Merchant Integration Guide

PHP API (Mag Swipe)

v. 1.1.4

Moneris SOLUTIONS

Table of Contents

**** PLEASE READ CAREFULLY****

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.

1. About this Documentation

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

2. System and Skill Requirements

In order to use the PHP API your system will need to have the following:
1. PHP 4 or later
2. Port 443 open
3. OpenSSL
4. cURL - PHP interface - 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 programming language

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 test environment into production as outlined in this document

4. Transaction Types and Transaction Flow

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

Basic Transactions

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

Open Totals – (Current Batch Report) When an Open Totals is performed it returns the details about the currently open Batch. This transaction is similar to the Batch Close, though it does not close the Batch for settlement.

Mag Swipe Transactions

Mag Swipe Purchase – (sale) The Mag Swipe Purchase transaction requires a credit card to be swiped. It then verifies funds on the customer's card, removes the funds and readies them for deposit into the merchant's account.

Mag Swipe PreAuth – (authorisation / preauthorisation) The Mag Swipe PreAuth requires a credit card to be swiped. It then 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 Mag Swipe PreAuth so that they may be settled in the merchant's account a Mag Swipe Capture must be performed.

Mag Swipe Capture – (Completion / PreAuth Completion) Once a Mag Swipe PreAuth is obtained the funds that are locked need to be retrieved from the customer's credit card. The Mag Swipe Capture retrieves the locked funds and readies them for settlement into the merchant's account.

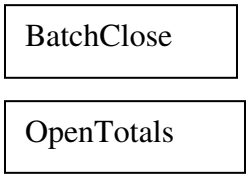
Mag Swipe Void – (Correction / Purchase Correction) Mag Swipe Purchases and Mag Swipe Captures can be voided the same day* that they occur. A Mag Swipe Void must be for the full amount of the transaction and will remove any record of it from the cardholder's statement.

Mag Swipe Refund – (Credit) A Mag Swipe Refund can be performed against a Mag Swipe Purchase or a Mag Swipe Capture to refund any part, or all of the transaction.

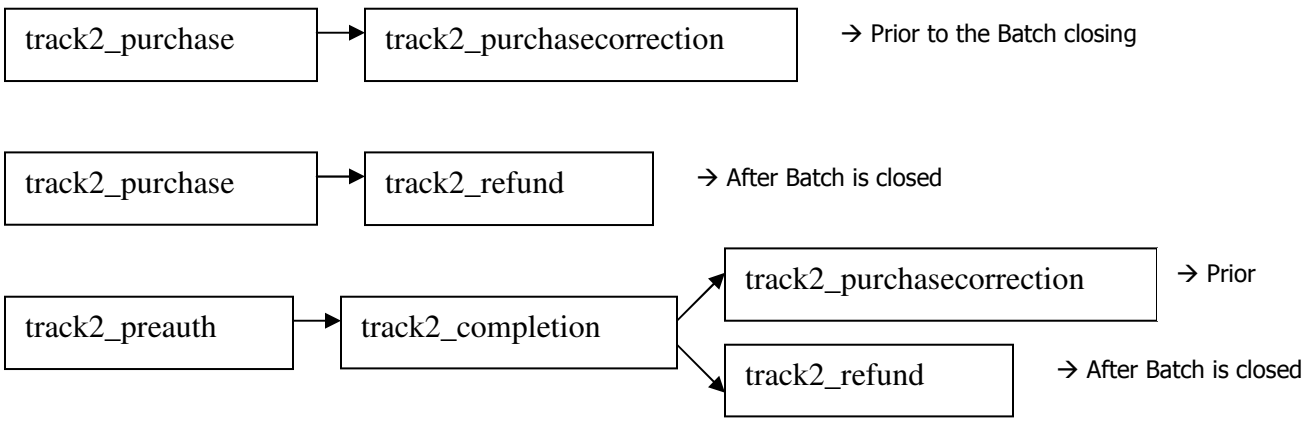
Mag Swipe Independent Refund – (Credit) A Mag Swipe Independent Refund requires a credit card to be swiped. It can be performed to credit money to this particular credit card. This transaction does not require a prior Mag Swipe Purchase or Mag Swipe Capture.

