Merchant Integration Guide

PHP API

V 1.2.6

Moneris SOLUTIONS

Revision Number	Date	Description
V1.1.0	January 2, 2007	-Document edited for coherence
V1.1.1	February 4, 2008	-Section 6. Transaction with Extra features: Purchase with CVD and AVS (eFraud) -Added CVD note. -Appendix A. Definition of Request Fields – Added CVD note. -Appendix E. Card Validation Digits (CVD) – Added CVD note.
V1.1.2	June 3, 2008	-Section 6. Transaction with Extra Features – Added Recur Billing Update (Customer Information) -Appendix C. Recur Fields -Added Recur Update Request Fields. -Added Recur Update Response codes.
V1.1.3	October 10, 2008	-Section 6. Transaction with Extra Features – Corrected Recur Update (Customer Information) -Section 2. System and Skill Requirement – Added PCI & PA-DSS requirements note.
V1.2.0	February 27, 2009	-Corrected Document Version number
V1.2.1	July 18, 2011	-New download link updated in various locations http://www.eselectplus.ca/en/downloadable-content -Section 8. How do I test my Solution? –Added CA Root Certificate File.
V1.2.2	September 9, 2011	-Section 4. Transaction Types and Transaction Process Flow -Added Process Flow for PreAuth/ReAuth/capture Transactions -Section 5. Transaction Examples -PreAuth (basic) – Added PreAuth reversal note -Added ReAuth example -Capture – Added PreAuth reversal note -Section 6. Transaction with Extra features: Purchase(with VbV) -Added enhanced AVS values -Section 6. Transaction with Extra features -Added Purchase with Status Check -Appendix A. Definition of Request Fields -Added orig_order_id -Added enhanced AVS request fields -Added status_check request field -Appendix B. Definitions of Response Fields -Added CavvResultCode response field -Added ITDResponse response field -Added StatusCode response field -Added StatusMsg response field -Added Appendix C. CustInfo Fields -Appendix F. Card Validation Digits (CVD) -Added American Express/JCB response codes -Appendix G. Address Verification Services(AVS) -Added American Express/JCB response codes -Appendix H. additional Information for CVD and AVS -Added American Express -Added Appendix I. CAVV Result Code
V1.2.3	March 26, 2012	-Section 4. Transaction Types and Transaction Flow -Added Card Verification -Section 5. Transaction Examples -Added Card Verification example -Section 8.How do I test my Solution? -Added Card verification Test Card numbers. -Appendix A. Definition of Request Fields -Added new Variable Name (dynamic_descriptor) -Appendix B. Definitions of Response Fields -Removed variable Name CardLevelResult. -Removed Appendix J. Card Level Result value
V1.2.4	January 25, 2013	- Section 4. Transaction Types and Transaction Flow - Updated PreAuth description - Updated Capture description

		- Removed ReAuth description - Removed Process Flow diagram - Section 5. Transaction Examples - Removed ReAuth - Updated PreAuth (basic) description - Updated Capture description - Updated Void description - Section 6. Transaction with Extra features - Examples - Added Capture (setShipIndicator) - Added Purchase Correction (SetShipIndicator)
V1.2.5	June 28, 2013	- Section 4 – Updated Transaction Types description - Section 5 & 6 – Updated different transaction types sample codes - Added support for dynamic_descriptor request variable to additional transaction types - Added IsVisaDebit response variable - Section 6 – Transaction with Extra features – Examples - Removed variable (setShipIndicator) -Appendix A – Updated the dynamic descriptor definition -Appendix B – Added IsVisaDebit definition - Appendix F and G – Moved the AVS and CVD response codes to a separate document -New download link updated in various locations: https://developer.moneris.com/
V1.2.6	July 29, 2014	-Removed references to setShipIndicator variable

Table of Contents

1. About this Documentation 6

2. System and Skill Requirements 6

3. What is the Process I will need to follow? 7

4. Transaction Types and Transaction Flow 7

Process Flow for PreAuth / ReAuth / Capture Transactions 9

5. Transaction Examples 10

Purchase (basic) 10
PreAuth (basic) 11
ReAuth 12
Capture 13
Void 14
Refund 15
Independent Refund 16
Batch Close 17
Open Totals 18
Card Verification 19

6. Transaction with Extra features - Examples 20

Purchase (with Customer and Order details) 20
CavvPurchase (Purchase with Verified by Visa – VBV or MasterCard SecureCode - MCSC) 23
Purchase (with Recurring Billing) 24
Recur Update 26
Purchase with CVD and AVS (eFraud) 27
Purchase with Status Check 29

7. What Information will I get as a Response to My Transaction Request? 31

8. How Do I Test My Solution? 31

9. What Do I Need to Include in the Receipt? 32

10. How Do I Activate My Store? 33

11. How Do I Configure My Store For Production? 33

12. How Do I Get Help? 33

13. Appendix A. Definition of Request Fields 34

14. Appendix B. Definitions of Response Fields 36

15. Appendix C. CustInfo Fields 38

16. Appendix D. Recur Fields 39

17. Appendix E. Error Messages 42

18. Appendix F. Card Validation Digits (CVD) & Address Verification Service (AVS). 43

Card Validation Digits (CVD) 43
Address Verification Service (AVS) 43
Additional Information for CVD and AVS 43

19. Appendix G. CAVV Result Code 44

20. Appendix H. Sample Receipt 45

**** 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 credit card transactions. In particular, it describes the format for sending transactions and the corresponding responses you will receive. If you are interested in also being able to accept INTERAC payments via your online application, please refer to the PHP API with INTERAC Online Payment document found at: https://developer.moneris.com

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

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.

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 https://developer.moneris.com 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.

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

CavvPurchase – (VBV/MCSC sale): CavvPurchase transaction is performed using cavv value obtained by performing Verified By Visa/MasterCard Securecode transaction on the card .This transaction verifies the validity of the cavv value, removes the funds and readies them for deposit into the merchant's account. This transaction is only applicable to Visa and MasterCard.

PreAuth – (authorisation / preauthorisation): PreAuth transaction 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's account a Capture must be performed. A PreAuth may only be Captured once. If you don't intend to capture a PreAuth, you must reverse it by performing a capture for zero dollar (0.00).

CavvPreAuth – (VBV/MCSC sale): CavvPreAuth transaction is performed using cavv value obtained by performing Verified By Visa/MasterCard Securecode transaction on the card . This transaction verifies the validity of the cavv value 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 this CavvPreAuth so that they may be settled in the merchant's account a Capture must be performed. A CavvPreAuth may only be captured once. This transaction is only applicable to Visa and MasterCard.

ReAuth – (reauthorisation): A PreAuth may only be Captured once. If the PreAuth is Captured for less than the original amount, the ReAuth will allow the merchant to verify and lock the remaining funds on the customer's credit card, so they may also be Captured. To retrieve the funds from a ReAuth so that they may be settled in the merchant's account, a Capture must be performed. A ReAuth can only be Captured once however if the Capture was for less than the ReAuth amount, another ReAuth will allow the merchant to verify and lock the remaining funds on the customer's credit card, so they may also be Captured.

