transactions onto the authorization network and thus Issuer responses are simulated. Additionally, the requirement to emulate approval, decline and error situations dictates that we use certain transaction variables to initiate various response and error situations.

cURL CA Root Certificate File:

The default installation of PHP/cURL does not include the cURL CA root certificate file. In order for the eSelectPlus PHP API to connect to the eSelectPlus 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. Follow the instructions below to set this up.

1) If cURL was not installed separately from your PHP installation, libcurl is included in your PHP installation. You will need to download the 'cacert.pem' file from 'http://curl.haxx.se/docs/caextract.html' and save it to the necessary directory. 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').

2) 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 manual.

14. What Do I Need to Include in the Receipt?

Visa and MasterCard expect certain variables be returned to the cardholder and presented as a receipt when a transaction is approved. These 12 fields are listed below. A sample receipt is provided in Appendix F. Sample Receipt .

1. Amount
2. Transaction Type
3. Date and Time
4. AuthCode
5. ResponseCode
6. ISO Code
7. Response Message
8. Reference Number
9. Goods and Services Order
10. Merchant Name
11. Merchant URL
12. Cardholder Name

15. How Do I Activate My Store?

Once you have received your activation letter/fax go to https://www3.moneris.com/connect/en/activate/index.php as instructed in the letter/fax. You will need to input your store id and merchant id. Once this is confirmed you will be provided with your API Token. You will also need to create an administrator account that you will use to log into the interface to access and administer your eSelect plus store. You will need to use the store ID and API Token to send transactions through the API.

Once you have created your first Merchant Interface user please logon to the Interface by clicking the “eSelect plus” button. Once you have logged in please proceed to “Admin” and then “Store Settings”. At the bottom please place a check beside the APIs that you are using. This will allow us to keep you up to date regarding any changes to the APIs that may affect your store.

16. How Do I Configure My Store For Production?

Once you have completed your testing you are ready to point your store to the production host. You will need to change the "host" to be www3.moneris.com. You will also need to change the store_id to reflect your production store ID and well the api_token must be changed to your production token to reflect the token that you received during activation.

Once you are in production you will access the Merchant Resource Centre at https://www3.moneris.com/mpg You can use the store administrator id you created during the activation process and then create additional users as needed.

For further information on how to use the Merchant Resource Centre please see the eSELECTplus Merchant Resource Centre User's Guide which is available at http://www.eselectplus.ca/en/downloadable-content.

17. How Do I Get Help?

If you require technical assistance while integrating your store, please contact the eSelectplus Support Team:

For technical support:
Phone: 1-866-319-7450 (Technical Difficulties)

For integration support:
Phone: 1-866-562-4354
Email: eselectplus@moneris.com

When sending an email support request please be sure to include your name and phone number, a clear description of the problem as well as the type of API that you are using. **For security reasons, please do not send us your API Token combined with your store ID, or your merchant number and device number in the same email.**

18. Appendix A. Definition of Request Fields

Request Fields		
Variable Name	Size/Type	Description
order_id	99 / an	Merchant defined unique transaction identifier - must be unique for every Purchase, PreAuth and Independent Refund attempt. For Refunds, Completions and Voids the order_id is must reference the original transaction.
pan	20 / variable	Credit Card Number - no spaces or dashes. Most credit card numbers today are 16 digits in length but some 13 digits are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and/or potential support of private label card ranges.
expdate	4 / num	Expiry Date - format YYMM no spaces or slashes. PLEASE NOTE THAT THIS IS REVERSED FROM THE DATE DISPLAYED ON THE PHYSICAL CARD WHICH IS MMYY
amount	9 / decimal	Amount of the transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
crypt_type	1 / an	E-Commerce Indicator: 1 - Mail Order/Telephone Order - Single 2 - Mail Order/Telephone Order - Recurring 3 - Mail Order Telephone Order - Installment 4 - Mail Order Telephone Order - Unknown Classification 5 - Authenticated Ecommerce Transaction (VBV or SPA) 6 – Non Authenticated Ecommerce Transaction (VBV or SPA) 7 - SSL enabled merchant 8 - Non Secure Transaction (Web or Email Based) 9 - SET nonAuthenticated transaction
txn_number	255 / varchar	Used when performing follow on transactions - this must be filled with the value that was return as the trans_id in the response of the original transaction. When performaing a capture this must reference the Preauth. When performing a refund or a void this must reference the capture or the purchase.
cust_id	99/an	This is an optional field that can be sent as part of a purchase or preauth request. IT is searchable from the Moneris Merchant Interface. It is commonly used for policy number, membership number, student id or invoice number.
cavv		This is a value that is provided by the Moneris MPI or by a third party MPI. It is part of a VBV transaction.
compamount	9/decimal	Amount of the capture transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99.

