Merchant Integration Guide

PHP API – v 1.2.4

Revision Number	Date	Description
V1.1.0	January 22, 2007	-Document edited for coherence
V1.1.1	February 4, 2008	-Section 2. System and Skill Requirement -Added PCI note -Section 8. Transaction with Extra features: Purchase with CVD and AVS (eFraud) -Added CVD note. -Appendix A. Definition of Request Fields – Added CVD note. -Appendix F. Card Validation Digits (CVD) – Added CVD note.
V1.1.2	July 15, 2008	-Section 6. Transaction Types and Transaction Flow - added ACH & Pinless Debit transactions to the process flow -Section 7. Basic Transaction Examples – Added Force Post transaction type -Section 9. Mag Swipe Transaction Examples - Added Mag Swipe Force Post transaction type -Section 11. Pinless Debit Transaction Examples – Added Pinless Debit Transaction Examples -Section 12. ACH Transaction Examples – Added ACH Transaction Examples -Section 13. ACH Transaction Examples with Extra Features -Added ACH Debit (with Customer and Order details) - ACH Debit (with Recurring Billing) -Section 14. Administrative Transactions – Added Recur Billing Update (Customer Information) -Section 16. How Do I Test My Solution? -Added new list of test stores -Added test card and bank account details -Appendix A. Definition of Request Fields – Updated pos_code definition -Appendix D. Recur Fields -Added Recur Update Request Fields. -Added Recur Update Response codes. -Appendix E. AchInfo Fields - Added new section -Appendix H. Address Verification Service (AVS) – Merged Visa/MC/Discover Response Code tables -Appendix J. Basic Transaction Receipt (Non Track2) -Added new section with screenshots -Replaced receipt example -Appendix K. Mag Swipe Transaction Receipt (Track2) – Added new section with screenshots -Appendix L. ACH Transaction Confirmation – Added new section with screenshots -Appendix M. Pinless Debit Fields – Added new section
V1.1.3	September 28, 2010	-Section 2. System and Skill Requirement – Added PCI & PA-DSS requirements note. -Section 6. Transaction Types and Transaction Flows – added ACH & Pinless Debit transaction definitions -Section 13. ACH Transaction Examples - Added ACH Debit (Check not present) example notes - Added ACH Debit (Check present) example -Section 18. What Do I Need to Inlcude in the Receipt? -Added ACH (Check Present) -Appendix B. Definitions of Response Fields – Added Recur Update response fields -Appendix D. Recur and Recur Update Fields – Updated Recur Update 'terminate' request variable definition -Appendix F. AchInfo Fields – Added ACH Debit Check Present request fields -Appendix G. ACH Sec Codes and Process Flow – Added Sec Code definitions and decision flow charts -Appendix N. Pinless Debit Transaction Receipt – Added new section with screenshots -Appendix O. ACH Transaction Receipt (Check Not Present) – Updated existing ACH receipt to identify it for Check Not Present only -Appendix P. ACH Transaction Receipt (Check Physically Present) -Added new section with screenshots
V1.2.0	June 2, 2011	-Section 2. System and Skill Requirement – Updated PCI & PA-DSS requirements note. -Section 6. Transaction Types and Transaction Flow

		-Added Process Flow for PreAuth/ReAuth/Capture Transactions -Added Card Verification & ReAuth transaction definitions -Section 7. Basic Transaction Examples -PreAuth (basic) - Added PreAuth reversal note -Added ReAuth example -Capture - Added PreAuth reversal note -Added Dynamic Descriptor to the following examples: -Purchase (basic) -PreAuth (basic) -ReAuth -Independent Refund -Force Post -Added CardLevelResult to the following examples: -Purchase (basic) -PreAuth (basic) -ReAuth -Independent Refund -Force Post -Section 8. Basic Transaction with Exttra Features -Added Dynamic Descriptor to the following examples: -Purchase (with Customer and Order Details) -Purchase (with Verified by Visa / MasterCard SecureCode) -Purchase (with Recurring Billing) -Purchase (with CVD and AVS-eFraud) -Added CardLevelResult to the following examples: -Purchase (with Customer and Order Details) -Purchase (with Recurring Billing) -Purchase (with CVD and AVS-eFraud) -Section 9. Mag Swipe Transaction Examples -Added Dynamic Descriptor to the following examples: -Mag Swipe Purchase -Mag Swipe PreAuth -Mag Swipe Independent Refund -Mag Swipe Force Post -Added CardLevelResult to the following examples: -Mag Swipe Purchase -Mag Swipe PreAuth -Mag Swipe Independent Refund -Mag Swipe Force Post -Section 10. Mag Swipe Transactions with Exttra Features -Mag Swipe Purchase (with AVS) Example -Added Dynamic Descriptor -Added CardLevelResult -Section 15. Administrative Transactions – Added Card Verification example -Section 21. How Do I Get Help? – Updated contact phone numbers and email addresses -Appendix A. Definition of Request Fields -Added orig_order_id -Added dynamic_descriptor -Appendix B. Definitions of Response Fields – Added CardLevelResult -Appendix D. Recur and Recur Update Fields – Added new 'eom' (end of month) recur_unit -Appendix L. Card Level Result Value – Added new section with definition table
V1.2.1	July 18, 2011	-Section 17. How Do I Test My Solution? – Added cURL CA Root Certificate File note
V1.2.2	December 13, 2011	-Appendix I. Card Validation Digits (CVD) – Added American Express response codes -Appendix J. Address Verification Services(AVS) – Added American Express/JCB response codes -Appendix K. Additional Information for CVD and AVS – Added American Express
V1.2.3	August 21, 2012	-New download link updated in various locations: https://developer.moneris.com/ -Section 6. Transaction Types and Transaction Flow – Added Encrypted Mag Swipe Credit Card Transactions -Section 7. Basic Transactions – Added note to Independent Refund

		-Section 8. Transaction with Extra features -Added CavvResultCode to Purchase(with VbV/MasterCard SecureCode) -Added Purchase (with Status Check) -Section 9. Mag Swipe Transactions – Added note to Mag Swipe Independent Refund -Section 11. Encrypted Mag Swipe Transactions – Added new section with examples -Section 12. Encrypted Mag Swipe Transactions with Extra Features – Added new section with examples -Appendix A. Definition of Request Fields -Added status_check request field -Added enc_track2 -Added device_type -Appendix B. Definitions of Response Fields -Added MaskedPan response field -Added CavvResultCode response field -Added StatusCode response field -Added StatusMessage response field -Appendix L. Card Level Result value – Updated table with new values -Appendix M. CAVV Result Code – Added new table with definitions
V1.2.4	November 6, 2012	-Section 6. Transaction Types and Transaction Flow – Added Encrypted Credit Card Transactions -Section 9. Encrypted Transactions – Added new section with examples -Section 10. Encrypted Transactions with Extra Features – Added new section with examples -Section 13. Encrypted Mag Swipe Transactions – Added Encrypted Mag Swipe Forcepost transaction type -Section 19. Administrative Transactions – Added Encrypted Card Verification transaction type

Table of Contents

1. About this Documentation 9
2. System and Skill Requirements 9
3. Verified by Visa 9
4. MasterCard SecureCode 9
5. What is the Process I will need to follow? 10
6. Transaction Types and Transaction Flow 10
   - Process Flow for Basic Credit Card Transactions 11
   - Process Flow for PreAuth / ReAuth / Capture Transactions 12
   - Process Flow for Encrypted Credit Card Transactions 13
   - Process Flow for Mag Swipe Credit Card Transactions 14
   - Process Flow for Encrypted Mag Swipe Credit Card Transactions 15
   - Process Flow for Pinless Debit Transactions 16
   - Process Flow for ACH Transactions 16
7. Basic Transaction Examples 17
   - Purchase (basic) 17
   - PreAuth (basic) 18
   - ReAuth… 19
   - Capture… 20
   - Void…… 21
   - Refund… 22
   - Independent Refund 23
   - Force Post 24
8. Basic Transactions with Extra Features - Examples 25
   - Purchase (with Customer and Order details) 25
   - Purchase (with Verified by Visa / MasterCard SecureCode) 27
   - Purchase (with Recurring Billing) 28
   - Purchase (with CVD and AVS - eFraud) 30
   - Purchase (with Status Check) 32
9. Encrypted Transaction Examples 34
   - Encrypted Purchase 34
   - Encrypted PreAuth 36
   - Encrypted Independent Refund 37
   - Encrypted Force Post 38
10. Encrypted Transactions with Extra Features - Examples 39
    - Encrypted Purchase (with Customer and Order details) 39
    - Encrypted Purchase (with Recurring Billing) 41
    - Encrypted Purchase (with CVD and AVS - eFraud) 43
11. Mag Swipe Transaction Examples 45
    - Mag Swipe Purchase 45
    - Mag Swipe PreAuth 46
    - Mag Swipe Capture 47
    - Mag Swipe Void 48
    - Mag Swipe Refund 49
    - Mag Swipe Independent Refund 50
    - Mag Swipe Forcepost 51
12. Mag Swipe Transactions with Extra Features - Examples 52
    - Mag Swipe Purchase (with Address Verification Service – AVS) 52
13. Encrypted Mag Swipe Transaction Examples 54
    - Encrypted Mag Swipe Purchase 54

Encrypted Mag Swipe PreAuth 55
Encrypted Mag Swipe Independent Refund 56
Encrypted Mag Swipe Forcepost 57

14. Encrypted Mag Swipe Transactions with Extra Features – Examples 58
Encrypted Mag Swipe Purchase (with Address Verification Service – AVS) 58

15. Pinless Debit Transaction Examples 60
Pinless Debit Purchase 60
Pinless Debit Refund 61

16. Pinless Debit Transaction with Extra Features - Examples 62
Pinless Debit Purchase (with Customer and Order details) 62
Pinless Debit Purchase (with Recurring Billing) 64

17. ACH Transaction Examples 65
ACH Debit 65
ACH Debit (Check not present) 65
ACH Debit (Check present) 67
ACH Credit 68
ACH Reversal 69
ACH FI Enquiry 70

18. ACH Transactions with Extra Features – Examples 71
ACH Debit (with Customer and Order details) 71
ACH Debit (with Recurring Billing) 73

19. Administrative Transactions 75
Batch Close 75
Open Totals 76
Card Verification 77
Encrypted Card Verification 78
Recur Update 80
Recur Update – Credit Card example 80
Recur Update – Pinless Debit example 81
Recur Update – ACH example 82

20. What Information will I get as a Response to My Transaction Request? 83

21. How Do I Test My Solution? 83

22. What Do I Need to Include in the Receipt? 85

23. How Do I Activate My Store? 85

24. How Do I Configure My Store For Production? 85

25. How Do I Get Help? 86

26. Appendix A. Definition of Request Fields 87

27. Appendix B. Definitions of Response Fields 89

28. Appendix C. CustInfo Fields 91

29. Appendix D. Recur and Recur Update Fields 92

30. Appendix E. Pinless Debit Fields 95

31. Appendix F. AchInfo Fields 96

32. Appendix G. ACH Sec Codes and Process Flow 97

33. Appendix H. Error Messages 99

34. Appendix I. Card Validation Digits (CVD) 100
35. Appendix J. Address Verification Service (AVS) 101
36. Appendix K. Additional Information for CVD and AVS 102
37. Appendix L. Card Level Result Value 103
38. Appendix M. CAVV Result Code 105
39. Appendix N. Basic Transaction Receipt (Non Track2) 106
40. Appendix O. Mag Swipe Transaction Receipt (Track2) 108
41. Appendix P. Pinless Debit Transaction Receipt 111
42. Appendix Q. ACH Transaction Receipt (Check Not Present) 113
43. Appendix R. ACH Transaction Receipt (Check Physically Present) 115

****** 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.

2. System and Skill Requirements

In order to use the PHP API your system will need to have the following:
1. PHP 4 or later
2. Port 443 open
3. OpenSSL
4. cURL - PHP interface - this can be downloaded from http://curl.haxx.se/download.html

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

Note:

It is important to note that all Merchants and Service Providers that store, process, or transmit cardholder data must comply with PCI DSS and the Card Association Compliance Programs. However, certification requirements vary by business and are contingent upon your "Merchant Level" or "Service Provider Level". Failure to comply with PCI DSS and the Card Association Compliance Programs may result in a Merchant being subject to fines, fees or assessments and/or termination of processing services. Non-compliant solutions may prevent merchants boarding with Moneris Solutions.