Capture – (Completion / PreAuth Completion): A PreAuth and a ReAuth can only be captured once. Once a PreAuth or a ReAuth is obtained the funds that are locked need to be retrieved from the customer's credit card. The Capture retrieves the locked funds and readies them for settlement into the merchant's account. Please note that the crypt value (ECI indicator) sent in the capture must match the crypt value sent in the original PreAuth or ReAuth transaction. For CavvPreAuth, the capture cypt type to be used should be determined by the crypt value or the Cavv result code in the response.

Void – (Correction / Purchase 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's statement.

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

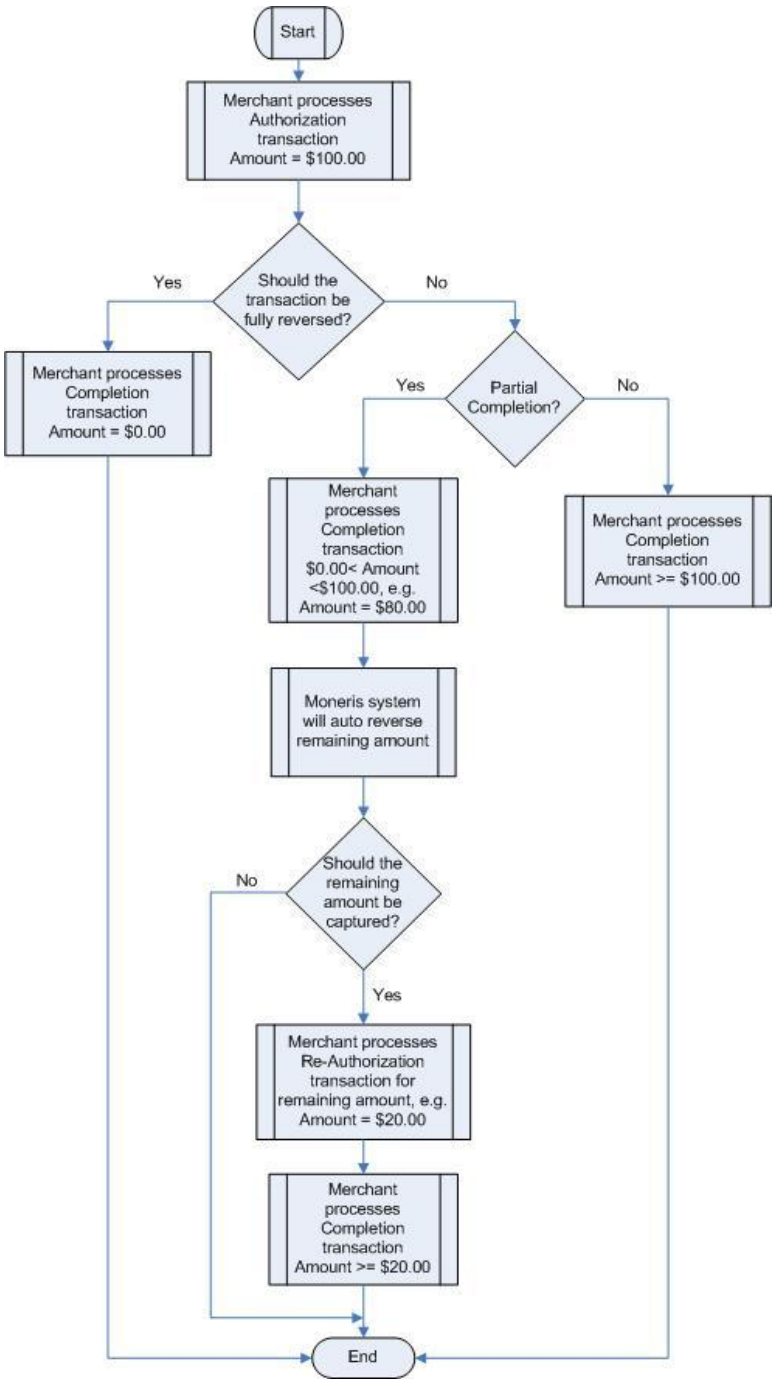
Batch Close – (End of Day / Settlement): When a batch close is performed it takes the monies from all Purchase, Capture and Refund transactions so they will be deposited or debited the following business day. For funds to be deposited the following business day the batch must close before 11pm EST.

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.

Card Verification – (Account Status Inquiry): Card Verification verifies the validity of the credit card, expiry date and any additional details, such as the Card Verification Digits or Address Verification details. It does not verify the available amount or lock any funds on the credit card. This transaction is only applicable for Visa and MasterCard.

* 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.

Process Flow for PreAuth / ReAuth / Capture Transactions



5. Transaction Examples

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

Purchase (basic)

In the purchase example we require several variables (store_id, api_token, order_id, amount, pan, expdate, and crypt_type). There are also a number of optional fields such as cust_id and dynamic_descriptor available. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php
## Example php -q TestPurchase.php store1
require "../mpgClasses.php";

$store_id='store5';
$api_token='yesguy';

$type='purchase';
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='4242424242424242';
$expiry_date='1111';
$crypt='7';
$dynamic_descriptor='123';
$status_check = 'false';

$txnArray=array('type'=>$type,
                  'order_id'=>$order_id,
                  'cust_id'=>$cust_id,
                  'amount'=>$amount,
                  'pan'=>$pan,
                  'expdate'=>$expiry_date,
                  'crypt_type'=>$crypt,
                  'dynamic_descriptor'=>$dynamic_descriptor
               );
$mpgTxn = new mpgTransaction($txnArray);

$mpgRequest = new mpgRequest($mpgTxn);

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

PreAuth (basic)

The Pre-Auth is similar to the Purchase transaction. The difference between a Purchase and a Pre-Auth is that with a Purchase the funds will be settled to the bank after the batch has been closed while with a Pre-Auth the funds are only “reserved” and a second step (a Capture) is required to have the funds deposited to the bank. Like the Purchase example, PreAuth’s require several variables (store_id, api_token, order_id, amount, pan, expdate, and crypt_type). There are also optional fields such as cust_id and dynamic_descriptor available. Please refer to Appendix A. Definition of Request Fields for variable definitions. For a process flow, please refer to Process Flow for PreAuth / ReAuth / Capture Transactions