* A Void can be performed against a transaction as long as the batch that contains the original transaction remains open.

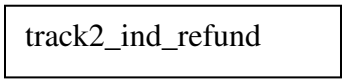
Process Flow for Basic Transactions



Process Flow for Mag Swipe Credit Card Transactions



Transactions with no Follow-on Required



5. Mag Swipe Transaction Examples

Included below is the sample code for the Mag Swipe transactions that can be found in the "Examples" folder of the PHP API download. Mag Swipe transactions allow the user to swipe their 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 using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction.

Mag Swipe Purchase

This example requires several variables (store_id, api_token, order_id, amount, track2 and/or pan, expdate, and pos_code). Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php

require "../msrMpgClasses.php";

/************************** Request Variables *****************************/

$store_id='store1';
$api_token='yesguy';

/************************* Transactional Variables ***************************/

$type='track2_purchase';
$cust_id='CUST13343';         ## This is an optional field
$order_id='ord-'.date("dmy-G:i:s");
$amount='10.00';
$pan='';
$expiry_date='';
$pos_code='00';

/************ Swipe card and read Track1 and/or Track2 ***********************/

$stdin = fopen("php://stdin", 'r');

print ("Please swipe your card:\n");
$track1 = fgets ($stdin);

$startDelim = ";";
$firstChar = $track1{0};
$track = '';

if($firstChar==$startDelim)
{
        $track = $track1;
}
else
{
        print ("\nPlease swipe your card again:\n");
        $track2 = fgets ($stdin);
        $track = $track2;
}

$track = trim($track);

/********************** Transactional Associative Array **********************/

$txnArray=array('type'=>$type,
                    'order_id'=>$order_id,
                    'cust_id'=>$cust_id,
                    'amount'=>$amount,
                    'track2'=>$track,
                    'pan'=>$pan,
                    'expdate'=>$expiry_date,
                    'pos_code'=>$pos_code
                );
```

```
/************************** Transaction Object ****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/*************************** Request Object ******************************/

$mpgRequest = new msrMpgRequest($mpgTxn);

/************************** HTTPS Post Object *****************************/

$mpgHttpPost = new msrMpgHttpsPost($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());

?>
```

Mag Swipe PreAuth

The Mag Swipe PreAuth requires several variables (store_id, api_token, order_id, amount, track2 and/or pan, expdate, and pos_code). Please refer to Appendix A. Definition of Request Fields for variable definitions. In the example below the test environment store_id (store1) and api_token (yesguy) have been hard coded, and highlighted in the msrMpgHttpsPost object.

```
<?php

require "../msrMpgClasses.php";

/************************* Transactional Variables ****************************/

$type='track2_preauth';
$cust_id='CUST13343';         ## This is an optional field
$order_id='ord-'.date("dmy-G:i:s");
$amount='10.00';
$pan='';
$expiry_date='';
$pos_code='00';

/************ Swipe card and read Track1 and/or Track2 ***********************/

$stdin = fopen("php://stdin", 'r');

print ("Please swipe your card:\n");
$track1 = fgets ($stdin);

$startDelim = ";";
$firstChar = $track1{0};
$track = '';

if($firstChar==$startDelim)
{
       $track = $track1;
}
else
{
       print ("\nPlease swipe your card again:\n");
       $track2 = fgets ($stdin);
       $track = $track2;
}

$track = trim($track);

/************************ Transactional Associative Array ***********************/

$txnArray=array(
               'type'=>$type,
               'order_id'=>$order_id,
               'cust_id'=>$cust_id,
               'amount'=>$amount,
               'track2'=>$track,
               'pan'=>$pan,
               'expdate'=>$expiry_date,
               'pos_code'=>$pos_code
               );

/***************************** Transaction Object *****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object *******************************/

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

```
/****************************** HTTPS Post Object *****************************/

$mpgHttpPost = new msrMpgHttpsPost('store1','yesguy',$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());

?>
```