As a Moneris Solutions client or partner using this method of integration, your solution must demonstrate compliance to the Payment Card Industry Data Security Standard (PCI DSS) and/or the Payment Application Data Security Standard (PA DSS). These standards are designed to help the cardholders and merchants in such ways as they ensure credit card numbers are encrypted when transmitted/stored in a database and that merchants have strong access control measures.

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

For more information on how to get your application PCI-DSS compliant, please contact our Integration Specialists and visit https://developer.moneris.com to download the PCI-DSS Implementation Guide.

3. Verified by Visa

Verified by Visa (VbV) is a program initiated by Visa. Before approving a transaction eSELECTplus and the Bank that issues the Visa credit cards will attempt to authenticate the cardholder through the use of a password, similar to a debit PIN. When an authentication is attempted the merchant is protected from chargebacks.

If you have enrolled in Verified by Visa (VbV) with Moneris and eSELECTplus, please also refer to the *PHP* VbV / SecureCode MPI document found at: https://developer.moneris.com

4. MasterCard SecureCode

MasterCard SecureCode (MCSC) is a new feature offered by MasterCard. Merchants who have enrolled in this program with Moneris and eSELECTplus will be able to offer their customers added protection against unauthorized credit card use, as well as protect themselves from fraud-related chargebacks. Cardholders that have applied for SecureCode with their issuing bank will be able to use this password similar to a debit PIN number for online transactions with participating online merchants.

Before approving a transaction, eSELECTplus and the Bank that issued the MasterCard will authenticate the cardholder through the use of this password. For merchants who have enrolled in SecureCode, please also refer to the PHP VbV / SecureCode MPI document found at: https://developer.moneris.com

5. 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

6. Transaction Types and Transaction Flow

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

Basic Transactions

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

PreAuth – (authorisation / preauthorisation) The PreAuth verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from a PreAuth so that they may be settled in the merchant's account a Capture must be performed.

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.

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

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.

Independent Refund – (Credit) An Independent Refund can be performed to credit money to a Credit Card. This transaction does not require a prior Purchase or Capture. Please note, the Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an independent refund, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

Force Post (Offline Sale) - The Force Post is used when a merchant obtains the authorization number directly from the issuer using a phone or any third party authorization method. The Force Post retrieves the locked funds and readies them for settlement into the merchant's account.

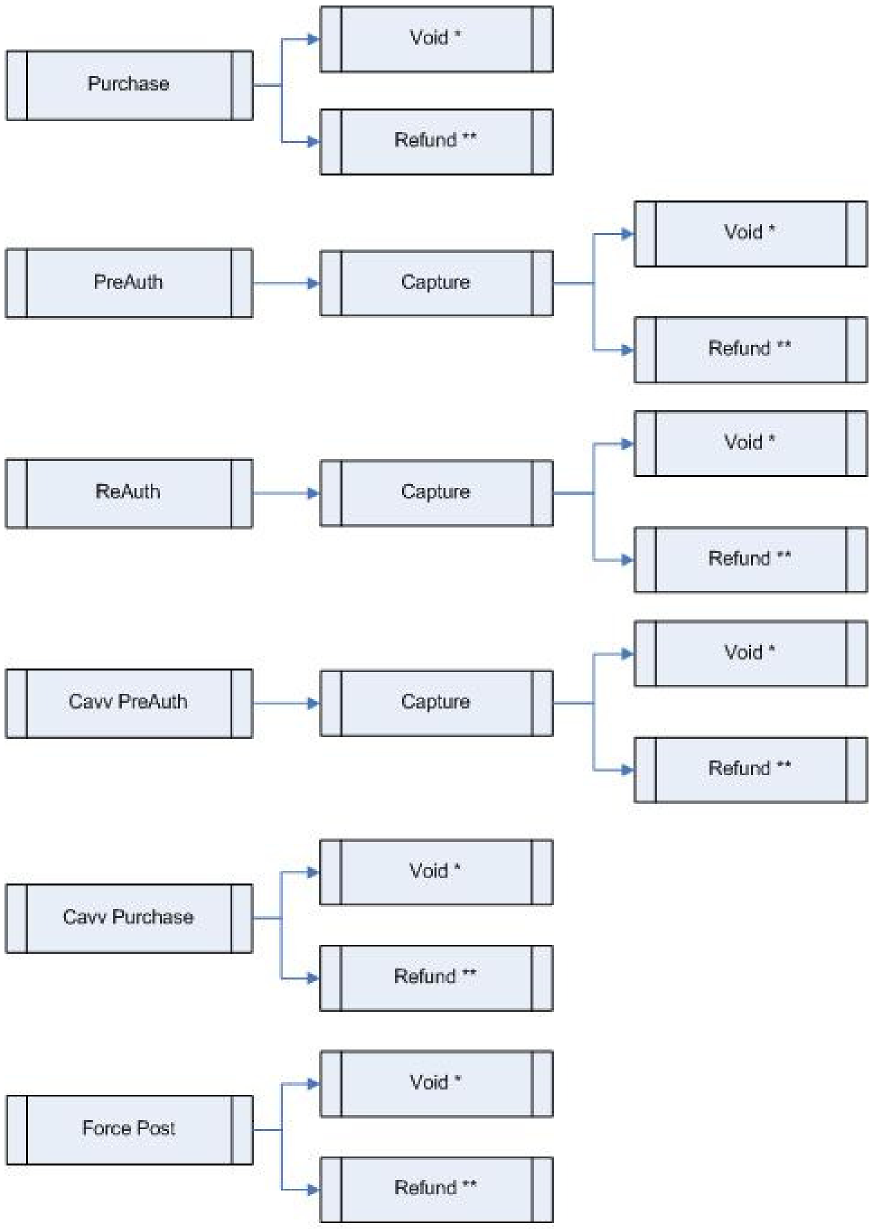
Batch Close – (End of Day / Settlement) When a Batch Close is performed it takes the monies from all Purchase, Capture and Refund transactions so they will be deposited or debited the following business day. For funds to be deposited the following business day the batch must close before 11pm EST.

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.

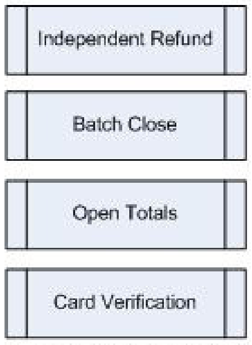
Card Verification – (Account Status Inquiry) Card Verification verifies the validity of the credit card, expiry date and any additional details, such as the Card Verification Digits or Address Verification details. It does not verify the available amount or lock any funds on the credit card.

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

Process Flow for Basic Credit Card Transactions

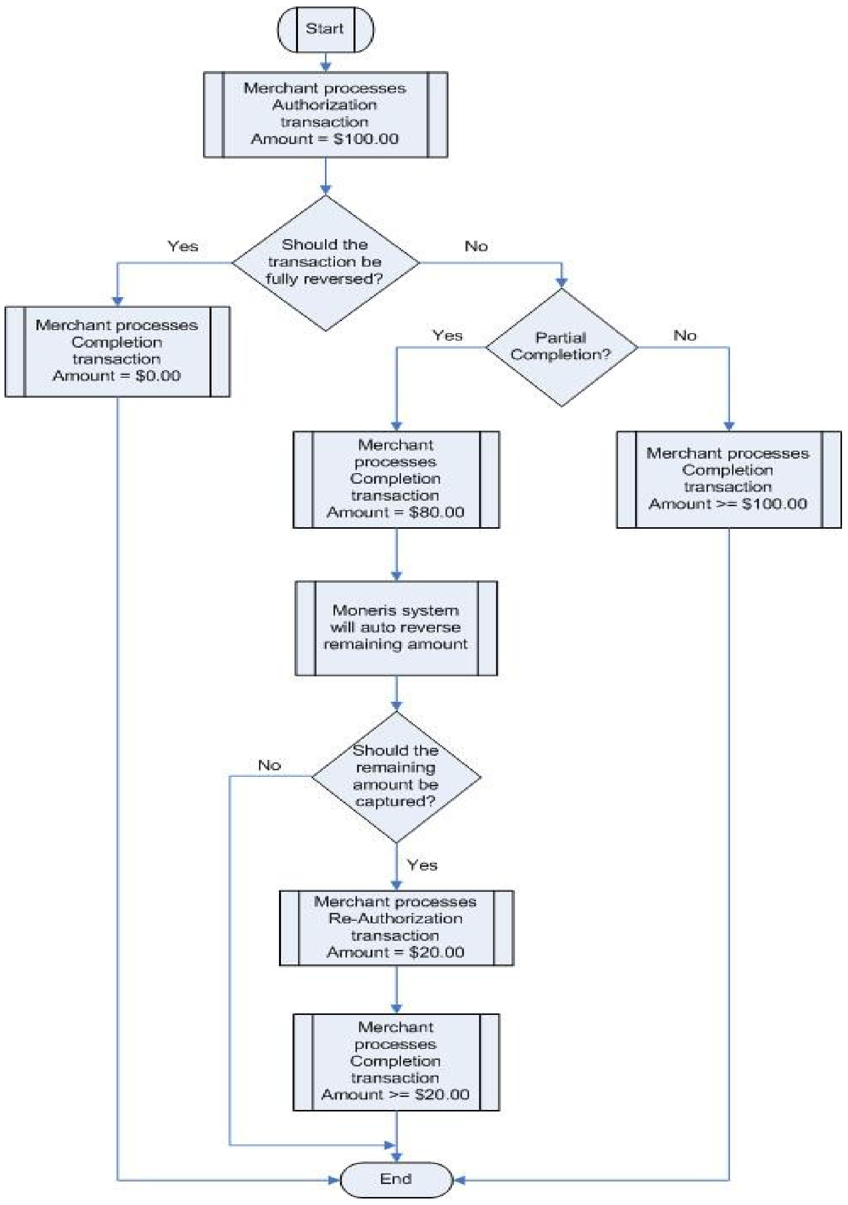


Transactions with no Follow-on required



* Prior to the Batch closing
** After Batch is closed

Process Flow for PreAuth / ReAuth / Capture Transactions


Start
Merchant processes Authorization transaction Amount = $100.00
Should the transaction be fully reversed?
Yes
No
Merchant processes Completion transaction Amount = $0.00
Partial Completion?
Yes
No
Merchant processes Completion transaction Amount = $80.00
Merchant processes Completion transaction Amount >= $100.00
Moneris system will auto reverse remaining amount
Should the remaining amount be captured?
No
Yes
Merchant processes Re-Authorization transaction Amount = $20.00
Merchant processes Completion transaction Amount >= $20.00
End

Encrypted Transactions

The following Encrypted Transactions are available:

- Encrypted Purchase (sale)
- Encrypted PreAuth (authorisation / preauthorisation)
- Encrypted Independent Refund (Credit)
- Encrypted Force Post (Offline Sale)
- Encrypted Card Verification (Account Status Inquiry)

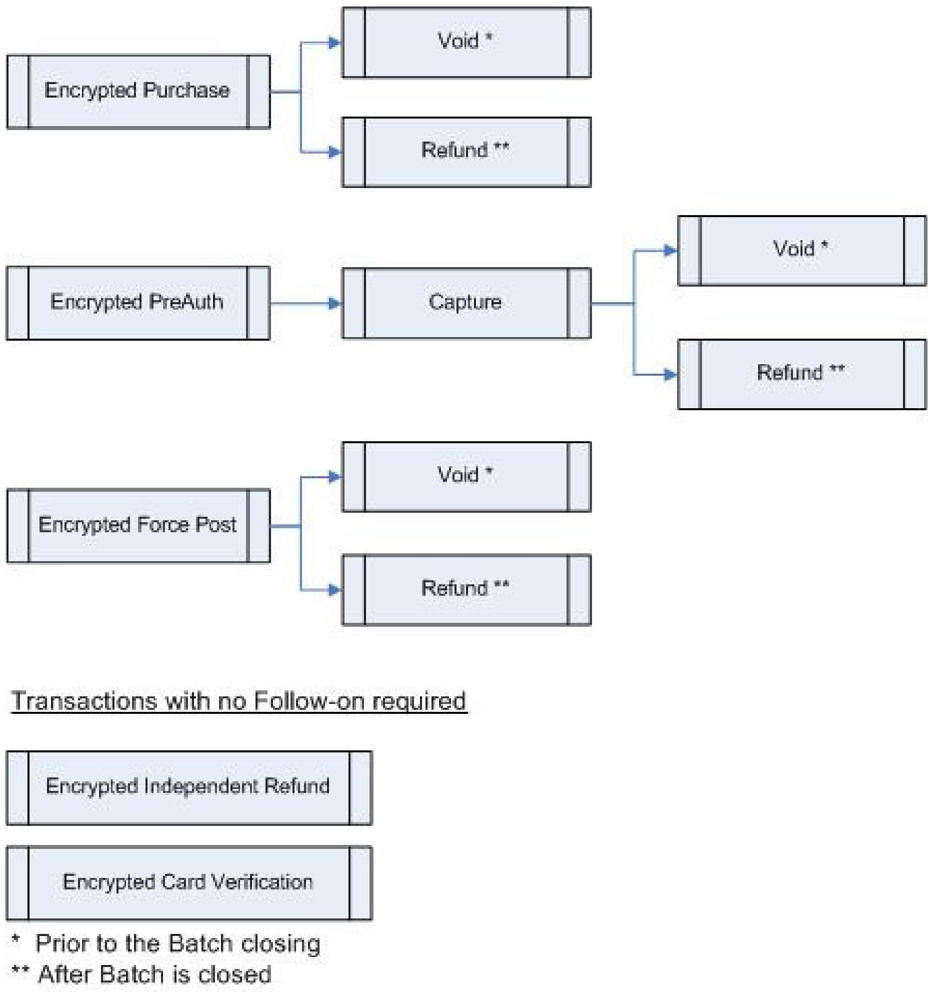
These transaction types are identical to those listed above in the Basic Transaction set, but in this case the card data must be entered via a Moneris provided encrypted MSR device.