A PreAuth transaction **must** be reversed within 72hrs if it is not to be captured (e.g. due to cancellation of order, error in order, or not able to fulfil). To reverse an authorization, please refer to the Capture transaction

```
<?php
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestPreAuth.php store1 yesguy

require "../mpgClasses.php";

$store_id="store5";
$api_token="yesguy";

## step 1) create transaction hash ###
$txnArray=array('type'=>'preauth',
         'order_id'=>'ord-'.date("dmy-G:i:s"),
         'cust_id'=>'my cust id',
         'amount'=>'1.00',
         'pan'=>'4242424242424242',
         'expdate'=>'0806',
         'crypt_type'=>'7',
         'dynamic_descriptor'=>'123456'
           );
## 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 ##
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType()."<br>");
print("\nTransAmount = " . $mpgResponse->getTransAmount()."<br>");
print("\nTxnNumber = " . $mpgResponse->getTxnNumber()."<br>");
print("\nReceiptId = " . $mpgResponse->getReceiptId()."<br>");
print("\nTransType = " . $mpgResponse->getTransType()."<br>");
print("\nReferenceNum = " . $mpgResponse->getReferenceNum()."<br>");
print("\nResponseCode = " . $mpgResponse->getResponseCode()."<br>");
print("\nISO = " . $mpgResponse->getISO()."<br>");
print("\nMessage = " . $mpgResponse->getMessage()."<br>");
print("\nAuthCode = " . $mpgResponse->getAuthCode()."<br>");
print("\nComplete = " . $mpgResponse->getComplete()."<br>");
print("\nTransDate = " . $mpgResponse->getTransDate()."<br>");
print("\nTransTime = " . $mpgResponse->getTransTime()."<br>");
print("\nTicket = " . $mpgResponse->getTicket()."<br>");
print("\nTimedOut = " . $mpgResponse->getTimedOut()."<br>");
print("\nIsVisaDebit = " . $mpgResponse->getIsVisaDebit());
?>
```

ReAuth

The ReAuth is similar to the PreAuth with the addition of the 'original_order_id' with the exception of the transaction type. It is 'ReAuth' instead of 'PreAuth'. Like the PreAuth example, ReAuth's require several variables (store_id, api_token, order_id, amount, orig_order_id, txn_number, and crypt). There are also optional fields such as cust_id and dynamic_descriptor available. Please refer to Appendix A. Definition of Request Fields for variable definitions.

Please note, a PreAuth may only be Captured once for less than, equal to, or greater than the original PreAuth amount. If the PreAuth is captured for less than its total amount, then a ReAuth is first required to be able to capture the remainder. The ReAuth references the original transaction by the orig_order_id and will only allow the merchant to re-authorize funds on the credit card used in the original transaction for no more than the upcaptured amount. For a process flow, please refer to Process Flow for PreAuth / ReAuth / Capture Transactions

```
import java.io.*;
import java.util.*;
import java.net.*;
import JavaAPI.*;

public class TestReAuth
{
  public static void main(String args[]) throws IOException
  {

        String host = "esqa.moneris.com";
        String store_id = "moneris";
        String api_token = "hurgle";

        String order_id;                    // will prompt for user inputs
        String cust_id = "Hilton_1";
        String amount = "1.00";
        String orig_order_id = "apr18test9";
        String txn_number = "59067-0_10";
        String crypt = "7";
        //String dynamic_descriptor = "123456";
       InputStreamReader isr = new InputStreamReader( System.in );
       BufferedReader stdin = new BufferedReader( isr );
       System.out.print( "Please enter an order ID: " );
       order_id = stdin.readLine();

        ReAuth r = new ReAuth (order_id, cust_id, amount, orig_order_id, txn_number, crypt);
        //r.setDynamicDescriptor(dynamic_descriptor);
        HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, r);

        try
        {
            Receipt receipt = mpgReq.getReceipt();
            System.out.println("CardType = " + receipt.getCardType());
            System.out.println("TransAmount = " + receipt.getTransAmount());
            System.out.println("TxnNumber = " + receipt.getTxnNumber());
            System.out.println("ReceiptId = " + receipt.getReceiptId());
            System.out.println("TransType = " + receipt.getTransType());
            System.out.println("ReferenceNum = " + receipt.getReferenceNum());
            System.out.println("ResponseCode = " + receipt.getResponseCode());
            System.out.println("BankTotals = " + receipt.getBankTotals());
            System.out.println("Message = " + receipt.getMessage());
            System.out.println("AuthCode = " + receipt.getAuthCode());
            System.out.println("Complete = " + receipt.getComplete());
            System.out.println("TransDate = " + receipt.getTransDate());
            System.out.println("TransTime = " + receipt.getTransTime());
            System.out.println("Ticket = " + receipt.getTicket());
            System.out.println("TimedOut = " + receipt.getTimedOut());
            System.out.println("IsVisaDebit = " + receipt.getIsVisaDebit());
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
  } } // end TestReAuth
```

Capture

The Capture (Completion) transaction is used to secure the funds locked by a PreAuth transaction. When sending a 'completion' request you will need two pieces of information from the original PreAuth – the order_id and the txn_number from the returned response.

To reverse the full amount of the PreAuth, please use the Capture transaction with a dollar amount of "0.00".

NOTE The crypt value (ECI indicator) sent in the capture must match the crypt value sent in the original PreAuth or ReAuth transaction. For CavvPreAuth, the capture cypt type to be used should be determined by the crypt value or the Cavv result code in the 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 yesguy original_order_id 76452-66-0

require "../mpgClasses.php";

$store_id=$argv[1];
$api_token=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];
$compamount=$argv[5];

## step 1) create transaction array ###
$txnArray=array('type'=>'completion',
         'txn_number'=>$txnnumber,
         'order_id'=>$orderid,
         'comp_amount'=>$compamount,
         'crypt_type'=>'7',
         'dynamic_descriptor'=>'123456'
           );
## step 2) create a transaction  object passing the hash created in
$mpgTxn = new mpgTransaction($txnArray);

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

## step 4) create mpgHttpsPost object which does an https post ##
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

Void

The Void (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 'PurchaseCorrection' 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 yesguy my_order_id 76452-77-0
##

require "../mpgClasses.php";

$store_id=$argv[1];
$api_token=$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',
         'dynamic_descriptor'=>'123456'
        );

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

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());

?>
```

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 yesguy my_order_id 45109-89-0
##

require "../mpgClasses.php";

$store_id=$argv[1];
$api_token=$argv[2];
$orderid=$argv[3];
$txnnumber=$argv[4];

## step 1) create transaction array ###
$txnArray=array('type'=>'refund',
         'txn_number'=>$txnnumber,
         'order_id'=>$orderid,
         'amount'=>'1.00',
         'crypt_type'=>'7',
         'dynamic_descriptor'=>'123456'
           );

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

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods

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


?>
```

Independent Refund

The Independent Refund (ind_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 eSELECTplus gateway; however, the credit card number and expiry date will need to be passed. The Independent Refund transaction requires several variables (store_id, api_token, order_id, amount, pan, expdate, and crypt_type). There are also a number of optional fields such as cust_id and dynamic_descriptor available. 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 yesguy unique_order_id

require "../mpgClasses.php";

$store_id='store5';
$api_token='yesguy';
$orderid='ord-'.date("dmy-G:i:s");

## step 1) create transaction array ###
$txnArray=array('type'=>'ind_refund',
         'order_id'=>$orderid,
         'cust_id'=>'my cust id',
         'amount'=>'1.00',
         'pan'=>'4242424242424242',
         'expdate'=>'1103',
         'crypt_type'=>'7',
         'dynamic_descriptor'=>'123456'
           );
## 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 ##
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods

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

?>
```

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 yesguy 66002173

require "../mpgClasses.php";

$store_id='store5';
$api_token='yesguy';
$ecr_number='66002163';

## 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 ##
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);

## step 4) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

##step 5) get array of all credit cards
$creditCards = $mpgResponse->getCreditCards($ecr_number);

## step 6) loop through the array of credit cards and get information

for($i=0; $i < count($creditCards); $i++)
 {
  print "\nCard Type = $creditCards[$i]<br>";

  print "\nPurchase Count = "
        . $mpgResponse->getPurchaseCount($ecr_number,$creditCards[$i])."<br>";

  print "\nPurchase Amount = "
        . $mpgResponse->getPurchaseAmount($ecr_number,$creditCards[$i])."<br>";


  print "\nRefund Count = "
        . $mpgResponse->getRefundCount($ecr_number,$creditCards[$i])."<br>";


  print "\nRefund Amount = "
        . $mpgResponse->getRefundAmount($ecr_number,$creditCards[$i])."<br>";

  print "\nCorrection Count = "
        . $mpgResponse->getCorrectionCount($ecr_number,$creditCards[$i])."<br>";

  print "\nCorrection Amount = "
        . $mpgResponse->getCorrectionAmount($ecr_number,$creditCards[$i])."<br>";

 }
?>
```

Open Totals

Open Totals allows the merchant to retrieve details about all Credit Card transactions within 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.