Mag Swipe Capture

The Mag Swipe Capture (track2_completion) transaction is used to secure the funds locked by a 'track2_preauth' transaction. When sending a 'track2_completion' request you will need two pieces of information from the original 'track2_preauth' – the order_id and the txn_number from the returned response; it does not require the customer to re-swipe the credit card.

```
<?php

require "../msrMpgClasses.php";

/************************** Request Variables *****************************/

$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transactional Variables ***************************/

$orderid=$argv[3];
$txnnumber=$argv[4];

$compamount='1.00';

/********************** Transactional Associative Array **********************/

$txnArray=array('type'=>'track2_completion',
                'txn_number'=>$txnnumber,
                'order_id'=>$orderid,
                'comp_amount'=>$compamount
               );


/************************** Transaction Object ****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object *****************************/

$mpgRequest = new msrMpgRequest($mpgTxn);

/***************************** HTTPS Post Object *****************************/

$mpgHttpPost  =new msrMpgHttpsPost($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());

?>
```

Mag Swipe Void

The Mag Swipe Void (track2_purchasecorrection) 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 'track2_purchasecorrection' the order_id and txn_number from the 'track2_completion' or 'track2_purchase' are required; it does not require the customer to re-swipe the credit card.

```
<?

require "../msrMpgClasses.php";

/************************** Request Variables *****************************/

$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transactional Variables ***************************/

$orderid=$argv[3];
$txnnumber=$argv[4];

/********************** Transactional Associative Array **********************/

$txnArray=array(
               'type'=>'track2_purchasecorrection',
               'txn_number'=>$txnnumber,
               'order_id'=>$orderid
               );



/************************** Transaction Object ****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object *****************************/

$mpgRequest = new msrMpgRequest($mpgTxn);

/***************************** HTTPS Post Object *****************************/

$mpgHttpPost  =new msrMpgHttpsPost($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());

?>
```

Mag Swipe Refund

The Mag Swipe Refund (track2_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 'track2_refund' you will require the order_id and txn_number from the original 'track2_completion' or 'track2_purchase'; it does not require the customer to re-swipe the credit card.

```
<?php

require "../msrMpgClasses.php";

/************************** Request Variables *****************************/

$store_id=$argv[1];
$api_token=$argv[2];

/************************* Transactional Variables ***************************/

$orderid=$argv[3];
$txnnumber=$argv[4];
$amount = '1.00';

/********************** Transactional Associative Array **********************/

$txnArray=array('type'=>'track2_refund',
               'txn_number'=>$txnnumber,
               'order_id'=>$orderid,
               'amount'=>$amount,
               );

/*************************** Transaction Object ******************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object ********************************/

$mpgRequest = new msrMpgRequest($mpgTxn);

/***************************** HTTPS Post Object *****************************/

$mpgHttpPost  =new msrMpgHttpsPost($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());

?>
```

Mag Swipe Independent Refund

The Mag Swipe Independent Refund (track2_ind_refund) will credit a specified amount to the cardholder's credit card. The Mag Swipe Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card will need to be swiped to provide the track2. In the example below the values are hard coded and highlighted in the appropriate locations. The transaction format is almost identical to a Mag Swipe Purchase or a Mag Swipe PreAuth.

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

/************ Swipe card and read Track1 and/or Track2 ***********************/

$stdin = fopen("php://stdin", 'r');

print ("Please swipe your card:\n");
$track1 = fgets ($stdin);

$startDelim = ";";
$firstChar = $track1{0};
$track = '';

if($firstChar==$startDelim){
        $track = $track1;
}
else{
        print ("\nPlease swipe your card again:\n");
        $track2 = fgets ($stdin);
        $track = $track2;
}

$track = trim($track);
/*********************** Transactional Associative Array **********************/
$txnArray=array('type' =>'track2_ind_refund',
                'order_id' =>'ord-'.date("dmy-G:i:s"),
                'cust_id' => 'CUST13343',           ## This is an optional field
                'amount' =>'1.00',
                'track2'=> $track,
                'pan' =>'',
                'expdate' =>'',
                'pos_code' =>'00'
               );
/***************************** Transaction Object *****************************/
$mpgTxn = new msrMpgTransaction($txnArray);
/****************************** Request Object ********************************/
$mpgRequest = new msrMpgRequest($mpgTxn);
/***************************** HTTPS Post Object ******************************/
$mpgHttpPost = new msrMpgHttpsPost('store1','yesguy',$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());