NOTE

Please note, the Encrypted Transactions may only be used with a Moneris provided encrypted mag swipe reader. To enquire about the encrypted MSR, please call the Service Centre at 1-866-423-8475.

This transaction set applies to card not present transactions. Please refer to the Encrypted Mag Swiped Transactions for the swiped and manually keyed card present transaction set.

Process Flow for Encrypted Credit Card Transactions



Mag Swipe Transactions

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

Mag Swipe PreAuth – (authorisation / preauthorisation) The Mag Swipe PreAuth requires a credit card to be swiped. It then verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from a Mag Swipe PreAuth so that they may be settled in the merchant's account a Mag Swipe Capture must be performed.

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

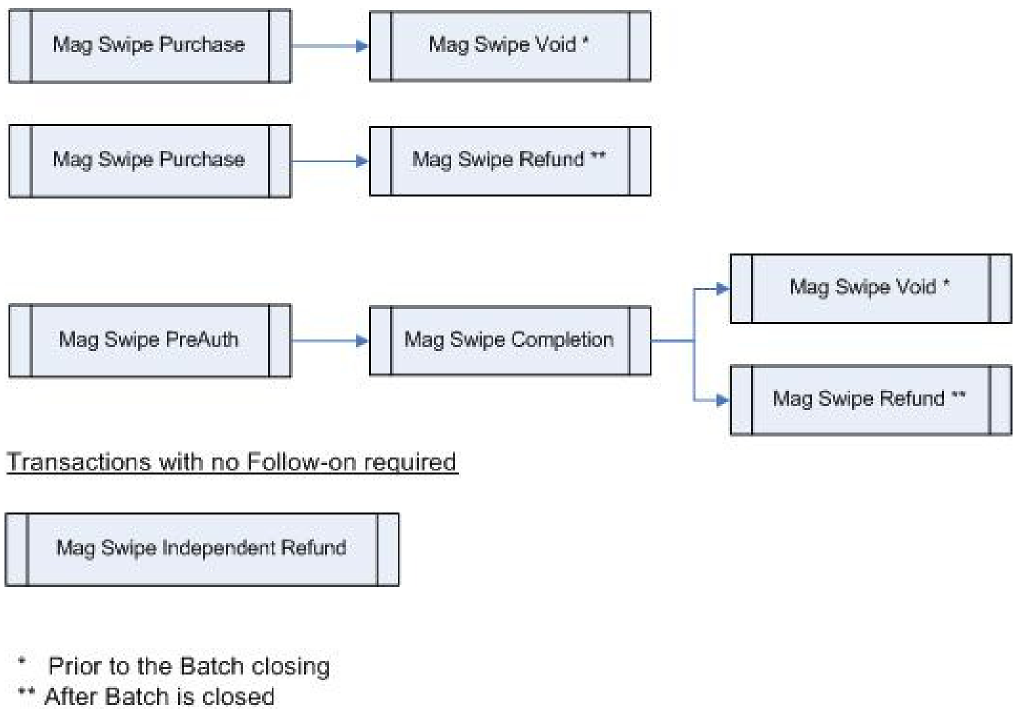
Mag Swipe Void – (Correction / Purchase Correction) Mag Swipe Purchases and Mag Swipe Captures can be voided the same day* that they occur. A Mag Swipe Void must be for the full amount of the transaction and will remove any record of it from the cardholder's statement.

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

Mag Swipe Independent Refund – (Credit) A Mag Swipe Independent Refund requires a credit card to be swiped. It can be performed to credit money to this particular credit card. This transaction does not require a prior Mag Swipe Purchase or Mag Swipe Capture. Please note, the Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an independent refund, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

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

Process Flow for Mag Swipe Credit Card Transactions



Encrypted Mag Swipe Transactions

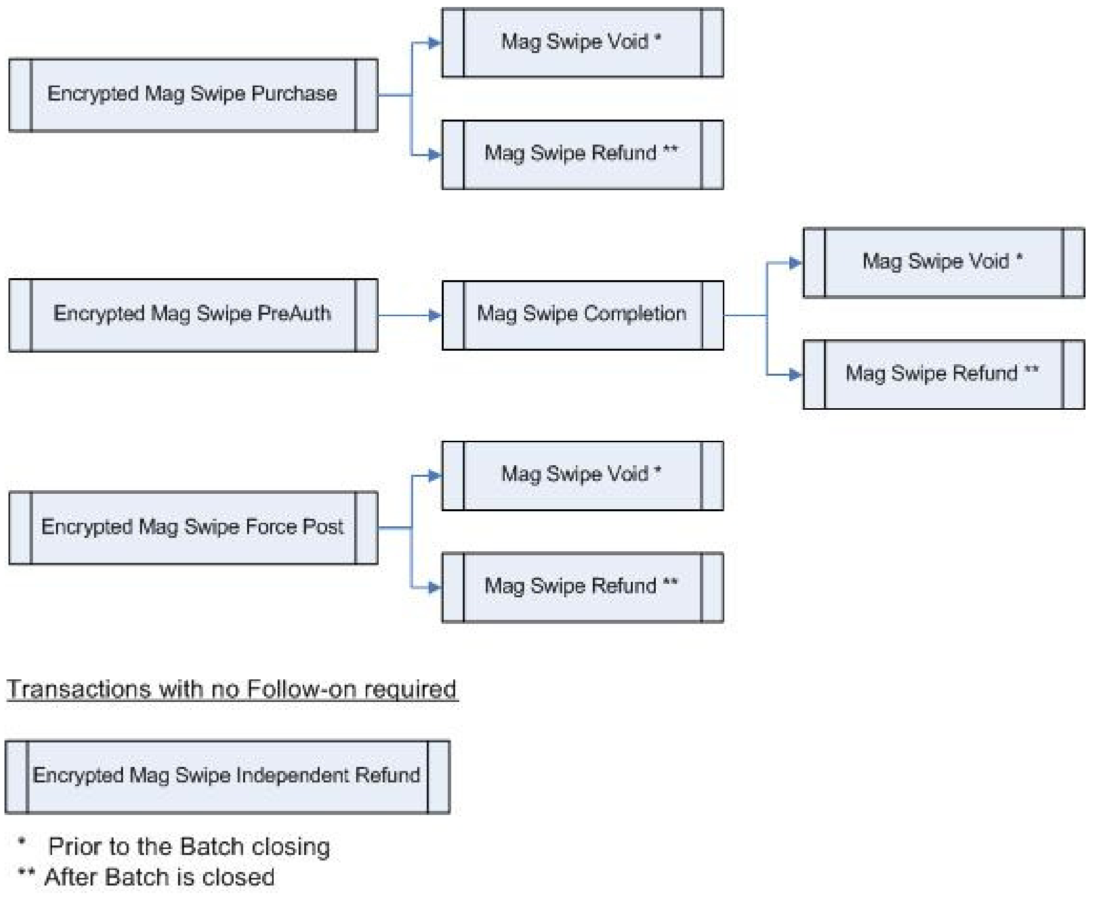
The following Encrypted Mag Swipe Transactions are available:

- Encrypted Mag Swipe Purchase (sale)
- Encrypted Mag Swipe PreAuth (authorisation / preauthorisation)
- Encrypted Mag Swipe Independent Refund (Credit)
- Encrypted Mag Swipe Force Post (Offline Sale)

These transaction types are identical to those listed above in the Mag Swipe Transaction set, but in this case the card data must be swiped or keyed in via a Moneris provided encrypted mag swipe reader

NOTE	Please note, the Encrypted Mag Swipe Transactions may only be used with a Moneris provided encrypted mag swipe reader. To enquire about the encrypted MSR, please call the Service Centre at 1-866-423-8475.

Process Flow for Encrypted Mag Swipe Credit Card Transactions

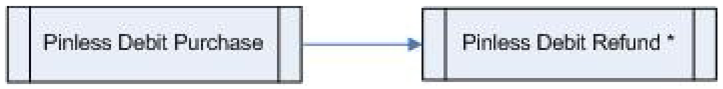

Encrypted Mag Swipe Purchase
Mag Swipe Void *
Mag Swipe Refund **
Encrypted Mag Swipe PreAuth
Mag Swipe Completion
Mag Swipe Void *
Mag Swipe Refund **
Encrypted Mag Swipe Force Post
Mag Swipe Void *
Mag Swipe Refund **
Transactions with no Follow-on required
Encrypted Mag Swipe Independent Refund
* Prior to the Batch closing
** After Batch is closed

Pinless Debit Transactions

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

Pinless Debit Refund – (Credit) A Pinless Debit Refund transaction can be performed against a Pinless Debit Purchase. No amount is required because the Pinless Debit Refund is always for the full amount of the original transaction.

Process Flow for Pinless Debit Transactions



* Prior to the Batch closing

ACH Transactions

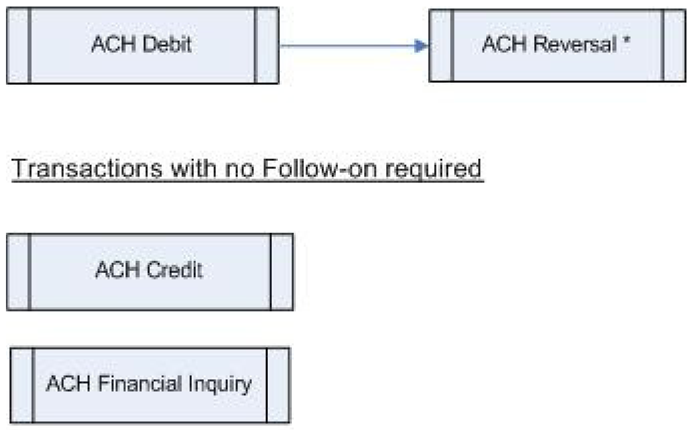
ACH Debit – The ACH Debit transaction verifies and collects the customer's bank account information, removes the funds directly from their bank account and readies them for deposit into the merchant's account.

ACH Reversal – The ACH Reversal transaction can be performed against a previously completed ACH Purchase transaction, the full amount of the original ACH Debit transaction will be refunded. An ACH Reversal may only be performed as long as the ACH Debit was performed within the last 3 months.

ACH Credit – The ACH Credit transaction verifies and collects the customer's bank account information to allow the merchant to transfer funds from their own bank account directly into the customer's bank.

ACH Financial Inquiry – The ACH Fi Inquiry allows the merchant to submit a routing number and verify which Financial Institution it belongs to. This transaction also allows the merchant to verify whether or not this is a valid routing number before submitting an ACH Debit or Credit transaction.