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. ecr number
##
## Example php -q TestOpenTotals.php store1 yesguy 66002163

require "../mpgClasses.php";

$store_id='store5';
$api_token='yesguy';
$ecr_number='66002163';

## step 1) create transaction array ###
$txnArray=array('type'=>'opentotals',
               'ecr_number'=>$ecr_number
               );

$mpgTxn = new mpgTransaction($txnArray);

## step 2) create mpgRequest object ###
$mpgReq=new mpgRequest($mpgTxn);

## step 3) create mpgHttpsPost object which does an https post ##
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);

## step 4) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

##step 5) get array of all credit cards
$creditCards = $mpgResponse->getCreditCards($ecr_number);

## step 6) loop through the array of credit cards and get information

for($i=0; $i < count($creditCards); $i++)
 {
  print "\nCard Type = $creditCards[$i]";

  print "\nPurchase Count = "
        . $mpgResponse->getPurchaseCount($ecr_number,$creditCards[$i]);

  print "\nPurchase Amount = "
        . $mpgResponse->getPurchaseAmount($ecr_number,$creditCards[$i]);

  print "\nRefund Count = "
        . $mpgResponse->getRefundCount($ecr_number,$creditCards[$i]);

  print "\nRefund Amount = "
        . $mpgResponse->getRefundAmount($ecr_number,$creditCards[$i]);

  print "\nCorrection Count = "
        . $mpgResponse->getCorrectionCount($ecr_number,$creditCards[$i]);

  print "\nCorrection Amount = "
        . $mpgResponse->getCorrectionAmount($ecr_number,$creditCards[$i]);
 }
?>
```

Card Verification

The Card Verification (CardVerification) transaction is available to check the validity of a credit card, expiry date and any additional details, such as the Card Verification Digits or Address Verification details. It does not verify the available amount or lock any funds on the credit card. The CardVerification transaction requires several variables (store_id, api_token, order_id, pan, expiry_date). Also, Address Verification (AVS) and Card Verification Digits (CVD) are optional. This transaction type will not place a charge on the credit card. Please refer to Appendix A. Definition of Request Fields for variable definitions.


NOTE Card Verification transaction is only applicable to Visa and MasterCard

```
<?php

require "../mpgClasses.php";

$store_id="store5";
$api_token="yesguy";


## step 1) create transaction hash ###
$txnArray=array('type'=>'card_verification',
         'order_id'=>'ord-'.date("dmy-G:i:s"),
         'cust_id'=>'my cust id',
         'pan'=>'4242424242424242',
         'expdate'=>'1212',
         'crypt_type'=>'7'
           );

## step 2) create a transaction  object passing the hash created in
$mpgTxn = new mpgTransaction($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 ##
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

## step 5) get an mpgResponse object ##
$mpgResponse=$mpgHttpPost->getMpgResponse();

## step 6) retrieve data using get methods

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

?>
```

6. Transaction with Extra features - Examples

In the previous section the instructions were provided for the basic transaction set. eSELECTplus also provides several extra features/functionalities. These features include storing customer and order details, Verified by Visa and sending transactions to the Recurring Billing feature. Verified by Visa and Recurring Billing must be added to your account, please call the Service Centre at 1 866 319 7450 to have your profile updated.

Purchase (with Customer and Order details)

Below is an example of sending a 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 PreAuth with the exception of transaction type which changes from 'purchase' to 'preauth'. Customer details can only be sent with Purchase and PreAuth. It can be used in conjunction with other extra features such as VBV and Recurring Billing. ***Please note that the mpgCustInfo fields are not used for any type of address verification or fraud check.***

```
<?php

## Example php -q TestPurchase-CustInfo.php

require "../mpgClasses.php";

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

$store_id='store5';
$api_token='yesguy';

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

$type='purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='my cust id';
$amount='1.00';
$pan='4242424242424242';
$expiry_date='0812';          //December 2008
$crypt='7';

/******************* Customer Information Variables *******************/

$first_name = 'Cedric';
$last_name = 'Benson';
$company_name = 'Chicago Bears';
$address = '334 Michigan Ave';
$city = 'Chicago';
$province = 'Illinois';
$postal_code = 'M1M1M1';
$country = 'United States';
$phone_number = '453-989-9876';
$fax = '453-989-9877';
$tax1 = '1.01';
$tax2 = '1.02';
$tax3 = '1.03';
$shipping_cost = '9.95';
$email ='Joe@widgets.com';
$instructions ="Make it fast";

/********************** Line Item Variables ***************************/

$item_name[0] = 'Guy Lafleur Retro Jersey';
$item_quantity[0] = '1';
$item_product_code[0] = 'JRSCDA344';
$item_extended_amount[0] = '129.99';

$item_name[1] = 'Patrick Roy Signed Koho Stick';
$item_quantity[1] = '1';
$item_product_code[1] = 'JPREEA344';
$item_extended_amount[1] = '59.99';
```

```
/******************** Customer Information Object ********************/

$mpgCustInfo = new mpgCustInfo();

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

$billing = array(
                              'first_name' => $first_name,
                 'last_name' => $last_name,
                 'company_name' => $company_name,
                 'address' => $address,
                 'city' => $city,
                 'province' => $province,
                 'postal_code' => $postal_code,
                 'country' => $country,
                 'phone_number' => $phone_number,
                 'fax' => $fax,
                 'tax1' => $tax1,
                 'tax2' => $tax2,
                 'tax3' => $tax3,
                 'shipping_cost' => $shipping_cost
                );

$mpgCustInfo->setBilling($billing);

$shipping = array(
                              'first_name' => $first_name,
                 'last_name' => $last_name,
                 'company_name' => $company_name,
                 'address' => $address,
                 'city' => $city,
                 'province' => $province,
                 'postal_code' => $postal_code,
                 'country' => $country,
                 'phone_number' => $phone_number,
                 'fax' => $fax,
                 'tax1' => $tax1,
                 'tax2' => $tax2,
                 'tax3' => $tax3,
                 'shipping_cost' => $shipping_cost
                );

$mpgCustInfo->setShipping($shipping);

$mpgCustInfo->setEmail($email);
$mpgCustInfo->setInstructions($instructions);