19. Appendix B. Definition of MC Level 2/3 Request Fields

Mastercard Level 2 Request Fields				
Req	Variable Name	Field Name	Size/Type	Description
N	customerCode	Customer Code	17 A/N	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.
Y	taxAmount	Tax Amount	9 A/N	Mandatory. The GST/HST on the total purchase with 2 decimals. ie. 34567.89
N	freightAmount	Freight Amount	9 A/N	The freight on the total purchase Must have 2 decimals.
N	shipToPosCode	Ship To Postal Code	10 A/N	The postal code or zip code where goods will be delivered
Y	shipFromPosCode	Ship From Postal Code	10 A/N	The postal code or zip code from which items were shipped
N	dutyAmount	Duty Amount	9 A/N	The duty on the total purchase, Must have 2 decimals
N	altTaxAmtInd	Alternate Tax Amount Indicator	1 A/N	An indicator used to reflect alternate tax/or Provincial Sales Tax (PST) capture and reporting. Values are: **Y**=alternate tax included in total purchase amount **N**=alternate tax not included in total purchase amount **Space**=not supported
N	altTaxAmt	Alternat Tax Amount	9 A/N	The amount of the Provincial Sales Tax (PST) with 2 decimals
N	desCouCode	Destination Country Code	3 A/N	The country code where goods will be delivered. Left justified with trailing spaces. ie. CAN-Canada.
N	supData	Supplemental Data	17 A/N	This field may be used to provide additional data to the corporate customer
N	salTaxColInd	Sales Tax Collected Indicator	1 A/N	An indicator used to reflect GST/HST captured and reported. Values are: **Y** = GST/HST included in total purchase amount. **N** = GST/HST exempt not included in total purchase amount **Space** = Information is unknown

Mastercard Level 3 Request Fields				
Req	Variable Name	Field Name	Size/Type	Description
Y	productCode	Product Code	12 A/N	The product code of the individual item purchased **Mandatory, cannot contain all spaces or all zeroes.**
Y	itemDescription	Item Description	35 A/N	The description of the individual item purchased. **Mandatory, cannot contain all spaces or all zeroes.**
Y	itemQuantity	Item Quantity	5 A/N	The quantity of the individual item purchased. **Mandatory, cannot contain all spaces or all zeroes.**
Y	itemUom	Item unit of measure	3 A/N	A three-position unit of measurement code. **Mandatory ,cannot contain all spaces or all zeroes.**
Y	extItemAmount	Extended item amount	9 A/N	The amount of the item that is normally calculated as price times quantity. **Mandatory, cannot contain all spaces or all zeroes. Must contain 2 decimals.**
N	discountInd	Discount indicator	1 A/N	Values are: Y = amount is discounted N = amount is not discounted Space = not supported
N	discountAmt	Discount amount	9 A/N	Leading zeros with 2 decimals.
N	netGroIndForExtItemAmt	Net/gross indicator for extended item amount	1 A/N	Values are: Y =Item amount includes tax amount N = Item amount does not include tax amount Space = not supported
N	taxRateApp	Tax rate applied	A/N	This is a numeric decimal rate for GST/HST. May contain 2 decimals.
N	taxTypeApp	Tax type applied	A/N	Description of tax applied as per tax type and tax amount. Use (GST) or (HST)
N	taxAmount	Tax Amount	A/N	The GST/HST amount applied to item. Must have 2 decimals
N	debitCreditInd	Debit or Credit Indicator	A/N	Values are: D=extended item amount is a Debit. C=extended item amount is a Credit. Space=does not apply
N	AltTaxIdeAmt	Alternate Tax Identifier (Amount)	A/N	Insert the QST/PST tax amount. Must have 2 decimals.