Process Flow for ACH Transactions



* Prior or After the Batch closing

7. Basic 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, dynamic_descriptor, and two optional Level 2 variables (commcard_invoice and commcard_tax_amount) available for Corporate Purchasing Cards. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expiry_date=$argv[6];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         pan=>$pan,
         expdate=>$expiry_date,
         crypt_type=>'7',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15',
         dynamic_descriptor='online sale'
           );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

PreAuth (basic)

The PreAuth is virtually identical to the Purchase with the exception of the transaction type. It is 'us_preauth' instead of 'us_purchase'. 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. Please refer to Appendix A. Definition of Request Fields for variable definitions.

A PreAuth transaction **<u>must</u>** be reversed if it is not to be captured. To reverse an authorization, please refer to the Capture transaction. If you have any questions regarding uncaptured authorization transactions, please refer to the Service Centre at 1-800-471-9511. Please use the 'us_card_verification' transaction type if the intent is to simply verify the card. For a process flow, please refer to Process Flow for PreAuth / ReAuth / Capture Transactions.

```php
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="preauth_".date("dmy-G:i:s");
$amount="1.00";
$pan="4242424242424242";
$expdate="1111";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_preauth',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         pan=>$pan,
         expdate=>$expdate,
         crypt_type=>'7',
         dynamic_descriptor='online sale'
           );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

ReAuth

The ReAuth is virtually identical to the PreAuth with the exception of the transaction type. It is 'us_reauth' instead of 'us_preauth'. Like the PreAuth example, ReAuth's require several variables (store_id, api_token, order_id, amount, orig_order_id, txn_number, and crypt_type). There are also optional fields, such as cust_id and dynamic_descriptor. 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-authorise 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.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$orig_order_id=$argv[4];
$txn_number=$argv[5];
$amount=$argv[6];
$crypt=$argv[7];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_reauth',
         order_id=>$orderid,
         cust_id=>'cust',
         orig_order_id=>$orig_order_id,
         txn_number=>$txn_number,
         amount=>$amount,
         crypt_type=>'7'
           );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Capture

The Capture transaction is used to secure the funds locked by a PreAuth or ReAuth transaction. When sending a 'us_completion' request you will need two pieces of information from the original PreAuth – the order_id and the txn_number from the returned response. There are also two optional Level 2 variables (commcard_invoice and commcard_tax_amount) that may be submitted for Corporate Purchasing Cards.

A PreAuth or ReAuth transaction can only be captured ones. Please refer to the ReAuth transaction for more information on how to perform multiple Captures.

To reverse the full amount of the PreAuth, please use the Capture transaction with a dollar amount of "0.00". For a process flow, please refer to the Process Flow for PreAuth / ReAuth / Capture Transactions.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

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

$compamount=$argv[5];

/************************ Transaction Array *********************************/

$txnArray=array(type=>'us_completion',
         order_id=>$orderid,
         comp_amount=>$compamount,
         txn_number=>$txnnumber,
         crypt_type=>'7',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15'
           );
/************************ Transaction Object ********************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ HttpsPost Object **********************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

/************************ Receipt ******************************************/

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

?>
```

Void

The Void (us_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 'us_purchasecorrection' the order_id and txn_number from the 'us_completion' or 'us_purchase' are required.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

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

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_purchasecorrection',
         order_id=>$orderid,
         txn_number=>$txnnumber,
         crypt_type=>'7'
        );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

print ("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\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 'us_refund' you will require the order_id and txn_number from the original 'us_completion' or 'us_purchase'.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$txnnumber=$argv[4];
$amount=$argv[5];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_refund',
         order_id=>$orderid,
         amount=>$amount,
         txn_number=>$txnnumber,
         crypt_type=>'7'
           );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Independent Refund

The Independent Refund (us_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 optional fields, such as cust_id and dynamic_descriptor. The transaction format is almost identical to a Purchase or a PreAuth.

NOTE The Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an **independent refund**, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="ind_refund_".date("dmy-G:i:s");
$custid="customer1";
$amount="1.00";
$pan="4242424242424242";
$expdate="1111";
$dynamic_descriptor='location #1234';

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_ind_refund',
         order_id=>$orderid,
         cust_id=>$custid,
         amount=>$amount,
         pan=>$pan,
         expdate=>$expdate,
         crypt_type=>'7',
         dynamic_descriptor=>$dynamic_descriptor
           );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

Force Post

The Force Post (us_forcepost) transaction is used when a merchant obtains the authorization number directly from the issuer using a phone or any third party authorization method. The Force Post does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card number, expiry date and the authorization number will need to be passed. There are also optional fields, such as cust_id and dynamic_descriptor.

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

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

$store_id='monusqa002';
$api_token='qatoken';

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

$type='us_forcepost';
$cust_id='CUST13343';
$order_id='ord-'.date("dmy-G:i:s");
$amount='10.00';
$pan='4242424242424242';
$expiry_date='0812';
$auth_code='123456';
$crypt='7';
$dynamic_descriptor='location #1234';

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

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/***************************** 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("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());

?>
```

8. Basic Transactions with Extra Features - Examples

In the previous section the instructions were provided for the basic transaction set. eSELECTplus also provides several extra features/functionalities for the basic transactions. These features include storing customer and order details, Verified by Visa / SecureCode and sending transactions to the Recurring Billing feature. Verified by Visa / SecureCode and Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 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. Please see Appendix C. CustInfo Fields for description of each of the fields. The identical format is used for PreAuth with the exception of transaction type which changes from 'us_purchase' to 'us_preAuth'. Customer details can only be sent with Purchase and PreAuth. It can be used in conjunction with other extra features such as VBV/MCSC and Recurring Billing. ***Please note that the mpgCustInfo fields are not used for any type of address verification or fraud check.***

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expiry_date=$argv[6];

/************************ CustInfo Object **********************************/

$mpgCustInfo = new mpgCustInfo();

/********************* Set E-mail and Instructions **************/

$email ='Joe@widgets.com';
$mpgCustInfo->setEmail($email);

$instructions ="Make it fast";
$mpgCustInfo->setInstructions($instructions);

/********************* Create Billing Array and set it **********/

$billing = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                  country => 'Canada',
                  phone_number => '416-555-5555',
                  fax => '416-555-5555',
                  tax1 => '123.45',
                  tax2 => '12.34',
                  tax3 => '15.45',
                  shipping_cost => '456.23');

$mpgCustInfo->setBilling($billing);

/********************* Create Shipping Array and set it **********/

$shipping = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                   country => 'Canada',
                   phone_number => '416-555-5555',
                   fax => '416-555-5555',
                   tax1 => '123.45',
                   tax2 => '12.34',
                   tax3 => '15.45',
                   shipping_cost => '456.23');

$mpgCustInfo->setShipping($shipping);

/********************** Create Item Arraya and set them **********/

$item1 = array (name=>'item 1 name',
                quantity=>'53',
                product_code=>'item 1 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item1);


$item2 = array(name=>'item 2 name',
                quantity=>'53',
                product_code=>'item 2 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item2);

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         pan=>$pan,
         expdate=>$expiry_date,
         crypt_type=>'7',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15',
         dynamic_descriptor='online sale'
           );

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

$mpgTxn = new mpgTransaction($txnArray);

/************************ Set CustInfo Object *****************************/

$mpgTxn->setCustInfo($mpgCustInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Purchase (with Verified by Visa / MasterCard SecureCode)

Below is an example of sending a Purchase with the Verified by Visa / SecureCode 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 'us_cavv_purchase' to 'us_cavv_preauth'. VBV/MCSC must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated. The optional customer and order details can be included in the transaction using the method outlined above - *Purchase (with Customer and Order Details).*

```
<?php

require "../mpgClasses.php";

/******************************* Request Variables ********************************/
$store_id='monusqa002';
$api_token='qatoken';

/****************************** Transactional Variables ***************************/
$type='us_cavv_purchase';
$order_id='apr25test12';
$cust_id='customer1';
$amount='1.00';
$pan='4242424242424242';
$expiry_date='1111';
$cavv='AAABBJg0VhI0VniQEjRWAAAAAAA=';
$commcard_invoice='Invoice 5757FRJ8';
$commcard_tax_amount='0.15';

/*************************** Transaction Associative Array ************************/
$txnArray=array(type=>$type,
               order_id=>$order_id,
               cust_id=>$cust_id,
               amount=>$amount,
               pan=>$pan,
               expdate=>$expiry_date,
               cavv=>$cavv,
               commcard_invoice=>$commcard_invoice,
               commcard_tax_amount=>$commcard_tax_amount,
               dynamic_descriptor='online sale' );

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

As part of the VbV response there will be an additional method called getCavvResultCode( ). Please refer to Appendix M. CAVV Result Code for a list of possible values.

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 and Recur Update Fields for description of each of the fields. The optional customer and order details can be included in the transaction using the method outlined above -*Purchase (with Customer and Order Details).* Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables ***************************/
$store_id=$argv[1];
$api_token=$argv[2];

/******************** Transactional Variables ************************/
$type='us_purchase';
$order_id=$argv[3];
$cust_id=$argv[4];
$amount=$argv[5];
$pan=$argv[6];
$expiry_date=$argv[7];
$crypt='7';
$commcard_invoice='Invoice 5757FRJ8';
$commcard_tax_amount='0.15';

/************************** Recur Variables *****************************/
$recurUnit = 'day';
$startDate = '2012/11/30';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';

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

/****************************** Recur Object *************************/
$mpgRecur = new mpgRecur($recurArray);

/***************** 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,
                commcard_invoice=>$commcard_invoice,
                commcard_tax_amount=>$commcard_tax_amount,
                dynamic_descriptor='online sale'
                );

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

/****************************** Set Recur Object *********************/
$mpgTxn->setRecur($mpgRecur);
/****************************** Request Object *************************/
$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

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.

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 'us_purchase', 'us_preauth', 'us_cavv_purchase', and 'us_cavv_preauth'. To form CvdInfo please refer to Appendix I. Card Validation Digits (CVD), to form AvsInfo please refer to Appendix J. Address Verification Service (AVS). To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

We strongly recommend that you include Address Verification (AVS) with all of your manually input transactions (MOTO/eCommerce). Doing so will ensure transactions are qualifying at the best possible interchange rate and will minimize costs to accept credit cards. If AVS is not present, the transaction may be assessed a higher interchange fee.

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

require "../mpgClasses.php";

/************************ Request Variables ***************************/
$store_id=$argv[1];
$api_token=$argv[2];

/******************** Transactional Variables ************************/
$type='us_purchase';
$order_id=$argv[3];
$cust_id=$argv[4];
$amount=$argv[5];
$pan=$argv[6];
$expiry_date=$argv[7];
$crypt='7';
$commcard_invoice='Invoice 5757FRJ8';
$commcard_tax_amount='0.15';

/************************* AVS Variables ****************************/
$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';

/************************* CVD Variables ****************************/
$cvd_indicator = '1';
$cvd_value = '198';

/********************* AVS Associative Array ************************/
$avsTemplate = array(
                  avs_street_number=>$avs_street_number,
                avs_street_name =>$avs_street_name,
                avs_zipcode => $avs_zipcode
                );

/********************* CVD Associative Array ************************/
$cvdTemplate = array(
                  cvd_indicator => $cvd_indicator,
                cvd_value => $cvd_value
                );

/************************* AVS Object ********************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

/************************* CVD Object ********************************/
$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
```

```
/***************** 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,
                 commcard_invoice=>$commcard_invoice,
                 commcard_tax_amount=>$commcard_tax_amount,
               dynamic_descriptor='online sale'
                 );

/********************** 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("\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("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());