/********************** Set Line Item Information **********************/

$item[0] = array(
                         'name'=>$item_name[0],
               'quantity'=>$item_quantity[0],
               'product_code'=>$item_product_code[0],
               'extended_amount'=>$item_extended_amount[0]
               );

$item[1] = array(
                         'name'=>$item_name[1],
               'quantity'=>$item_quantity[1],
               'product_code'=>$item_product_code[1],
               'extended_amount'=>$item_extended_amount[1]
               );

$mpgCustInfo->setItems($item[0]);
$mpgCustInfo->setItems($item[1]);

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

$txnArray=array(
                             'type'=>$type,
                       'order_id'=>$order_id,
                       'cust_id'=>$cust_id,
                       'amount'=>$amount,
```

```
                    'pan'=>$pan,
                    'expdate'=>$expiry_date,
                    'crypt_type'=>$crypt
                );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgTxn->setCustInfo($mpgCustInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

/***************8********** Response *********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
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());

?>
```

CavvPurchase (Purchase with Verified by Visa – VBV or MasterCard SecureCode - MCSC)

Below is an example of sending a Purchase with the Verified by Visa extra fields. The 'cavv' is obtained by using either the Moneris MPI or a third party MPI. The format outlined below is identical for a PreAuth with the exception of the TransType which changes from 'cavv_purchase' to 'cavv_preauth'. VBV must be added to your account, please call the Service Centre at 1-866-319-7450 to have your profile updated. The optional customer and order details can be included in the transaction using steps 1, 2 and 5 from the method above - *Purchase (with Customer and Order Details).*

```
<?php

## Example php -q TestPurchase-VBV.php "moneris" store

require "../mpgClasses.php";

$store_id='store5';
$api_token='yesguy';

$type='cavv_purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='CUST887763';
$amount='10.00';
$pan="4242424242424242";
$expiry_date="0812";
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';

$txnArray=array('type'=>$type,
                'order_id'=>$order_id,
                 'cust_id'=>$cust_id,
                  'amount'=>$amount,
                  'pan'=>$pan,
                  'expdate'=>$expiry_date,
                 'cavv'=>$cavv,
                 'dynamic_descriptor'=>'123456'
                );

$mpgTxn = new mpgTransaction($txnArray);

$mpgRequest = new mpgRequest($mpgTxn);

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

Purchase (with Recurring Billing)

Recurring Billing is a feature that allows the transaction information to be sent once and then re-billed on a specified interval for a certain number of times. This is a feature commonly used for memberships, subscriptions, or any other charge that is re-billed on a regular basis. The transaction is split into two parts; the recur information and the transaction information. Please see Appendix D. Recur Fields for description of each of the fields. The optional customer and order details can be included in the transaction using steps 1, 2 and 5 from the method above *-Purchase (with Customer and Order Details).* Recurring Billing must be added to your account, please call the Service Centre at 1 866 319 7450 to have your profile updated.

```
<?php

##
## Example php -q TestPurchase-Recur.php store3 yesguy unique_order_id
##

require "../mpgClasses.php";

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

$store_id = 'store5';
$api_token = 'yesguy';

/******************************** Recur Variables *************************/
$recurUnit = 'eom';
$startDate = '2008/11/30';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';

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

$orderId = 'ord-'.date("dmy-G:i:s");
$custId = 'student_number';
$creditCard = '5454545454545454';
$nowAmount = '10.00';
$expiryDate = '0912';
$cryptType = '7';

/********************** Recur Associative Array *********************/

$recurArray = array('recur_unit'=>$recurUnit, // (day | week | month)
                                    'start_date'=>$startDate, //yyyy/mm/dd
                                    'num_recurs'=>$numRecurs,
                                    'start_now'=>$startNow,
                                    'period' => $recurInterval,
                                    'recur_amount'=> $recurAmount
                                    );

$mpgRecur = new mpgRecur($recurArray);

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

$txnArray=array('type'=>'purchase',
                               'order_id'=>$orderId,
                               'cust_id'=>$custId,
                               'amount'=>$nowAmount,
                               'pan'=>$creditCard,
                               'expdate'=>$expiryDate,
                               'crypt_type'=>$cryptType
                               );

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

$mpgTxn = new mpgTransaction($txnArray);

/***************************** Recur Object *********************************/

$mpgTxn->setRecur($mpgRecur);
```

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

As part of the Recurring Billing response there will be an additional method called getRecurSuccess( ). This can return a value of ‘true’ or ‘false’ based on whether the recurring transaction was successfully registered in our database.

Recur Update

Recur Update allows a user to alter characteristics of a previously registered Recurring Billing transaction. This feature is commonly used to update customer's credit card information and the number of recurs to the account. Please see Appendix A. Definition of Request Fields and Appendix D. Recur Fields for description of each of the fields.
Recurring Billing must be added to your account, please call the Service Centre at 1 866-319-7450 to have your profile updated.

```
<?php

## Example php -q TestRecurUpdate.php store1
require "../mpgClasses.php";

/************************** Request Variables *****************************/
$store_id='store5';
$api_token='yesguy';

/************************ Transactional Variables ************************/
$type='recur_update';
$cust_id='my cust id';
$order_id='test310707';
$recur_amount='1.00';
$pan='4242424242424242';
$expiry_date='1111';
$add_num='20';
$total_num='999';
$hold = 'false';
$terminate = 'false';

/********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
                'order_id'=>$order_id,
                'cust_id'=>$cust_id,
                'recur_amount'=>$recur_amount,
                'pan'=>$pan,
                'expdate'=>$expiry_date,
                'add_num_recurs' => $add_num,
                'total_num_recurs' => $total_num,
                'hold' => $hold,
                'terminate' => $terminate
                );
/************************** Transaction Object ****************************/
$mpgTxn = new mpgTransaction($txnArray);

/****************************** Request Object ****************************/
$mpgRequest = new mpgRequest($mpgTxn);

/***************************** HTTPS Post Object *****************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

/****************************** Response **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nRecurUpdateSuccess = " . $mpgResponse->getRecurUpdateSuccess());
print("\nNextRecurDate = " . $mpgResponse->getNextRecurDate());
print("\nRecurEndDate = " . $mpgResponse->getRecurEndDate());
?>
```

As part of the Recurring Billing update response there will be an additional method called getRecurUpdateSuccess(). This can return a value of 'true' or 'false' based on whether the recurring transaction was successfully updated in our database

Purchase with CVD and AVS (eFraud)

Below is an example of a Purchase transaction with CVD and AVS information. These values can be sent in conjunction with other additional variables such as Recurring Billing or customer information. With this feature enabled in your merchant profile, you will be able to pass in these fields for the following transactions 'purchase', 'preauth', 'cavv_purchase', and 'cavv_preauth'. To have the eFraud feature added to your profile, please call the Service Center at 1-866-319-7450 to have your profile updated.

When testing eFraud (AVS and CVD) you **must only use** the Visa test card numbers, 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at https://developer.moneris.com.

NOTE The CVD Value supplied by the cardholder should simply be passed to the eSelectPlus payment gateway. Under no circumstances should it be stored for subsequent uses or displayed as part of the receipt information.