?>
```

6. Mag Swipe Transactions with Extra Features - Examples

In the previous section the instructions were provided for the Mag Swipe transaction set. eSELECTplus also provides the ability to store customer and order details.

Mag Swipe Purchase (with Customer and Order details)

Below is an example of sending a Mag Swipe Purchase with the customer and order details. If one piece of information is sent then all fields must be included in the request. Unwanted fields need to be blank. The identical format is used for Mag Swipe PreAuth with the exception of transaction type which changes from 'track2_purchase' to 'track2_preauth'. Customer details can only be sent with Mag Swipe Purchase and PreAuth. ***Please note that the msrMpgCustInfo fields are not used for any type of address verification or fraud check.***

```
<?

require "../msrMpgClasses.php";

/************************ Request Variables ***************************/

$store_id='store1';
$api_token='yesguy';

/********************* Transactional Variables ************************/

$type='track2_purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='My Cust ID';               ## This is an optional field
$amount='17.00';
$pan='';
$expiry_date='';
$pos_code='00';

/************ Swipe card and read Track1 and/or Track2 ***********************/

$stdin = fopen("php://stdin", 'r');

print ("Please swipe your card:\n");
$track1 = fgets ($stdin);

$startDelim = ";";
$firstChar = $track1{0};
$track = '';

if($firstChar==$startDelim)
{
        $track = $track1;
}
else
{
        print ("\nPlease swipe your card again:\n");
        $track2 = fgets ($stdin);
        $track = $track2;
}

$track = trim($track);

/******************* 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[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 msrMpgCustInfo();

/********************** 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],
                '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,         ## This is an optional field
                'amount'=>$amount,
                'track2'=>$track,
                'pan'=>$pan,
                'expdate'=>$expiry_date,
                'pos_code'=>$pos_code
                );

/********************** Transaction Object ****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/******************** Set Customer Information ***********************/

$mpgTxn->setCustInfo($mpgCustInfo);

/************************ Request Object ****************************/

$mpgRequest = new msrMpgRequest($mpgTxn);

/************************ HTTPS Post Object ****************************/

$mpgHttpPost = new msrMpgHttpsPost($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());
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());

?>
```

7. Basic Transaction Examples

Included below is the sample code for the basic transactions that can be found in the "Examples" folder of the PHP API download.

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 automatically close your Batch 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.

```
<?

require "../msrMpgClasses.php";

/************************** Request Variables *******************************/

$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transactional Variables ***************************/

$ecr_number=$argv[3];

/********************** Transactional Associative Array *********************/

$txnArray=array('type'=>'batchclose',
                'ecr_number'=>$ecr_number
               );

/************************** Transaction Object ******************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object *******************************/

$mpgReq = new msrMpgRequest($mpgTxn);

/************************** HTTPS Post Object *******************************/

$mpgHttpPost = new msrMpgHttpsPost($store_id,$api_token,$mpgReq);

/****************************** Response ************************************/

$mpgResponse = $mpgHttpPost->getMpgResponse();

$creditCards = $mpgResponse->getCreditCards($ecr_number);

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]) . "\n";
 }

?>
```

Open Totals

Open Totals allows the merchant to retrieve details about the currently open Batch. The response will include the transaction count and amount for each type of transaction. Open Totals returns a similar response to the Batch Close without closing the current Batch.

```
<?

require "../msrMpgClasses.php";

/************************** Request Variables *****************************/

$store_id = $argv[1];
$api_token = $argv[2];

/************************ Transactional Variables ************************/

$ecr_number = $argv[3];

/********************** Transactional Associative Array **********************/

$txnArray = array('type'=>'opentotals',
                  'ecr_number'=>$ecr_number
                 );

/************************** Transaction Object ****************************/

$mpgTxn = new msrMpgTransaction($txnArray);

/***************************** Request Object *****************************/

$mpgReq = new msrMpgRequest($mpgTxn);

/*************************** HTTPS Post Object ****************************/

$mpgHttpPost = new msrMpgHttpsPost($store_id,$api_token,$mpgReq);

/******************************* Response *********************************/

$mpgResponse = $mpgHttpPost->getMpgResponse();

$creditCards = $mpgResponse->getCreditCards($ecr_number);