?>
```

As part of the eFraud response there will be two additional methods called getAvsResultCode( ) and getCvdResultCode( ). For a list of possible CVD responses please refer to Appendix I. Card Validation Digits (CVD) and for a list of AVS responses, please refer to Appendix J. Address Verification Service (AVS).

Purchase (with Status Check)

The Status Check functionality may be used to verify whether a transaction processed successfully. If once a transaction is processed no response is received due to communication/technical difficulties, you may resend the transaction with the Status Check flag set to true. This will then initiate a lookup of the original transaction and eSelectPlus will verify whether it was processed or not. If the Status Check is set to false, the transaction will process as a net new transaction.

The Status Check 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 Status Check request, i.e. if you send a Completion with Status Check, include the same values as the original Completion such as the Order ID, amount, Txn number, etc. What you will get back is a Status Code and Status Message in the Receipt as shown in the sample below. A Status Code of 0-49 indicates successful and 50-999 is not successful. The Status Message will be Found (Status Code of 0-49) or Not Found or null (Status Code of 50-999). When it is found, the other Response will also return the other parameters in the Receipt which will contain the response values for the original transaction allowing you to update your system and provide a receipt to the customer. When it is not found, these additional response values 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_check. Please refer Appendix B. Definitions of Response Fields for variable definitions.

Please note, Status Check is supported on the following transaction types:

Basic Transactions	**VbV/SecureCode**	**Mag Swipe Transactions**	**ACH Transactions**
us_purchase	us_cavv_purchase	us_track2_purchase	us_ach_debit
us_refund	us_cavv_preauth	us_track2_refund	us_ach_reversal
us_ind_refund		us_track2_ind_refund	us_ach_credit
us_preauth		us_track2_preauth	
us_completion		us_track2_completion	
us_purchasecorrection		us_track2_purchasecorrection	
us_forcepost		us_track2_forcepost	

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

require "../mpgClasses.php";

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

$store_id=$argv[1];
$api_token=$argv[2];
$status = 'true';

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expiry_date=$argv[6];

/************************ Transaction Array **********************************/
```

```
$txnArray=array(type=>'us_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         pan=>$pan,
         expdate=>$expiry_date,
         crypt_type=>'7',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15'
           );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

$mpgHttpPost = new mpgHttpsPostStatus($store_id,$api_token,$status,$mpgRequest);

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
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 for more a full description of each field.

9. Encrypted Transaction Examples

Included below is the sample code for the Encrypted transactions that can be found in the "Examples" folder of the PHP API download. Encrypted Basic transactions allow the merchant to key in the credit card using a Moneris provided encrypted reader and submit the encrypted details. These transactions support the submission of the 'enc_track2' value only. Please note, these Encrypted Transactions are only applicable to card not present transactions requiring a credit card number. For the corresponding follow-on transactions such as Capture, Void and Refund please refer to the Basic Transaction Examples

The encrypted MSR device may be used for processing swiped card present transactions, manually keyed card present transactions, as well as card not present transactions. This section refers only to the card not present transaction set. For card present encrypted transactions, please refer to the Encrypted Mag Swipe Transaction Examples

NOTE Please note, the Encrypted Transactions may only be used with a Moneris provided encrypted mag swipe reader. To enquire about the encrypted MSR, please call the Service Centre at 1-866-423-8475.

Encrypted Purchase

Similar to the Basic Purchase example (us_purchase), in the Encrypted Purchase (us_enc_purchase) example we require several variables (store_id, api_token, order_id, amount, enc_track2, crypt_type, and device_type). There are also a number of optional fields, such as cust_id, dynamic_descriptor, and two optional Level 2 variables (commcard_invoice and commcard_tax_amount) available for Corporate Purchasing Cards. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/*********************** Transaction Variables ****************************/

$orderid="enc_purchase_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
        ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
        890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
        239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
        60742A3506C31415939320000A000283C5E03";
$device_type="idtech";
$crypt="7";

// Dynamic Descriptor
//$dynamic_descriptor = "INV002"

/*********************** Transaction Array ********************************/

$txnArray=array(type=>'us_enc_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         //dynamic_descriptor=>$dynamic_descriptor,
         device_type=>$device_type,
         crypt_type=>$crypt
        );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);


/*********************** mpgHttpsPost Object ******************************/
```

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

/*********************** Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();


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

?>
```

Encrypted PreAuth

The Encrypted PreAuth is virtually identical to the Encrypted Purchase with the exception of the transaction type. It is ‘us_enc_preauth’ instead of ‘us_enc_purchase’. Like the Purchase example, the PreAuth requires several variables (store_id, api_token, order_id, amount, enc_track2, crypt_type, and device_type). There are also optional fields, such as cust_id and dynamic_descriptor. Please refer to Appendix A. Definition of Request Fields for variable definitions.

A PreAuth transaction **must** be reversed if it is not to be captured. To reverse an authorization, please refer to the Capture transaction in the Basic Transaction Examples. If you have any questions regarding uncaptured authorization transactions, please refer to the Service Centre at 1-800-471-9511. Please use the USEncCardVerification transaction type if the intent is to simply verify the card. For a process flow, please refer to Process Flow for PreAuth / ReAuth / Capture Transactions.

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

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_preauth_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
         ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
         890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
         239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
         60742A3506C31415939320000A000283C5E03";
$device_type="idtech";
$crypt="7";
//$dynamic_descriptor = "INV002"

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_preauth',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         //dynamic_descriptor=>$dynamic_descriptor,
         device_type=>$device_type,
         crypt_type=>$crypt
        );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

Encrypted Independent Refund

The Encrypted Independent Refund (us_enc_ind_refund) will credit a specified amount to the cardholder's credit card. The Encrypted 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 keyed in via the Moneris provided encrypted MSR device. The Independent Refund transaction requires several variables (store_id, api_token, order_id, amount, enc_track2, crypt_type and device_type). There are also optional fields, such as cust_id and dynamic_descriptor. The transaction format is almost identical to an Encrypted Purchase or PreAuth.

NOTE The Encrypted Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting an encrypted **independent refund**, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_ind_refund".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="";
$device_type="idtech";
$crypt='7';

//dynamic descriptor
//$dyn_descriptor="MYSTORE 12345 INV 2";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_ind_refund',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         device_type=>$device_type,
//         dynamic_descriptor=>$dyn_descriptor,
         crypt_type=>$crypt
          );
/************************ Transaction Object *******************************/

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Encrypted Force Post

The Encrypted Force Post (us_enc_forcepost) transaction is used when a merchant obtains the authorization number directly from the issuer using a phone or any third party authorization method. The Force Post does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card number, expiry date and the authorization number will need to be passed. This transaction allows the merchant to key the card number and expiry date via the Moneris provided Encrypted MSR device. There are also optional fields, such as cust_id and dynamic_descriptor.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_forcepost".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="";
$auth_code="654321";
$device_type="idtech";
$crypt='7';

//dynamic descriptor
//$dyn_descriptor="MYSTORE 12345 INV 2";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_forcepost',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         auth_code=>$auth_code,
         device_type=>$device_type,
//         dynamic_descriptor=>$dyn_descriptor,
         crypt_type=>$crypt
           );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object *****************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

10. Encrypted Transactions with Extra Features - Examples

In the previous section the instructions were provided for the card not present encrypted transaction set. eSELECTplus also provides several extra features/functionalities for the encrypted transactions. These features include storing customer and order details, verfying Card Verification Digits (CVD) and Address Verification (AVS) and sending transactions to the Recurring Billing feature. Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated.

Encrypted Purchase (with Customer and Order details)

Below is an example of sending an Encrypted 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. Please see Appendix C. CustInfo Fields for description of each of the fields. The identical format is used for Encrypted PreAuth with the exception of transaction type which changes from ‘us_enc_purchase’ to ‘us_enc_preauth’. Customer details can only be sent with Purchase and PreAuth. It can be used in conjunction with other extra features such as CVD/AVS and Recurring Billing. ***Please note that the cust_info fields are not used for any type of address verification or fraud check.***

```
<?php

require "mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_purchase_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^*****************************************?*;4924*****
        ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
        890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
        239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
        60742A3506C31415939320000A000283C5E03";
$device_type="idtech";
$crypt="7";
// Dynamic Descriptor
$dynamic_descriptor = "desc";

/************************ CustInfo Object **********************************/

$mpgCustInfo = new mpgCustInfo();

/********************* Set E-mail and Instructions **************/

$email ='Joe@widgets.com';
$mpgCustInfo->setEmail($email);

$instructions ="Make it fast";
$mpgCustInfo->setInstructions($instructions);

/********************* Create Billing Array and set it **********/

$billing = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                  country => 'Canada',
                  phone_number => '416-555-5555',
                  fax => '416-555-5555',
                  tax1 => '123.45',
                  tax2 => '12.34',
                  tax3 => '15.45',
                  shipping_cost => '456.23');

$mpgCustInfo->setBilling($billing);

/********************* Create Shipping Array and set it **********/
```

```
$shipping = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                  country => 'Canada',
                  phone_number => '416-555-5555',
                  fax => '416-555-5555',
                  tax1 => '123.45',
                  tax2 => '12.34',
                  tax3 => '15.45',
                  shipping_cost => '456.23');

$mpgCustInfo->setShipping($shipping);


/******************** Create Item Arraya and set them **********/

$item1 = array (name=>'item 1 name',
                quantity=>'53',
                product_code=>'item 1 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item1);


$item2 = array(name=>'item 2 name',
                quantity=>'53',
                product_code=>'item 2 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item2);

/************************ Transaction Array *********************************/

$txnArray=array(type=>'us_enc_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
       //  dynamic_descriptor=>$dynamic_descriptor,
         device_type=>$device_type,
         crypt_type=>$crypt
         );

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

$mpgTxn = new mpgTransaction($txnArray);

/************************ Set CustInfo Object ****************************/

$mpgTxn->setCustInfo($mpgCustInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Encrypted 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 and Recur Update Fields for description of each of the fields. The optional customer and order details can be included in the transaction using the method outlined above – *Encrypted Purchase (with Customer and Order Details).* Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_purchase_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
        ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
        890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
        239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
        60742A3506C31415939320000A000283C5E03";
$device_type="idtech";
$crypt="7";
// Dynamic Descriptor
//$dynamic_descriptor = "INV002"

/************************** Recur Variables *****************************/

$recurUnit = 'day';
$startDate = '2013/11/30';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';

/****************************** Recur 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
        );

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

$mpgRecur = new mpgRecur($recurArray);

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         //dynamic_descriptor=>$dynamic_descriptor,
         device_type=>$device_type,
         crypt_type=>$crypt
        );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgTxn->setRecur($mpgRecur);

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

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

```
/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

Encrypted Purchase (with CVD and AVS - eFraud)

Below is an example of a Encrypted 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 encrypted transactions: ‘us_enc_purchase’, ‘us_enc_preauth’. To form cvd_info please refer to Appendix I. Card Validation Digits (CVD), to form avs_info please refer to Appendix J. Address Verification Service (AVS). To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

We strongly recommend that you include Address Verification (AVS) with all of your manually input transactions (MOTO/eCommerce). Doing so will ensure transactions are qualifying at the best possible interchange rate and will minimize costs to accept credit cards. If AVS is not present, the transaction may be assessed a higher interchange fee.

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

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_purchase_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^***************************************?*;4924*****
        ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
        890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
        239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
        60742A3506C31415939320000A000283C5E03";
$device_type="idtech";
$crypt="7";
//$commcard_invoice = "INV123";
//$commcard_tax_amount = "1.00";
// Dynamic Descriptor
//$dynamic_descriptor = "INV002"

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

$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';

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

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

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

$avsTemplate = array(
                  avs_street_number=>$avs_street_number,
                  avs_street_name =>$avs_street_name,
                  avs_zipcode => $avs_zipcode
                  );

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

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

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

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

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);

/*********************** Transaction Array **********************************/

$txnArray=array(type=>'us_enc_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         //dynamic_descriptor=>$dynamic_descriptor,
         //commcard_invoice=>$commcard_invoice,
         //commcard_tax_amount=>$commcard_tax_amount,
         device_type=>$device_type,
         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);

/*********************** mpgHttpsPost Object ******************************/

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

/*********************** Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();


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

?>
```

As part of the eFraud response there will be two additional methods called getAvsResultCode( ) and getCvdResultCode( ). For a list of possible CVD responses please refer to Appendix I. Card Validation Digits (CVD) and for a list of AVS responses, please refer to Appendix J. Address Verification Service (AVS).

11. Mag Swipe Transaction Examples

Included below is the sample code for the Mag Swipe transactions that can be found in the "Examples" folder of the PHP API download. Mag Swipe transactions allow the user to swipe their credit card and submit the Track2 details. These transactions support the submission of 'track2', as well as a manual entry of the credit card number and expiry date using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction.