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestPurchase-Efraud.php store1 45728773 45109
##
require "../mpgClasses.php";

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

$store_id='store5';
$api_token='yesguy';

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

$type='purchase';
$order_id='ord-'.date("dmy-G:i:s");
$cust_id='my cust id';
$amount='10.30';
$pan='4242424242424242';
$expiry_date='0812';          //December 2008
$crypt='7';

/************************** AVS Variables *****************************/

$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';
$avs_email = 'test@host.com';
$avs_hostname = 'www.testhost.com';
$avs_browser = 'Mozilla';
$avs_shiptocountry = 'Canada';
$avs_merchprodsku = '123456';
$avs_custip = '192.168.0.1';
$avs_custphone = '5556667777';

/************************** CVD Variables *****************************/

$cvd_indicator = '1';
$cvd_value = '198';

/********************** AVS Associative Array *************************/

$avsTemplate = array('avs_street_number'=>$avs_street_number,
                     'avs_street_name' =>$avs_street_name,
                     'avs_zipcode' => $avs_zipcode,
                     'avs_hostname'=>$avs_hostname,
                                     'avs_browser' =>$avs_browser,
                                     'avs_shiptocountry' => $avs_shiptocountry,
```

```
                                        'avs_merchprodsku' => $avs_merchprodsku,
                                        'avs_custip'=>$avs_custip,
                                        'avs_custphone' => $avs_custphone
                    );

/********************** CVD Associative Array ************************/

$cvdTemplate = array('cvd_indicator' => $cvd_indicator,
                     'cvd_value' => $cvd_value
                    );

/************************** AVS Object ******************************/

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

/************************** CVD Object ******************************/

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);

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

$txnArray=array(       'type'=>$type,
                       'order_id'=>$order_id,
                       'cust_id'=>$cust_id,
                       'amount'=>$amount,
                       'pan'=>$pan,
                       'expdate'=>$expiry_date,
                       'crypt_type'=>$crypt
                       );
/********************** Transaction Object ****************************/

$mpgTxn = new mpgTransaction($txnArray);

/************************ Set AVS and CVD *****************************/

$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCvdInfo($mpgCvdInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nCVDResponse = " . $mpgResponse->getCvdResultCode());
print("\nITDResponse = " . $mpgResponse->getITDResponse());

?>
```

Purchase with Status Check

The Status Check (SC) flag is set to true or false at the request level as opposed to being at the transaction level. You should send the same parameter values for the transaction level fields in the SC request, i.e. if you send a Completion with SC, include the same values as the original Completion such as the Order ID, amount, Txn number, etc. What you will get back is a SC and Status Message in the Receipt as shown in the sample. A Status Code (SC) of 0-49 indicates successful and 50-999 is not successful. The Status Message will be Found (SC of 0-49) or Not Found or null (SC of 50-999). When it is found, the other Response parameter in the Receipt values will be those of the original transaction. When it is not found, they will be null

Below is an example of a Purchase transaction with Status Check. The same parameter values for the original transaction should be sent in the Status Check request, i.e. if you send a purchase with Status Check, include the same values as the original Purchase such as the store_id, api_token, order_id, amount, pan, expdate, crypt_type and status. Please refer to Appendix A. Definition of Request Fields for variable definitions.

With this feature enabled in your merchant profile, you will be able to pass in the Status Check flag for the following transactions: 'Purchase', 'Refund', 'IndependentRefund', 'PreAuth', 'Completion', 'PurchaseCorrection'. To have the Status Check feature added to your profile, please call the Service Centre at 1-866-319-7450 to have your profile updated.

NOTE

The Status Check request should only be used once and immediately (within 2 minutes) after the last transaction that had failed.

The Status Check request should not be used to check openTotals & batchClose requests.

Do not resend the Status Check request if it has timed out as additional investigation is required.

```
<?php

##
## Example php -q TestPurchase.php store1
##

require "../mpgClasses.php";

/************************** Request Variables *****************************/
$store_id='store5';
$api_token='yesguy';

/************************ Transactional Variables ***************************/
$type='purchase';
$cust_id='cust id';
$order_id='ord-'.date("dmy-G:i:s");
$amount='1.00';
$pan='4242424242424242';
$expiry_date='1111';
$crypt='7';
$status_check = 'false';

/********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
                'order_id'=>$order_id,
                'cust_id'=>$cust_id,
                'amount'=>$amount,
                'pan'=>$pan,
                'expdate'=>$expiry_date,
                'crypt_type'=>$crypt,
                );

/**************************** Transaction Object *****************************/
$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);
/***************************** HTTPS Post Object ****************************/
```

```
/* Status Check Example
$mpgHttpPost  =new mpgHttpsPostStatus($store_id,$api_token,$status_check,$mpgRequest);
*/

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

As part of the Status Check response there will be two additional methods called getStatusCode( ) and getStatusMessage(). Please refer to Appendix B. Definitions of Response Fields.

7. 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 https://developer.moneris.com.

8. 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
moneris **	hurgle	demouser	password

* "store5" is for testing eFraud (AVS & CVD)
** "moneris" is for testing VBV

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
JCB	3566007770015365
Diners	36462462742008

Test Card Numbers for Card Verification	
Card Plan	**Card Number**
Visa	4761739012345678 4761739012345686 4761739012345694 4761739012345603 4761739012345611 4761739012345629 4761739012345637 4761739012345645

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 https://developer.moneris.com.

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

9. 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 H. Sample Receipt.

1. Amount
2. Transaction Type
3. Date and Time
4. Auth Code
5. Response Code
6. ISO Code
7. Response Message
8. Reference Number
9. Goods and Services Order
10. Merchant Name
11. Merchant URL
12. Cardholder Name
13. Return Policy (only a requirement for e-commerce transactions)

10. 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.

11. 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 mpgClasses.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', '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', '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 https://developer.moneris.com.

12. 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)
Email: eselectplus@moneris.com

For Integration Support:
Phone: 1-866-562-4354
Email: eproducts@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.**

13. 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. The last 10 characters of the order_id will be displayed in the “Invoice Number” field on the Merchant Direct Reports. Only the following character sets will be sent to Merchant Direct (A_Z, a-z, space, 0-9). A minimum of 3 and a maximum of 10 characters will be sent to Merchant Direct. If the order_id has less than 3 characters, it may display a blank or 0000000000 in the Invoice Number field. Only the last characters up to the invalid character will be sent. E.G. 1234-567890, 567890 will be sent to Merchant Direct.
orig_order_id	50 / an	Merchant defined transaction identifier – used in the ReAuth transaction to refer to the original PreAuth that has been partially captured.
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	1 / an	E-Commerce Indicator: 1 - Mail Order / Telephone Order - Single 2 - Mail Order / Telephone Order - Recurring 3 - Mail Order / Telephone Order - Instalment 4 - Mail Order / Telephone Order - Unknown Classification 5 - Authenticated E-commerce Transaction (VBV) 6 – Non Authenticated E-commerce Transaction (VBV) 7 - SSL enabled merchant 8 - Non Secure Transaction (Web or Email Based) 9 - SET non - Authenticated transaction
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.
dynamic_descriptor	20/an	Merchant defined description sent on a per-transaction basis that will appear on the credit card statement. Dependent on the card Issuer, the statement will typically show the dynamic desciptor appended to the merchant's existing business name separated by the "/" character. Please note that the combined length of the merchant's business name, forward slash "/" character, and the dynamic descriptor may not exceed 22 characters. -Example- Existing Business Name: ABC Painting Dynamic Descriptor: Booking 12345 Cardholder Statement Displays: ABC Painting/Booking 1