20. Appendix C. Definition of Visa Level 2/3 Request Fields

Visa Level 2/3 Request Fields				
Req	Variable Name	Field Name	Size/Type	Description
	orderLevelGst	Order Level GST	9 A/N	The total Goods and Services Tax of the order. Must contain 2 decimal places.
	merchantGstNo	Merchant GST Number		The Merchant GST account number
	cri	Customer Reference Indentifier		Can contain a unique value to identify the client
	OrderLevelPst	Order Level PST	9 A/N	The total Provincial Sales Tax of the order. Must contain 2 decimal places.
	merchantPstNo	Merchant PST Number		The Merchant PST account number
	dutyAmount	Duty Amount	12 N	The duty on the total purchase. Must contain 2 decimal places.
	shipToPosCode	Ship To Postal Code	10 A/N	The postal code or zip code where goods will be delivered
	shipFromPosCode	Ship From Postal Code	10 A/N	The postal code or zip code from which items were shipped
	desCouCode	Destination Country Code	3 A/N	The country code where goods will be delivered. Left justified with trailing spaces. ie. CAN-Canada.
	vatRefNum	VAT Reference Number	15 N	Unique Value Added Tax Invoice Reference Number
	itemComCode	Item Commodity Code	12 A/N	Line item Comodity Code
	productCode	Product Code	12 A/N	Line item product code. 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”.
	itemDescription	Item Description	26 N	Line item description
	ItemQuatity	Item Quantity	12 N	Quantity of line item
	itemUofm	Item Unit of Measure	12 A/N	Unit of Measure
	unitCost	Unit Cost	12 N	Line item cost per unit. Must contain 2 decimal places.
	vatTaxAmt	GST/HST Amount	12 N	Amount of GST/HST for line item. (Do not include PST in the calculation) Must contain 2 decimal places.
	vatTaxRate	GST/HST Rate	4 N	GST/HST rate applied to line item (Do not include PST in the calculation). May contain 2 decimal places.
	discountAmt	Discount Amount	12 N	The discountAmt can only be set when the product code is set to “Discount”. When the product code is set to “Discount” then discountAmt cannot be blank. Must contain 2 decimal places.

21. Appendix D. Definition of American Express Level 2/3 Request Fields

American Express Level 2/3 Request Fields – Table 1 / Heading Fields				
Req	Variable Name	Field Name	Size/Type	Description
N	big04	Purchase Order Number	22 A/N	
N	big05	Release Number	30 A/N	
N	big10	Invoice Number	10 A/N	
Y	n101	Entity Identifier Code	2 A/N	• 'R6' - Requester (required) • 'BG' - Buying Group (optional) • 'SF' - Ship From (optional) • 'ST' - Ship To (optional) • '40' - Receiver (optional)
Y	n102	Name	40 A/N	**n101** / **n102 denotation** - 'R6' Requester Name - 'BG' Buying Group Name - 'SF' Ship From Name - 'ST' Ship To Name - '40' Receiver Name
N	n301	Address	40 A/N	Address
N	n401	City	30 A/N	City
N	n402	State or Province	2 A/N	State or Province
N	n403	Postal Code	15 A/N	Postal Code
Y	ref01	Reference Identification Qualifier	2 A/N	• 'VR' – Vendor ID Number • '14' – Master Account Number • '12' – Billing Account • '4C' – Shipment Destination Code (required) • 'CR' – Customer Reference Number
Y	ref02	Reference Identification	**ref01** / **ref02 size/type** -'VR' 10 A/N -'14' 10 A/N -'12' 30 A/N -'4C' 6 A/N -'CR' 17 A/N	**ref01** / **ref02 denotation** - '14' Amex CAP Number (optional) - '12' Billing Account (optional) - '4C' Ship-to-Zip or Canadian Postal Code (required) - 'CR' Cardmember Reference Number (optional)

American Express Level 2/3 Request Fields – Table 2 / Detail Fields				
Req	Variable Name	Field Name	Size/Type	Description
Y	it102	Line Item Quantity Invoiced	10 R	
Y	it103	Unit or Basis for Measurement Code	2 A/N	
Y	it104	Unit Price	15 R	
N	it105	Basis or Unit Price Code	2 A/N	

N	it10618	Product/Service ID Qualifier	2 A/N	• '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
N	it10719	Product/Service ID	**it10618** / **it10719 - size/type** - VC 20 A/N - PO 22 A/N - Other 30 A/N	
Y	txi01	Tax Type code	2 A/N	• 'CA' – City Tax (optional) • 'CP' – County/Parish Sales 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)
N	txi02	Monetary Amount	6 R	
N	txi03	Percent	10 R	
N	txi06	Tax Exempt Code	1 A/N	• '1' – Yes (Tax Exempt) • '2' – No (Not Tax Exempt) • 'A' – Labor Taxable, Material Exempt • 'B' – Material Taxable, Labor Exempt • 'C' – Not Taxable • 'F' – Exempt (Good / Services Tax) • 'G' – Exempt (Provincial Sales Tax) • 'L' – Exempt Local Service • 'R' – Recurring Exempt • 'U' – Usage Exempt
Y	pam05	Line Item Extended Amount	8 R	
Y	pid06	Line Item Description	80 A/N	