Mag Swipe Purchase

Similar to the basic Purchase, in the Mag Swipe Purchase (us_track2_purchase) example we require several variables (store_id, api_token, order_id, amount, track2 and/or pan, expdate, and pos_code). There are also a number of optional fields, such as cust_id, dynamic_descriptor, and two optional Level 2 variables (commcard_invoice and commcard_tax_amount) available for Corporate Purchasing Cards. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php
require "../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];
/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$custid=$argv[4];
$amount=$argv[5];
/************ Swipe card and read Track1 and/or Track2 **********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);
$startDelim = ";";
$firstChar = $track1{0};
$track = '';
if($firstChar==$startDelim)
{       $track = $track1;
}
else
{       $track2 = fgets ($stdin);
        $track = $track2;
}
$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_purchase',
         order_id=>$orderid,
         cust_id=>$custid,
         amount=>$amount,
         track2=>$track,
         pan=>'',
         expdate=>'',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15',
         pos_code=>'12',
         dynamic_descriptor=>'location no456'
          );
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
?>
```

Mag Swipe PreAuth

The Mag Swipe PreAuth is virtually identical to the Purchase with the exception of the transaction type. It is 'us_track2_preauth' instead of 'us_track2_purchase'. Like the Purchase example, PreAuth's require several variables (store_id, api_token, order_id, amount, track2 and/or pan, expdate, and pos_code). There are also optional fields, such as cust_id and dynamic_descriptor. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];

/************ Swipe card and read Track1 and/or Track2 **********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);

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

$track = '';
if($firstChar==$startDelim)
{
        $track = $track1;
}
else
{
        $track2 = fgets ($stdin);
        $track = $track2;
}

$track = trim($track);

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_preauth',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         track2=>$track,
         pan=>$pan,
         expdate=>$expdate,
         pos_code=>'12',
         dynamic_descriptor=>'location no456'
           );
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);

/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

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

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

Mag Swipe Capture

The Mag Swipe Capture (us_track2_completion) transaction is used to secure the funds locked by a 'us_track2_preauth' transaction. When sending a 'us_track2_completion' request you will need two pieces of information from the original 'us_track2_preauth' – the order_id and the txn_number from the returned response; it does not require the customer to re-swipe the credit card. There are also two optional Level 2 variables (commcard_invoice and commcard_tax_amount) that may be submitted for Corporate Purchasing Cards.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$txnnumber=$argv[4];

$compamount=$argv[5];

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_completion',
         order_id=>$orderid,
         comp_amount=>$compamount,
         txn_number=>$txnnumber,
         pos_code=>'12',
         commcard_invoice=>'Invoice 5757FRJ8',
         commcard_tax_amount=>'0.15'
         );

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

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

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

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

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

?>
```

Mag Swipe Void

The Mag Swipe Void (us_track2_purchasecorrection) transaction is used to cancel a transaction that was performed in the current batch. No amount is required because a Void is always for 100% of the original transaction. The only transactions that can be Voided are Captures and Purchases. To send a 'us_track2_purchasecorrection' the order_id and txn_number from the 'us_track2_completion' or 'us_track2_purchase' are required; it does not require the customer to re-swipe the credit card.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$txnnumber=$argv[4];

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_purchasecorrection',
         order_id=>$orderid,
         txn_number=>$txnnumber
         );

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

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

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

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

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

?>
```

Mag Swipe Refund

The Mag Swipe Refund (us_track2_refund) will credit a specified amount to the cardholder's credit card. A Refund can be sent up to the full value of the original Capture or Purchase. To send a 'us_track2_refund' you will require the order_id and txn_number from the original 'us_track2_completion' or 'us_track2_purchase'; it does not require the customer to re-swipe the credit card.

```
<?php

require "../mpgClasses.php";

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

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

/************************ Transaction Variables ******************************/

$orderid=$argv[3];
$amount=$argv[4];
$txnnumber=$argv[5];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_track2_refund',
         order_id=>$orderid,
         amount=>$amount,
         txn_number=>$txnnumber
           );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Mag Swipe Independent Refund

The Mag Swipe Independent Refund (us_track2_ind_refund) will credit a specified amount to the cardholder's credit card. The Mag Swipe Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card will need to be swiped to provide the track2. There are also optional fields that may be submitted, such as cust_id and dynamic_descriptor. The transaction format is almost identical to a Mag Swipe Purchase or a Mag Swipe PreAuth.

NOTE The Mag Swipe Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting a mag swipe **independent refund**, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

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

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];
$orderid=$argv[3];
$custid=$argv[4];
$amount=$argv[5];

/*************** Swipe Card and read Track1 and/or Track2 *******************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);

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

$track = '';

if($firstChar==$startDelim)
{       $track = $track1;   }
else
{       $track2 = fgets ($stdin);
        $track = $track2;   }

$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_ind_refund',
         order_id=>$orderid,
         cust_id=>$custid,
         amount=>$amount,
         track2=>$track,
         pan=>'',
         expdate=>'',
         pos_code=>'12',
         dynamic_descriptor=>'location no456'  );

/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($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("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
?>
```

Mag Swipe Forcepost

The Mag Swipe Force Post (us_track2_forcepost) is used when a merchant obtains the authorization number directly from the issuer using a phone or any third party authorization method. The Mag Swipe Force Post does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card will need to be swiped to provide the track2 data. There are also optional fields that may be submitted, such as cust_id and dynamic_descriptor. To complete the transaction, the authorization number obtained from the issuer must also be entered.

```
<?php
require "../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='monusqa002';
$api_token='qatoken';
/************************ Transaction Variables ******************************/
$orderid='ord-'.date("dmy-G:i:s");
$custid='cust id';
$amount='1.00';
$authcode='123456';
$dynamic_descriptor='location #456';
/*************** Swipe Card and read Track1 and/or Track2 *******************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);

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

$track = '';

if($firstChar==$startDelim)
{
        $track = $track1;
}
else
{
        $track2 = fgets ($stdin);
        $track = $track2;
}
$track = trim($track);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_forcepost',
         order_id=>$orderid,
         cust_id=>$custid,
         amount=>$amount,
         track2=>$track,
         pan=>'',
         expdate=>'',
         pos_code=>'00',
         auth_code=>$authcode,
         dynamic_descriptor=>$dynamic_descriptor
           );
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

12. Mag Swipe Transactions with Extra Features - Examples

In the previous section the instructions were provided for the Mag Swipe transaction set. eSELECTplus also provides the ability to perform an Address Verification Service check with a Mag Swipe transaction.

Mag Swipe Purchase (with Address Verification Service – AVS)

Below is an example of a Mag Swipe Purchase transaction with AVS information. With this feature enabled in your merchant profile, you will be able to pass in these fields for both 'us_track2_purchase' and 'us_track2_preauth' transactions. To form mpgAvsInfo please refer to Appendix J. Address Verification Service (AVS). To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

We strongly recommend that you include Address Verification (AVS) with all of your manually input transactions (MOTO/eCommerce). Doing so will ensure transactions are qualifying at the best possible interchange rate and will minimize costs to accept credit cards. If AVS is not present, the transaction may be assessed a higher interchange fee.

When testing AVS (eFraud) 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

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

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$custid=$argv[4];
$amount=$argv[5];

/************************** AVS Variables ****************************/
$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';

/********************** AVS Associative Array *************************/
$avsTemplate = array(
                  avs_street_number=>$avs_street_number,
                  avs_street_name =>$avs_street_name,
                  avs_zipcode => $avs_zipcode
                  );
/************************** AVS Object ********************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

/************ Swipe card and read Track1 and/or Track2 **********************/
$stdin = fopen("php://stdin", 'r');
$track1 = fgets ($stdin);

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

$track = '';

if($firstChar==$startDelim)
{
        $track = $track1;
}
else
{
        $track2 = fgets ($stdin);
        $track = $track2;
}

$track = trim($track);
```

```
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_track2_purchase',
                     order_id=>$orderid,
                     cust_id=>$custid,
                     amount=>$amount,
                     track2=>$track,
                     pan=>'',
                     expdate=>'',
                     commcard_invoice=>'Invoice 5757FRJ8',
                     commcard_tax_amount=>'0.15',
                     pos_code=>'12',
                     dynamic_descriptor=>'location no456'
                     );

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

/************************ Set AVS *****************************************/
$mpgTxn->setAvsInfo($mpgAvsInfo);

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

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

As part of the AVS (eFraud) response there will be an additional method called getAvsResultCode( ). For a list of possible AVS responses, please refer to Appendix J. Address Verification Service (AVS)

NOTE Please note, the above transaction with AVS information may not be tested within our current test environment because the test Visa card numbers can not be swiped. To receive an AVS response you may pass the test Visa card number as the 'pan'.

13. Encrypted Mag Swipe Transaction Examples

Included below is the sample code for the Encrypted Mag Swipe transactions that can be found in the “Examples” folder of the PHP API download. Encrypted Mag Swipe transactions allow the user to swipe their credit card using a Moneris provided encrypted mag swipe reader and submit the encrypted Track2 details. These transactions support the submission of the ‘enc_track2’ value only. Please note, the Encrypted Mag Swipe Transactions are only applicable to transactions requiring a credit card number. For card present manually entered and follow-on transactions such as Capture, Void and Refund please refer to the Mag Swipe Transaction Examples.

The encrypted MSR device may be used for processing swiped card present transactions, manually keyed card present transactions, as well as card not present transactions. This section refers only to the swiped and manually keyed card present transaction set. For card not present encrypted transactions, please refer to the Encrypted Transaction Examples.

NOTE Please note, the Encrypted Mag Swipe Transactions may only be used with a Moneris provided encrypted mag swipe reader. To enquire about the encrypted MSR, please call the Service Centre at 1-866-423-8475.

Encrypted Mag Swipe Purchase

Similar to the Mag Swipe Purchase (us_track2_purchase), in the Encrypted Mag Swipe Purchase (us_enc_track2_purchase) example we require several variables (store_id, api_token, order_id, amount, enc_track2, pos_code, and device_type). There are also a number of optional fields, such as cust_id, dynamic_descriptor, and two optional Level 2 variables (commcard_invoice and commcard_tax_amount) available for Corporate Purchasing Cards. Please refer to Appendix A. Definition of Request Fields for variable definitions.

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

/************************ Request Variables **********************************/
$store_id="monusqa002";
$api_token="qatoken";

$orderid="purch_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
    ***4030=********************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00890F
    347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B2
    3837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C3
    1415939320000A000283C5E03";
$pos_code="00";
$device_type="idtech";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_track2_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         pos_code=>$pos_code,
         device_type=>$device_type  );

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

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

/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

Encrypted Mag Swipe PreAuth

The Encrypted Mag Swipe PreAuth is virtually identical to the Encrypted Purchase with the exception of the transaction type. It is ‘us_enc_track2_preauth’ instead of ‘us_enc_track2_purchase’. Like the Purchase example, the PreAuth requires several variables (store_id, api_token, order_id, amount, enc_track2, pos_code, and device_type). There are also optional fields, such as cust_id and dynamic_descriptor. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="preauth_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^*****************************************?*;4924*****
    ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00890F
    347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B2
    3837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C3
    1415939320000A000283C5E03";
$pos_code="00";
$device_type="idtech";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_track2_preauth',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         pos_code=>$pos_code,
         device_type=>$device_type
           );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Encrypted Mag Swipe Independent Refund

The Encrypted Mag Swipe Independent Refund (us_enc_track2_ind_refund) will credit a specified amount to the cardholder's credit card. The Encrypted Mag Swipe Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card will need to be swiped using the Moneris provided encrypted mag swipe reader to provide the encrypted track2 details. There are also optional fields that may be submitted, such as cust_id and dynamic_descriptor. The transaction format is almost identical to an Encrypted Mag Swipe Purchase or an Encrypted Mag Swipe PreAuth.

NOTE The Encrypted Mag Swipe Independent Refund transaction may or may not be supported on your account. If you receive a transaction not allowed error when attempting a mag swipe **independent refund**, it may mean the transaction is not supported on your account. If you wish to have the Independent Refund transaction type temporarily enabled (or re-enabled), please contact the Service Centre at 1-800-471-9511.

```php
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

$orderid="indref_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
    ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00890F
    347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B2
    3837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C3
    1415939320000A000283C5E03";
$pos_code="00";
$device_type="idtech";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_track2_ind_refund',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         pos_code=>$pos_code,
         device_type=>$device_type  );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
?>
```

Encrypted Mag Swipe Forcepost

The Encrypted Mag Swipe Force Post (us_enc_track2_forcepost) is used when a merchant obtains the authorization number directly from the issuer using a phone or any third party authorization method. The Encrypted Mag Swipe Force Post does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card will need to be swiped or keyed in using a Moneris provided encrypted mag swipe reader and submit the encrypted Track2 details. There are also optional fields that may be submitted, such as cust_id and dynamic_descriptor. To complete the transaction, the authorization number obtained from the issuer must also be entered.