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.
avs_street_number avs_street_name	19 / an	Street Number & Street Name (max – 19 digit limit for street number and street name combined). This must match the address that the issuing bank has on file.
avs_zipcode	10 / an	Zip or Postal Code – This must match what the issuing banks has on file.
cvd_value	4 / num	Credit Card CVD value – this number accommodates either 3 or 4 digit CVD values. Note: The CVD value supplied by the cardholder should simply be passed to the eSELECTplus payment gateway. Under no circumstances should it be stored for subsequent uses or displayed as part of the receipt information.
cvd_indicator	1 / num	CVD presence indicator (1 digit – refer to section 18 for values). Typically the value is 1.
status_check	true/false	Once set to “true”, the gateway will check the status of a transaction that has an order_id that matches the one passed. • If the transaction is found, the gateway will respond with the specifics of that transaction. • If the transaction is not found, the gateway will respond with a not found message. Once it is set to “false”, the transaction will process as a new transaction. The Status Check flag can be passed with the following transactions: ‘Purchase’, ‘Refund’, ‘IndependentRefund’, ‘PreAuth’, ‘Completion’, ‘PurchaseCorrection’.

NOTE

The alphanumeric fields will allow the following characters: **a-z A-Z 0-9 _ - : . @ spaces**

All other request fields allow the following characters: **a-z A-Z 0-9 _ - : . @ $ = /**

14. Appendix B. Definitions of Response Fields

Response Fields		
Variable Name	**Size/Type**	**Description**
ReceiptId	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 https://developer.moneris.com.
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.
AvsResultCode	1/alpha	Indicates the address verification result. Refer to section 18
CvdResultCode	2/an	Indicates the CVD validation result. Refer to section 18
CavvResultCode	1 / an	The CAVV result code indicates the result of the CAVV validation. 0 = CAVV authentication results invalid 1 = CAVV failed validation; authentication 2 = CAVV passed validation; authentication 3 = CAVV passed validation; attempt 4 = CAVV failed validation; attempt 7 = CAVV failed validation; attempt (US issued cards only) 8 = CAVV passed validation; attempt (US issued cards only) 9 = CAVV failed validation; attempt (US issued cards only)

		A = CAVV passed validation; attempt (US issued cards only) B = CAVV passed validation but downgraded; treat this transaction same as ECI 7 Please refer to section 19 for a description for each response.
StatusCode	3/an	The StatusCode is populated when status_check is set to “true” in the request. <50: Transaction found >=50: Transaction not found
StatusMsg	found/ not found	The StatusMsg is populated when status_check is set to “true” in the request.
IsVisaDebit	true/false/ null	Indicates whether the card that the transaction was performed on is Visa debit. true = Card is Visa Debit false = Card is not Visa Debit null = there was an error in identifying the card

15. Appendix C. CustInfo Fields

Field Definitions		
Field Name	**Size/Type**	**Description**
Billing and Shipping Information		
NOTE: The fields for billing and shipping information are identical. Please refer to section 6 - Purchase (with Customer and Order details) for an example.		
first_name	30 / an	
last_name	30 / an	
company_name	50 / an	
address	70 / an	
city	30 / an	
province	30 / an	
postal_code	30 / an	
country	30 / an	
phone	30 / an	
fax	30 / an	
tax1	10 / an	
tax2	10 / an	
tax3	10 / an	
shipping_cost	10 / an	
Item Information		
NOTE: You may send multiple items. Please refer to section 6 - Purchase (with Customer and Order details) for an example.		
item_name	45 / an	
item_quantity	5 / num	You must send a quantity > 0 or the item will not be added to the item list (i.e. minimum 1, maximum 99999)
item_product_code	20 / an	
item_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	60 / an	
instructions	100 / an	

If you send characters that are not included in the allowed list, these extra transaction details may not be stored.

NOTE

All fields are alphanumeric and allow the following characters: **a-z A-Z 0-9 _ - : . @ $ = /**
All French accents should be encoded as HTML entities, e.g. &eacute

Also, the data sent in Billing and Shipping Address fields will not be used for any address verification. Please refer to section 6 - Purchase (with Customer and Order details) for an example.

16. Appendix D. Recur Fields

Recur Request Fields		
Variable Name	**Size/Type**	**Description**
recur_unit	day, week, month	The unit that you wish to use as a basis for the Interval. This can be set as day, week or month. Then using the "period" field you can configure how many days, weeks, months between billing cycles.
period	0 – 999 / num	This is the number of recur_units you wish to pass between billing cycles. Example : period = 3, recur_unit=month -> Card will be billed every 3 months. period = 4, recur_unit=weeks -> Card will be billed every 4 weeks. period = 45, recur_unit=day -> Card will be billed every 45 days. Please note that the total duration of the recurring billing transaction should not exceed 5-10 years in the future.
start_date	YYYY/MM/DD	This is the date on which the first charge will be billed. The value must be in the future. It cannot be the day on which the transaction is being sent. If the transaction is to be billed immediately the start_now feature must be set to true and the start_date should be set at the desired interval after today.
start_now	true / false	When a charge is to be made against the card immediately start_now should be set to 'true'. If the billing is to start in the future then this value is to be set to 'false'. When start_now is set to 'true' the amount to be billed immediately may differ from the recur amount billed on a regular basis thereafter.
recur_amount	9 / decimal	Amount of the recurring transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99. This is the amount that will be billed on the start_date and every interval thereafter.
num_recurs	1 – 99 / num	The number of times to recur the transaction.
amount	9 / decimal	When start_now is set to 'true' the amount field in the transaction array becomes the amount to be billed immediately. When start_now is set to 'false' the amount field in the transaction array should be the same as the recur_amount field.