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]) . "\n";
 }

?>
```

8. 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 B. Definitions of Response Fields.

To determine whether a transaction is successful or not the field that must be checked is ResponseCode. 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.

9. 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 test environment we cannot guarantee 100% availability. Also, please be aware that other merchants are using the test 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_id and api_token. 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**
store1	yesguy	DemoUser	password
store2	yesguy	DemoUser	password
store3	yesguy	DemoUser	password
store5	yesguy	DemoUser	password

When testing you may use the following test card numbers with any future expiry date.

Test Card Numbers	
Card Plan	**Card Number**
MasterCard	5454545454545454
Visa	4242424242424242
Amex	373599005095005
Diners	36462462742008
Track2	;5258968987035454=06061015454001060101?

To access the Merchant Resource Centre in the test environment go to https://esqa.moneris.com/mpg. 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 production 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.

The test environment will approve and decline transactions based on the penny value of the amount field.
For example, a transaction made for the amount of $399.00 or $1.00 will approve since the .00 penny value is set to approve in the test environment. 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 as well as the Test Environment eFraud Response table, available for download at http://www.eselectplus.ca/en/downloadable-content.

NOTE These responses may change without notice. Moneris Solutions recommends you regularly refer to our website to check for possible changes.

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

10. Transaction Receipt Requirements

Production of the receipt must begin when the appropriate response to the transaction request is received by the terminal. The transaction may be any of the following:

Pre-authorization
Pre-auth Completion
Purchase
Purchase Correction
Refund - *Notes:* *For Refunds, the word "Refund" ("Remise d'achat" in French) MUST appear after the dollar amount.*

The Amount MUST appear even if the transaction is declined or not completed. The Merchant number MUST NOT appear on the receipt.

The transaction receipt must be printed in the language of the cardholder. If the language is not returned as part of the transaction, the receipt should be printed in the default language of the terminal/ECR. The transaction must contain the following information:

- Merchant Name and Address
- Transaction Type (Purchase, Refund …| defined by *TransType*)
- Card Number (as defined by *Pan*)
- Card Type (Visa, MasterCard . . . | defined by *CardType*)
- Transaction Date and Time (format chosen by merchant | defined by *TransDate & TransTime*)
- Authorization Code (defined by *AuthCode*)
- Amount (defined by *Amount*)
- Response Message (contains *IsoCode*, message and *ResponseCode* - see below for further details)
- Reference Number (defined by *RefNum* & *Ecom*)

Credit card Purchase, Pre-authorization, and Pre-auth Completion transactions must also contain two extra items, these are:

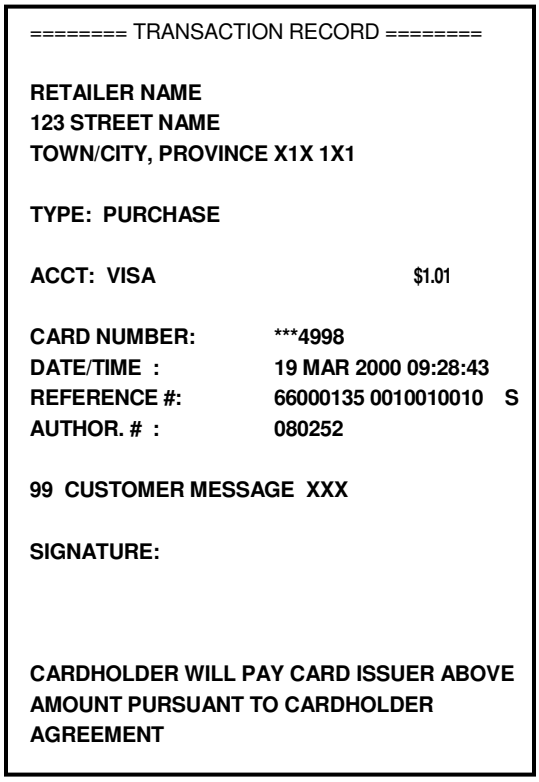
- Signature Line
- Cardholder Agreement Text

Credit Card Receipt:

- Credit card receipts must be printed only when the transaction is approved.
- Credit card number on the receipt must be masked, with only the last four digits printed (e.g.: ******1234).

The following is a suggested layout for an English language Credit card receipt using a 40-column printer. Two copies of the receipt are required; one for the cardholder and the other for the merchant. The layout examples illustrated are recommended so as to provide a clear and easily read receipt for the cardholder and to provide a consistent presentation to Moneris tracing staff.

Note: *The card entry Indicator for Purchase Completion transactions must match that of the original Pre-authorization. For example if the transaction was originally electronically pre-authorized with a swiped entry, then the completion message must print "S" or "Swiped" on the receipt.*



Note	Title	Explanation
1	Retailer Name	The name of the store.
2	Store Address	The postal address of the store, which must include the street, town/city, province and, if possible, the postal code.