American Express Level 2/3 Request Fields – Table 3 / Summary Fields				
Req	Variable Name	Field Name	Size/Type	Description
Y	txi01	Tax Type code	2 A/N	• 'CA' – City Tax • 'CP' – County/Parish Sales Tax • 'CT' – County/Tax • 'EV' – Environmental Tax • 'GS' – Good and Services Tax (GST) • 'LS' – State and Local Sales Tax • 'LT' – Local Sales Tax • 'PG' – Provincial Sales Tax (PST) • 'SP' – State/Provincial Tax a.k.a. Quebec Sales Tax

				(QST) • 'ST' – State Sales Tax • 'TX' – All Taxes • 'VA' – Value-Added Tax a.k.a. Canadian Harmonized Sales Tax (HST)
Y	txi02	Monetary Amount	6 R	
N	txi03	Percent	10 R	
N	txi06	Tax Exempt Code	1 A/N	• '1' – Yes (Tax Exempt) • '2' – No (Not Tax Exempt) • 'A' – Labor Taxable, Material Exempt • 'B' – Material Taxable, Labor Exempt • 'C' – Not Taxable • 'F' – Exempt (Good / Services Tax) • 'G' – Exempt (Provincial Sales Tax) • 'L' – Exempt Local Service • 'R' – Recurring Exempt • 'U' – Usage Exempt

22. Appendix E. Definitions of Response Fields

Response Fields		
Variable Name	Size/Type	Description
ReceiptId	99 / an	order_id specified in request
ReferenceNum	18 / num	The reference number is an 18 character *string* that references the 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 should be stored by the merchant. The following illustrates the breakdown of this field where "660123450010690030" is the reference number returned in the message, "66012345" is the terminal id, "001" is the shift number, "069" is the batch number and "003" is the transaction number within the batch. Moneris Host Transaction identifier
ResponseCode	3 / num	Transaction Response Code < 50: Transaction approved >= 50: Transaction declined NULL: Transaction was not sent for authorization * If you would like further details on the response codes that are returned please see the Response Codes document available for download at http://www.eselectplus.com/downloadable-content.
ISO	2 / num	ISO response code
AuthCode	8 / an	Authorization code returned from the issuing institution
TransTime	##:##:##	Processing host time stamp
TransDate	yyyy-mm-dd	Processing host date stamp
TransType	an	Type of transaction that was performed
Complete	True/False	Transaction was sent to authorization host and a response was received
Message	100 / an	Response description returned from issuing institution.
TransAmount		
CardType	2 / alpha	Credit Card Type
Txn_number	20 / an	Gateway Transaction identifier
TimedOut	True/False	Transaction failed due to a process timing out
Ticket	n/a	reserved
RecurSucess	True/false	Indicates whether the transaction successfully registered.
CorporateCard	True/False/Null	Will return true if the card is a Corporate Card – this card is capable of accepting the subsequent Level 2/3 data transaction types. If it is false the card is not corporate, if the result is Null then you are not enrolled in Level2/3
MessageId	15 num / Null	This value is returned with a Visa Level 2/3 corporate card transaction. It is a unique transaction identifier.

23. Appendix F. Sample Receipt

Your order has been Approved
Print this receipt for your records

QA Merchant #1
3250 Bloor St West
Toronto Ontario
M8X2X9

1 800 987 1234
www.moneris.com

Transaction Type: Purchase

Order ID:	mhp3495435587		
Date/Time:	2002-10-18 11:27:48	Approval Code:	030012
Sequence Number:	660021630012090020	Response / ISO Code:	028/04
Amount:	12.04	APPROVED * =	

Item	Description	Qty	Amount	Subtotal
cir-001	Med Circle	1	2.01	2.01
tri-002	Big triangle	1	1.01	1.01
squ-003	small square	2	1.01	3.02
			Shipping:	4.00
			GST :	1.00
			PST :	1.00
			Total:	12.04 CAD

Bill To:

Test Customer

123 Main St
Springfield
ON
Canada
M1M 1M1
tel: 416 555 1111
fax: 416 555 1111

Ship To:

Test

1 King St
Bakersville
ON
Canda
M1M 1M1
tel: 416 555 2222
fax: 416 555 2222

Special Instructions

Knock on Back door when delivering
E-Mail Address:eselectsupport@moneris.com

Refund Policy

30 Days - Must be unopened, 10% restocking charge.

eSELECTplus™

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

Moneris and the Moneris Solutions logo are registered trademarks of Moneris Solutions Corporation.

Any software, hardware and or technology products named in this document are claimed as trademarks or registered trademarks of their respective companies.

Printed in Canada.

10 9 8 7 6 5 4 3 2 1