Recur Request Examples	
Recur Request Exampl	**Description**

`$recurArray = array('recur_unit'=>'month',`
`'start_date'=>'2007/01/02',`
`'num_recurs'=>'12',`
`'start_now'=>'false',`
`'period' => '2',`
`'recur_amount'=> '30.00'`
`);`
 	
`$mpgRecur = new mpgRecur($recurArray);`
`// ------ step 3) create transaction array ###`
`$txnArray=array('type'=>'purchase',`
`'order_id'=>'monthly_bill',`
`'cust_id'=>'mem-1234-01',`
`'amount'=>'30.00',`
`'pan'=>'5454545454545454',`
`'expdate'=>'0712',`
`'crypt_type'=>'2'`
`);`
	In the example to the left the first transaction will occur in the future on Jan 2nd 2007. It will be billed $30.00 every 2 months on the 2nd of each month. The card will be billed a total of 12 times.

```
$recurArray = array('recur_unit'=>'week',
        'start_date'=>'2007/01/02',
        'num_recurs'=>'26',
        'start_now'=>'true',
        'period' => '2',
        'recur_amoun't=> '30.00'
        );

$mpgRecur = new mpgRecur($recurArray);

// ------ step 3) create transaction array ###
$txnArray=array('type'=>'purchase',
        'order_id'=>'biweekly_bill',
        'cust_id'=>'mem-1234-02',
        'amount'=>'15.00',
        'pan'=>'5454545454545454',
        'expdate'=>'0712',
        'crypt_type'=>'2'
        );
```

In the example on the left the first charge will be billed immediately. The initial charge will be for $15.00. Then starting on Jan 2nd 2007 the credit card will be billed $30.00 every 2 weeks for 26 recurring charges. The card will be billed a total of 27 times. (1 x $15.00 (immediate) and 26 x $30.00 (recurring))

NOTE

When completing the recurring billing portion please keep in mind that to prevent the shifting of recur bill dates, avoid setting the start_date for anything past the 28th of any given month when using the recur_unit set to "month". For example, all billing dates set for the 31st of May will shift and bill on the 30th in June and will then bill the cardholder on the 30th for every subsequent month. To set the billing dates for the end of the month please set the recur_unit to "eom".

Recur Update Request Fields		
Variable Name	**Size/Type**	**Description**
cust_id	50 / an	This updates the current cust_id associated with the recurring transaction and will be submitted with all future recurring purchases.
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. This will be the new credit card number charged with all future recurs. This field pertains only to credit card transactions.
expiry_date	YYMM / num	Expiry Date - format YYMM no spaces or slashes, replaces the current expiry date in the payment details and must be today's date or later. PLEASE NOTE THAT THIS IS REVERSED FROM THE DATE DISPLAYED ON THE PHYSICAL CARD WHICH IS MMYY
avs_street_number avs_street_name	19 / an	Street Number & Street Name (max – 19 digit limit for street number and street name combined). This must match the address that the issuing bank has on file. The updated AVS details will be submitted for all future credit card recurs. Please note; the store must have the AVS feature enabled.
avs_zipcode	9 / an	Zip or Postal Code – This must match what the issuing banks has on file.
recur_amount	9 / decimal	Amount of all future recurring transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99.
add_num	1-999 / num	This is the number of recurring transactions to be added to the current total number of recurs on file. Example: num_recurs* = 5, add_num = 2, New total number of recurs = 7

		*the “num_recurs” initially sent in while registering the recurring transaction. Please refer to Recur Request Fields table for variable definition.
total_num	1-999 / num	This is an update to replace the current total number of recurs on file. Example: num_recurs* = 5, total_num = 2, New total number of recurs = 2 *the “num_recurs” initially sent in while registering the recurring transaction. Please refer to Recur Request Fields table for variable definition.
hold	true / false	A transaction can be put 'On Hold' at any time. While a transaction is 'On Hold' it will not be billed when the time comes for it to recur, but the number of recurs will be decremented.
terminate	true / false	A Recurring Billing transaction can be Terminated at any time. A terminated Recurring transaction can no longer be reactivated.

Recur Update Response codes:

The Recur Update response is a 3 digit numeric value. The following is a list of all possible responses once a Recur Update transaction has been sent thru.

Recur Update RESPONSE CODES	
RESULT VALUE	**DEFINITION**
001	Recurring transaction successfully updated (optional: terminated)
983	Can not find the previous transaction
984	Data error: (optional: field name)
985	Invalid number of recurs
986	Incomplete: timed out
null	Error: Malformed XML

NOTE When completing the update recurring billing portion please keep in mind that the recur bill dates cannot be changed to have an end date greater than 10 years from today and cannot be changed to have an end date end today or earlier.

17. Appendix E. 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

18. Appendix F. Card Validation Digits (CVD) & Address Verification Service (AVS)

Card Validation Digits (CVD)

The Card Validation Digits (CVD) value refers to the numbers appearing on the back of the credit card which are not imprinted on the front. The exception to this is with American Express cards where this value is indeed printed on the front

Address Verification Service (AVS)

The Address Verification Service (AVS) value refers to the cardholder's street number, street name and zip/postal code as it would appear on their statement.

Additional Information for CVD and AVS

The responses that are received from CVD and AVS verifications are intended to provide added security and fraud prevention, but the response itself will not affect the issuer's approval of a transaction. Upon receiving a response, the choice to proceed with a transaction is left entirely to the merchant.

Please note that all responses coming back from these verification methods are not direct indicators of whether a merchant should complete any particular transaction. The responses should not be used as a strict guideline of which transaction will approve or decline.

NOTE

Please note that CVD verification is only applicable towards Visa, MasterCard and American Express transactions.

Also, please note that AVS verification is only applicable towards Visa, MasterCard, Discover and American Express transactions. This verification method is not applicable towards any other card type.

***For additional information on how to handle these responses, please refer to the eFraud (CVD & AVS) Result Codes document which is available at https://developer.moneris.com.**

19. Appendix G. CAVV Result Code

The Cardholder Authentication Verification Value (CAVV) is a value that allows VisaNet to validate the integrity of the VbV transaction data. These values are passed back from the issuer to the merchant after the VbV/SecureCode authentication has taken place. The merchant then integrates the CAVV value into the authorization request using the 'cavv_purchase' or 'cavv_preauth' transaction type.
For more information on sending VBV/SecureCode transactions, please refer to our "Moneris MPI - Verified By Visa / MasterCard SecureCode ColdFusion API" document.

The following table describes the contents of the CAVV data response and what it means to the merchant.

Table of CAVV result codes		
Result Code	**Message**	**What this means to you as a merchant…**
0	CAVV authentication results invalid.	For this transaction you may not receive protection from chargebacks as a result of using VBV as the CAVV was considered invalid at the time the financial transaction was processed. Please check that you are following the VBV process correctly and passing the correct data in our transactions.
1	CAVV failed validation; authentication	Provided that you have implemented the VBV process correctly the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
2	CAVV passed validation; authentication	The CAVV was confirmed as part of the financial transaction. This transaction is a fully authenticated VBV transaction (ECI 5)
3	CAVV passed validation; attempt	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
4	CAVV failed validation; attempt	Provided that you have implemented the VBV process correctly the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
7	CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VBV process correctly and passing the correct data in our transactions. Provided that you have implemented the VBV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
8	CAVV passed validation; attempt (US issued cards only	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
9	CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VBV process correctly and passing the correct data in our transactions. Provided that you have implemented the VBV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
A	CAVV passed validation; attempt (US issued cards only)	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
B	CAVV passed validation	The CAVV was confirmed as part of the financial transaction. However, this transaction does qualify for the liability shift. Treat this transaction the same as an ECI 7.

20. Appendix H. 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:	12.00	APPROVED * =	

Item	Description	Qty	Amount	Subtotal
cir-001	Med Circle	1	2.00	2.00
tri-002	Big triangle	1	1.00	1.00
squ-003	small square	2	1.00	3.00
			Shipping:	4.00
			GST :	1.00
			PST :	1.00
			Total:	12.00 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

Trademarks

Printed in Canada.

10 9 8 7 6 5 4 3 2 1