3	Transaction Identification	The type of transaction, e.g. Purchase. Note, for Refund transactions, the word “Refund” (“Remise D’Achat” in French) MUST appear after the dollar amount.
4	Account Type	The type of Credit card: VISA MASTERCARD AMERICAN EXPRESS DINERS While 'AMEX' is the accepted abbreviation for 'American Express', it is preferable to use the full name of the card type for other Credit cards.
5	Amount	Total amount being paid by Credit card.
6	Primary Account Number	Cardholder's card number (PAN) with masking.
7	Transaction Date/time	The date may be in any format, but must include the day, month, and year. The time must be in 24-hour format.
8	Unique Transaction ID	66000135 = ECR number 001 = shift 133 = batch number

		001 = serial number 0 = control flag; may be either 0 or 1
9	Card Entry Indicator	Credit cards may be entered manually or swiped. If entered manually, must be 'M'; if swiped, must be 'S'. Card Entry Indicator for Pre-auth Completion transactions must match that of the original Pre-authorization. For example if the transaction was originally electronically pre-authorized with a swiped entry, then the completion message must print “S” or “Swiped” on the receipt.
10	Authorization Number	The approval number is only to be printed if the transaction is approved. If the transaction is declined, the title is to be printed, but leave the actual number blank.
11	Approval/Decline	See section entitled "Response Message".
12	Signature	The signature forms the cardholder’s authority for the transaction. ***Note:*** *Only the merchant’s copy requires the cardholder’s signature.* For Refund and Purchase Correction transactions, the merchant must sign the Cardholder’s copy of the CTR.
13	Cardholder Agreement Message	This text (see below for exact wording in English and French) is required on CTRs for the following types of transactions: - Purchase - Pre-Authorization - Pre-Authorization Completion - Refund Correction This text is NOT required on CTRs for the following types of transactions: - Refund - Purchase Correction **English text:** “Cardholder will pay card issuer above amount pursuant to Cardholder Agreement.” **French text:** “Le Titulaire versera ce montant a L’emetteur conformement au contrat adherent.”

NOTE Please note a split tender transaction is required to have separate lines for each credit/debit card type.

Response Message:

Format: ii message rrr (where ii = iso code, *message* = defined below, rrr = response code)
Message Definition:

Credit Card

If the Response Code is between 00 and 49 (inclusive) (‘0’ <= Response Code =< ‘49’)
Message(English) = “APPROVED - THANK YOU”
Message(French) = “APPROUVEE – MERCI”

Any other response code (including ‘null’ and empty)
Message(English) = “TRANSACTION NOT APPROVED”
Message(French) = “OPERATION REFUSEE”

11. Certification Requirements

Once 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. Please contact the eSELECTplus integration desk for the certification test case sheet. Be sure to include the application version of your product. Any changes to the product with respect to the payment functionalities will require re-certification. Once all the certification requirements are met, we will provide you with an official certification letter.

12. 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 then click on 'Activate'. In this process you will need to create an administrator account that you will use to log into the Merchant Resource Centre to access and administer your eSELECTplus 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 Resource Centre user, please log on to the Interface by clicking the "eSELECTplus" 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. Also, this is where you may locate your production API Token.

13. How Do I Configure My Store For Production?

Once you have completed you testing you are ready to point your store to the production host. You will need to edit the msrMpgClasses.php file and change the $Globals array as highlighted below in red. You will also need to change the store_id to reflect your production store ID as well the api_token must be changed to your production token to reflect the token that you received during activation.

PRODUCTION	DEVELOPMENT
var $Globals=array('MONERIS_PROTOCOL' => 'https', 'MONERIS_HOST' => **'www3.moneris.com'**, 'MONERIS_PORT' =>'443', 'MONERIS_FILE' => '/gateway2/servlet/MpgRequest', 'API_VERSION' =>'MPG Version 2.01 (MSR)', 'CLIENT_TIMEOUT' => '60');	var $Globals=array('MONERIS_PROTOCOL' => 'https', 'MONERIS_HOST' => **'esqa.moneris.com'**, 'MONERIS_PORT' =>'443', 'MONERIS_FILE' => '/gateway2/servlet/MpgRequest', 'API_VERSION' =>'MPG Version 2.01 (MSR)', 'CLIENT_TIMEOUT' => '60');