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_track2_forcepost".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="";
$pos_code="00";
$auth_code="654321";
$device_type="idtech";

//dynamic descriptor
//$dyn_descriptor="MYSTORE 12345 INV 2";

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_track2_forcepost',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         auth_code=>$auth_code,
         pos_code=>$pos_code,
         device_type=>$device_type,
//         dynamic_descriptor=>$dyn_descriptor
           );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

14. Encrypted Mag Swipe Transactions with Extra Features – Examples

In the previous section the instructions were provided for the Encrypted Mag Swipe transaction set. eSELECTplus also provides the ability to perform an Address Verification Service check with an Encrypted Mag Swipe transaction.

Encrypted Mag Swipe Purchase (with Address Verification Service – AVS)

Below is an example of an Encrypted Mag Swipe Purchase transaction with AVS information. With this feature enabled in your merchant profile, you will be able to pass in these fields for both 'us_enc_track2_purchase' and 'us_enc_track2_preauth' transactions. To form avs_info please refer to Appendix J. Address Verification Service (AVS). To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

When testing AVS (eFraud) 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

```
<?php

require "../mpgClasses.php";

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="purch_".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
    ***4030=********************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00890F
    347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B2
    3837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C3
    1415939320000A000283C5E03";
$pos_code="00";
$device_type="idtech";

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

$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';

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

$avsTemplate = array(
        avs_street_number=>$avs_street_number,
         avs_street_name =>$avs_street_name,
         avs_zipcode => $avs_zipcode
           );

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_track2_purchase',
         order_id=>$orderid,
         cust_id=>'cust',
         amount=>$amount,
         enc_track2=>$enc_track2,
         pos_code=>$pos_code,
         device_type=>$device_type
           );

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

$mpgTxn = new mpgTransaction($txnArray);

/************************ Set AVS *****************************************/

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

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nCardLevelResult = " . $mpgResponse->getCardLevelResult());
print("\nMaskedPan = " . $mpgResponse->getMaskedPan());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());

?>
```

As part of the AVS (eFraud) response there will be an additional method called GetAvsResultCode( ). For a list of possible AVS responses, please refer to Appendix J. Address Verification Service (AVS)

NOTE Please note, the above transaction with AVS information may not be tested within our current test environment because the test Visa card numbers can not be swiped. To receive an AVS response you may pass the test Visa card number as the 'pan'.

15. Pinless Debit Transaction Examples

Included below is the sample code for the Pinless Debit transactions that can be found in the "Examples" folder of the PHP API download. Pinless Debit transactions allow the user to submit billing invoice account information and have funds debited from their bank account for bill payment. This transaction type lets the merchant know if the funds are available or not.

Pinless Debit Purchase

In the Pinless Debit Purchase (us_pinless_debit_purchase) example we require several mandatory variables: store_id, api_token, order_id, amount, pan, presentation_type, intended_use and p_account_number. There are also many optional variables, such as the expdate, cust_id. Please refer to Appendix A. Definition of Request Fields and Appendix E. Pinless Debit Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];
$presentation_type = $argv[7];
$intended_use = $argv[8];
$p_account_number = $argv[9];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_pinless_debit_purchase',
                                   order_id=>$orderid,
                                   cust_id=>'cust',                //This field is optional
                                   amount=>$amount,
                                   pan=>$pan,
                                   expdate=>$expdate,              //This field is optional
                                   presentation_type=>$presentation_type,
                                   intended_use=>$intended_use,
                                   p_account_number=>$p_account_number
                                  );

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

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

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

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

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

?>
```

Pinless Debit Refund

The Pinless Debit Refund (us_pinless_debit_refund) transaction is used to refund a prior Pinless Debit Purchase transaction that was performed within the past 3 months. No amount is required because a Pinless Debit Refund is always for 100% of the original transaction. To send a 'us_pinless_debit_refund' the order_id and txn_number from the 'us_pinless_debit_purchase' are required; it does not require the Pinless Debit information to be re-entered.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$amount=$argv[4];
$txnnumber=$argv[5];

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_pinless_debit_refund',
                          order_id=>$orderid,
                          amount=>$amount,
                          txn_number=>$txnnumber
                          );
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);

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

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

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

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

?>
```

16. Pinless Debit Transaction with Extra Features - Examples

Pinless Debit Purchase (with Customer and Order details)

In the Pinless Debit Purchase (us_pinless_debit_purchase) example we require several mandatory variables: store_id, api_token, order_id, amount, pan, presentation_type, intended_use and p_account_number. There is also an optional variable, such as the expdate, cust_id. Please refer to Appendix A. Definition of Request Fields, Appendix C. CustInfo Fields and Appendix E. Pinless Debit Fields for variable definitions.

```
<?php

require "../mpgClasses.php";

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

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

$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];
$presentation_type = $argv[7];
$intended_use = $argv[8];
$p_account_number = $argv[9];

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_pinless_debit_purchase',
                                   order_id=>$orderid,
                                   cust_id=>'cust',                          //This field is optional
                                   amount=>$amount,
                                   pan=>$pan,
                                   expdate=>$expdate,                        //This field is optional
                                   presentation_type=>$presentation_type,
                                   intended_use=>$intended_use,
                                   p_account_number=>$p_account_number
                          );

/************************ CustInfo Object **********************************/

$mpgCustInfo = new mpgCustInfo();

/******************** Set E-mail and Instructions **************/

$email ='Joe@widgets.com';
$mpgCustInfo->setEmail($email);

$instructions ="Make it fast";
$mpgCustInfo->setInstructions($instructions);

/******************** Create Billing Array and set it **********/

$billing = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                  country => 'Canada',
                  phone_number => '416-555-5555',
                  fax => '416-555-5555',
                  tax1 => '123.45',
                  tax2 => '12.34',
                  tax3 => '15.45',
                  shipping_cost => '456.23');

$mpgCustInfo->setBilling($billing);

/******************** Create Shipping Array and set it **********/

$shipping = array( first_name => 'Joe',
                  last_name => 'Thompson',
                  company_name => 'Widget Company Inc.',
                  address => '111 Bolts Ave.',
                  city => 'Toronto',
                  province => 'Ontario',
                  postal_code => 'M8T 1T8',
                  country => 'Canada',
                  phone_number => '416-555-5555',
                  fax => '416-555-5555',
                  tax1 => '123.45',
                  tax2 => '12.34',
                  tax3 => '15.45',
                  shipping_cost => '456.23');

$mpgCustInfo->setShipping($shipping);

/********************* Create Item Arraya and set them **********/

$item1 = array (name=>'item 1 name',
                quantity=>'53',
                product_code=>'item 1 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item1);

$item2 = array(name=>'item 2 name',
                quantity=>'53',
                product_code=>'item 2 product code',
                extended_amount=>'1.00');

$mpgCustInfo->setItems($item2);

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

$mpgTxn = new mpgTransaction($txnArray);

/************************ Set CustInfo Object *****************************/

$mpgTxn->setCustInfo($mpgCustInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Pinless Debit Purchase (with Recurring Billing)

The Pinless Debit Purchase with Recurring Billing (us_pinless_debit_purchase) transaction allows the merchant to submit the transaction information once and then re-bill 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. Please see Appendix A. Definition of Request Fields, Appendix D. Recur and Recur Update Fields and Appendix E. Pinless Debit Fields for description of each of the fields.

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];
/************************ Transaction Variables ******************************/
$orderid=$argv[3];
$amount=$argv[4];
$pan=$argv[5];
$expdate=$argv[6];
$presentation_type = $argv[7];
$intended_use = $argv[8];
$p_account_number = $argv[9];

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_pinless_debit_purchase',
                                   order_id=>$orderid,
                                   cust_id=>'cust',                          //This field is optional
                                   amount=>$amount,
                                   pan=>$pan,
                                   expdate=>$expdate,                        //This field is optional
                                   presentation_type=>$presentation_type,
                                   intended_use=>$intended_use,
                                   p_account_number=>$p_account_number
                          );
/************************** Recur Variables *****************************/
$recurUnit = 'day';
$startDate = '2008/11/30';
$numRecurs = '4';
$recurInterval = '10';
$recurAmount = '31.00';
$startNow = 'true';
/****************************** Recur 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
        );
/****************************** Recur Object *************************/
$mpgRecur = new mpgRecur($recurArray);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/****************************** Set Recur Object *********************/
$mpgTxn->setRecur($mpgRecur);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

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

?>
```

17. ACH Transaction Examples

Included below is the sample code for the ACH transactions that can be found in the "Examples" folder of the PHP API download. ACH transactions allow the user to submit bank account information to have funds either debited or credited.

ACH Debit

In the ACH Debit (us_ach_debit) example we require several mandatory variables: store_id, api_token, order_id, amount, sec, routing_num, account_num, and account_type. There are also many optional variables, such as the cust_id, check_num and the customer details. Please refer to Appendix A. Definition of Request Fields for all request variables and Appendix F. AchInfo Fields for all ACH variables. ***Please note that the mpgAchInfo fields are not used for any type of address verification or fraud check.***

ACH Debit (Check not present)

SEC codes for physical check **not present** include: 'web', 'ccd', 'ppd'. In the example below the following variables are required for an ACH Debit (check not present) transaction: routing_num, account_num, check_num, account_type and micr. Please refer to Appendix G. ACH Sec Codes and Process Flow for a full description on the mandatory fields.

```
<?php

require "../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='monusqa002';
$api_token='qatoken';

$orderid='ach-'.date("dmy-G:i:s");
$amount='1.00';
$custid = 'my cust id';

/************************ Transaction Array *********************************/
$txnArray=array(type=>'us_ach_debit',
                         order_id=>$orderid,
                         cust_id=>$custid,
                         amount=>$amount
                         );
/************************** ACH Info Variables *****************************/
$sec = 'ppd';
$cust_first_name = 'Bob';
$cust_last_name = 'Smith';
$cust_address1 = '101 Main St';
$cust_address2 = 'Apt 102';
$cust_city = 'Chicago';
$cust_state = 'IL';
$cust_zip = '123456';

$routing_num = '54321';
$account_num = '23456';
$check_num = '100';
$account_type = 'savings';

/********************* ACH Info Associative Array ************************/
$achTemplate = array(sec =>$sec,
                       cust_first_name => $cust_first_name,
                     cust_last_name => $cust_last_name,
                     cust_address1 => $cust_address1,
                     cust_address2 => $cust_address2,
                     cust_city => $cust_city,
                     cust_state => $cust_state,
                     cust_zip => $cust_zip,
                     routing_num => $routing_num,
                     account_num => $account_num,
                     check_num => $check_num,
                     account_type => $account_type
                    );

/************************** ACH Info Object ********************************/
$mpgAchInfo = new mpgAchInfo ($achTemplate);

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

/************************ Set ACH Info *************************************/
$mpgTxn->setAchInfo($mpgAchInfo);
```

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

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

ACH Debit (Check present)

SEC codes for physical check **present** include: ‘arc’, ‘boc’, ‘pop’. In the example below the following variables are required for an ACH Debit (check present) transaction: micr, dl_num, magstripe, image_front and image_back. Please refer to Appendix G. ACH Sec Codes and Process Flow for a full description on the mandatory fields.

```php
<?php
require "../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='monusqa002';
$api_token='qatoken';
/************************ Transaction Variables ******************************/
$orderid='ach-'.date("dmy-G:i:s");
$amount='1.00';
$custid = 'my cust id';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_ach_debit',
                          order_id=>$orderid,
                          cust_id=>$custid,
                          amount=>$amount
                          );
/************************** ACH Info Variables *****************************/
$sec = 'pop';
$cust_first_name = 'Bob';
$cust_last_name = 'Smith';
$cust_address1 = '101 Main St';
$cust_address2 = 'Apt 102';
$cust_city = 'Chicago';
$cust_state = 'IL';
$cust_zip = '123456';

$micr = 't071000013t742941347o125';
$dl_num = 'CO-12312312';
$magstripe = 'no';
$image_front = '123456789012345678901234567890…';
$image_back = '123456789012345678901234567890…';

/********************** ACH Info Associative Array ************************/
$achTemplate = array(sec =>$sec,
                     cust_first_name => $cust_first_name,
                     cust_last_name => $cust_last_name,
                     cust_address1 => $cust_address1,
                     cust_address2 => $cust_address2,
                     cust_city => $cust_city,
                     cust_state => $cust_state,
                     cust_zip => $cust_zip,
                     micr => $micr,
                     dl_num => $dl_num,
                     magstripe => $magstripe,
                     image_front => $image_front,
                     image_back => $image_back
                     );

/************************** ACH Info Object *******************************/
$mpgAchInfo = new mpgAchInfo ($achTemplate);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set ACH Info *************************************/
$mpgTxn->setAchInfo($mpgAchInfo);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());

?>
```