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 for download at http://www.eselectplus.ca/en/downloadable-content.

14. 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.**

15. Appendix A. Definition of Request Fields

Request Fields		
Variable Name	**Size/Type**	**Description**
order_id	50 / 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 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
pos_code	2 / num	Under normal presentment situations the value should be '00'. If the solution is not "merchant and cardholder present" please call the support desk and we will provide the proper POS Code.
txn_number	255 / varchar	Used when performing follow on transactions - this must be filled with the value that was returned as the Txn_number in the response of the original transaction. When performing a Capture this must reference the PreAuth. When performing a Refund or a Void this must reference the Capture or the Purchase.
cust_id	50/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 Resource Centre. It is commonly used for policy number, membership number, student ID or invoice number.
track2		This is a string that is retrieved from the mag swipe of a credit card by swiping the credit card through a card reader. It is part of a mag swipe/track2 transaction.
ecr_number	8/an	Terminal ID – no spaces or dashes. The ECR/terminal number is 8 digits long and generally begins with the number "66" (ex. 66012345). The terminal ID is assigned when the merchant account is first set up. This number will be returned in the response as part of the ReferenceNum.

16. Appendix B. Definitions of Response Fields

Response Fields		
Variable Name	**Size/Type**	**Description**
order_id	50 / 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.
ReponseCode	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.ca/en/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

17. Appendix C. CustInfo Fields

Recur Request Fields		
Variable Name	**Size/Type**	**Description**
Billing and Shipping Information		
NOTE: The fields for billing and shipping information are identical – please refer to section 6 for an example.		
first_name	30 / an	
last_name	30 / an	
company_name	30 / an	
address	30 / an	
city	30 / an	
province	30 / an	
postal_code	30 / an	
country	30 / an	
phone_number	30 / an	
fax	30 / an	
tax1	30 / an	
tax2	30 / an	
tax3	30 / an	
shipping_cost	30 / an	
Item Information		
NOTE: You may send multiple items – please refer to section 6 for an example.		
name	15 / an	
quantity	4 / num	You must send a quanitity > 0 or the item will not be added to the item list (ie. minimum 1, maximum 9999)
product_code	10 / an	
extended_amount	9 /decimal	This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
Extra Details		
email	50 / an	
instructions	50 / an	


NOTE The data sent in Billing and Shipping Address fields will not be used for any address verification.

18. Appendix D. Error Messages

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. A 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, either because they are offline or you are unable to connect to the internet. A 'NULL' can also be returned when a transaction message is improperly formatted.

Below are error messages that are returned in the Message field of the response.

Message: XML Parse Error in Request: <System specific detail>
Cause: For some reason an improper XML document was sent from the API to the servlet

Message: XML Parse Error in Response: <System specific detail>
Cause: For some reason an improper XML document was sent back from the servlet

Message: Transaction Not Completed Timed Out
Cause: Transaction times out before the host responds to the gateway

Message: Request was not allowed at this time
Cause: The host is disconnected

Message: Could not establish connection with the gateway:
<System specific detail>
Cause: Gateway is not accepting transactions or server does not have proper access to internet

Message: Input/Output Error: <System specific detail>
Cause: Servlet is not running

Message: The transaction was not sent to the host because of a duplicate order id
Cause: Tried to use an order id which was already in use

Message: The transaction was not sent to the host because of a duplicate order id
Cause: Expiry Date was sent in the wrong format