ACH Credit

In the ACH Credit (us_ach_credit) example we require several mandatory variables: store_id, api_token, order_id, amount, sec, routing_num, account_num, and account_type. There are also many optional variables, such as the cust_id, check_num and the customer details. Please refer to Appendix A. Definition of Request Fields and Appendix F. AchInfo Fields for variable definitions. ***Please note that the mpgAchInfo fields are not used for any type of address verification or fraud check.***

NOTE Please note, the ACH Credit transaction may only be submitted with a SEC Code of ‘ppd’ or ‘ccd’.

```
<?php
require "../mpgClasses.php";
$store_id='monusqa002';
$api_token='qatoken';
$orderid='ach-'.date("dmy-G:i:s");
$amount='1.00';
$custid = 'my cust id';
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_ach_credit',
                          order_id=>$orderid,
                          cust_id=>$custid,
                          amount=>$amount
                          );
/************************** ACH Info Variables *****************************/
$sec = 'ppd';
$cust_first_name = 'Bob';
$cust_last_name = 'Smith';
$cust_address1 = '101 Main St';
$cust_address2 = 'Apt 102';
$cust_city = 'Chicago';
$cust_state = 'IL';
$cust_zip = '123456';
$routing_num = '54321';
$account_num = '23456';
$check_num = '100';
$account_type = 'savings';
/********************** ACH Info Associative Array ************************/
$achTemplate = array(sec =>$sec,
                     cust_first_name => $cust_first_name,
                     cust_last_name => $cust_last_name,
                     cust_address1 => $cust_address1,
                     cust_address2 => $cust_address2,
                     cust_city => $cust_city,
                     cust_state => $cust_state,
                     cust_zip => $cust_zip,
                     routing_num => $routing_num,
                     account_num => $account_num,
                     check_num => $check_num,
                     account_type => $account_type
                    );
/************************** ACH Info Object ********************************/
$mpgAchInfo = new mpgAchInfo ($achTemplate);
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set ACH Info *************************************/
$mpgTxn->setAchInfo($mpgAchInfo);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTicket = " . $mpgResponse->getTicket());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
?>
```

ACH Reversal

The ACH Reversal (us_ach_reversal) transaction is used to reverse a prior ACH Debit transaction that was performed within the past 3 months. No amount is required because a Reversal is always for 100% of the original transaction. To send a 'us_ach_reversal' the order_id and txn_number from the 'us_ach_debit' are required; it does not require the bank account information to be re-entered.

```
<?php

require "../mpgClasses.php";

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

$store_id='monusqa002';
$api_token='qatoken';

/************************ Transaction Variables ******************************/

$orderid='ORDER_ID_FROM_ACHDEBIT';
$txnnumber = 'TXN_ID_FROM_ACHDEBIT';

/************************ Transaction Array **********************************/

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

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

ACH FI Enquiry

The ACH FI Enquiry (us_ach_fi_inquiry) transaction allows the merchant to submit a routing number and verify which Financial Institution it belongs to. This transaction also allows the merchant to verify whether or not this is a valid routing number before submitting an ACH Debit or Credit transaction.

```
<?php

require "../mpgClasses.php";

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

$store_id='monusqa002';
$api_token='qatoken';

/************************ Transaction Variables ******************************/

$routingnum='9123456';

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_ach_fi_enquiry',
                         routing_num=>$routingnum
                         );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

18. ACH Transactions with Extra Features – Examples

In the previous section the instructions were provided for the ACH transaction set. eSELECTplus also provides several extra features/functionalities for the ACH transactions. These features include storing customer and order details and sending transactions to the Recurring Billing feature. Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated.

ACH Debit (with Customer and Order details)

Below is an example of sending an ACH Debit (us_ach_debit) 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. Please see Appendix C. CustInfo Fields for description of each of the fields. Customer details can only be sent with the ACH Debit transaction. It can be used in conjunction with other extra features such as Recurring Billing. ***Please note that the mpgCustInfo fields are not used for any type of address verification or fraud check.***

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

/************************ Request Variables **********************************/
$store_id='monusqa024';
$api_token='qatoken';

/************************ Transaction Variables ******************************/
$orderid='ach-'.date("dmy-G:i:s");
$amount='1.00';
$custid = 'my cust id';

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_ach_debit',
                          order_id=>$orderid,
                          cust_id=>$custid,
                          amount=>$amount
                          );

/************************** ACH Info Variables *****************************/
$sec = 'ppd';
$cust_first_name = 'Bob';
$cust_last_name = 'Smith';
$cust_address1 = '101 Main St';
$cust_address2 = 'Apt 102';
$cust_city = 'Chicago';
$cust_state = 'IL';
$cust_zip = '123456';
$routing_num = '987654321';
$account_num = '23456';
$check_num = '100';
$account_type = 'savings';

/********************** ACH Info Associative Array *************************/
$achTemplate = array(
                       sec =>$sec,
                     cust_first_name => $cust_first_name,
                     cust_last_name => $cust_last_name,
                     cust_address1 => $cust_address1,
                     cust_address2 => $cust_address2,
                     cust_city => $cust_city,
                     cust_state => $cust_state,
                     cust_zip => $cust_zip,
                     routing_num => $routing_num,
                     account_num => $account_num,
                     check_num => $check_num,
                     account_type => $account_type
                    );

/************************** ACH Info Object ********************************/
$mpgAchInfo = new mpgAchInfo ($achTemplate);

/************************ CustInfo Object **********************************/
$mpgCustInfo = new mpgCustInfo();

/********************* Set E-mail and Instructions **************/
$email ='Joe@widgets.com';
$mpgCustInfo->setEmail($email);

$instructions ="Make it fast";
$mpgCustInfo->setInstructions($instructions);
```

```
/********************* Create Billing Array and set it **********/
$billing = array( first_name => 'Joe',
                 last_name => 'Thompson',
                 company_name => 'Widget Company Inc.',
                 address => '111 Bolts Ave.',
                 city => 'Toronto',
                 province => 'Ontario',
                 postal_code => 'M8T 1T8',
                 country => 'Canada',
                 phone_number => '416-555-5555',
                 fax => '416-555-5555',
                 tax1 => '123.45',
                 tax2 => '12.34',
                 tax3 => '15.45',
                 shipping_cost => '456.23');

$mpgCustInfo->setBilling($billing);

/********************* Create Shipping Array and set it **********/
$shipping = array(first_name => 'Joe',
                 last_name => 'Thompson',
                 company_name => 'Widget Company Inc.',
                 address => '111 Bolts Ave.',
                 city => 'Toronto',
                 province => 'Ontario',
                 postal_code => 'M8T 1T8',
                 country => 'Canada',
                 phone_number => '416-555-5555',
                 fax => '416-555-5555');

$mpgCustInfo->setShipping($shipping);

/********************* Create Item Arrays and set them **********/
$item1 = array (name=>'item 1 name',
                 quantity=>'53',
                 product_code=>'item 1 product code',
                 extended_amount=>'1.00');

$mpgCustInfo->setItems($item1);

$item2 = array(name=>'item 2 name',
                 quantity=>'53',
                 product_code=>'item 2 product code',
                 extended_amount=>'1.00');

$mpgCustInfo->setItems($item2);

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

/************************ Set ACH Info *************************************/
$mpgTxn->setAchInfo($mpgAchInfo);

/************************ Set CustInfo *************************************/
$mpgTxn->setCustInfo($mpgCustInfo);

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

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

ACH Debit (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 and Recur Update Fieldsfor descry iption of each of the Recur fields. The optional customer and order details can be included in the transaction using the method outlined above – *ACH Debit (with Customer and Order Details)*. Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated. ***Please note that the Recurring Billing fields are only available to SEC codes 'ppd' 'ccd' and 'web'.***

```
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id='monusqa024';
$api_token='qatoken';

/************************ Transaction Variables ******************************/
$orderid='ach-'.date("dmy-G:i:s");
$amount='1.00';
$custid = 'my cust id';

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_ach_debit',
                          order_id=>$orderid,
                          cust_id=>$custid,
                          amount=>$amount
                          );

/************************** ACH Info Variables *****************************/
$sec = 'ppd';
$cust_first_name = 'Bob';
$cust_last_name = 'Smith';
$cust_address1 = '101 Main St';
$cust_address2 = 'Apt 102';
$cust_city = 'Chicago';
$cust_state = 'IL';
$cust_zip = '123456';
$routing_num = '987654321';
$account_num = '23456';
$check_num = '100';
$account_type = 'savings';

/********************* ACH Info Associative Array *************************/
$achTemplate = array(
                       sec =>$sec,
                     cust_first_name => $cust_first_name,
                     cust_last_name => $cust_last_name,
                     cust_address1 => $cust_address1,
                     cust_address2 => $cust_address2,
                     cust_city => $cust_city,
                     cust_state => $cust_state,
                     cust_zip => $cust_zip,
                     routing_num => $routing_num,
                     account_num => $account_num,
                     check_num => $check_num,
                     account_type => $account_type
                    );

/************************** ACH Info Object *******************************/
$mpgAchInfo = new mpgAchInfo ($achTemplate);

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

/****************************** Recur 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
                     );

/****************************** Recur Object *************************/
$mpgRecur = new mpgRecur($recurArray);

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

/************************ Set ACH Info ************************************/
$mpgTxn->setAchInfo($mpgAchInfo);

/****************************** Set Recur Object *********************/
$mpgTxn->setRecur($mpgRecur);

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

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

19. Administrative Transactions

Included below is the sample code for the Administrative transactions that can be found in the “Examples” folder of the PHP API download. Administrative transactions allow the user to perform such tasks as manually closing an open Batch and preparing the funds for settlement. Also, the user may retrieve details about the currently open Batch without needing to close it.

Batch Close

At the end of every financial day (11pm EST) the Batch needs to be closed in order to have the Credit Card funds settled the next business day and the ACH funds settled, on average, within the next 5 business days. *By default eSELECTplus will automatically close your Batch 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. To disable automatic close please access the Merchant Resource Centre (https://esplus.moneris.com/usmpg), go to the Admin menu item and then choose Store Settings; the Batch Close options are located on this page.

```php
<?php

require "../mpgClasses.php";

/************************ Request Variables ******************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variables **************************/
$ecr_number=$argv[3];

/************************ Transaction Array ******************************/
$txnArray=array(type=>'us_batchclose',
                          ecr_number=>$ecr_number
                          );

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

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

/************************ mpgHttpsPost Object *****************************/
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);

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

/************************ Array of Credit Cards ***************************/
$creditCards = $mpgResponse->getCreditCards($ecr_number);

/************************ Display Loop ************************************/
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]);
 }
```

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
<?php

require "../mpgClasses.php";

/************************ Request Variables **********************************/
$store_id=$argv[1];
$api_token=$argv[2];

/************************ Transaction Variable ******************************/
$ecr_number=$argv[3];

/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_opentotals',
                 ecr_number=>$ecr_number
                 );

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

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

/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost=new mpgHttpsPost($store_id,$api_token,$mpgReq);

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

/************************ Array of Credit Cards  ***************************/
$creditCards = $mpgResponse->getCreditCards($ecr_number);

/************************ Loop through Array and Display *******************/
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 (us_card_verification) 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, expdate). Also, Address Verification (AVS) is required while the 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.

```
<?php
require "../mpgClasses.php";
/************************ Request Variables **********************************/
$store_id='monusqa002';
$api_token='qatoken';
/************************ Transaction Variables ******************************/
$orderid="cardverification".date("dmy-G:i:s");
$pan="4242424242424242";
$expiry_date="1111";
/************************** AVS Variables *****************************/
$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';
/************************** CVD Variables *****************************/
$cvd_indicator = '1';
$cvd_value = '198';

/********************** AVS Associative Array *************************/
$avsTemplate = array( avs_street_number=>$avs_street_number,
                      avs_street_name =>$avs_street_name,
                      avs_zipcode => $avs_zipcode
                      );

/********************** CVD Associative Array *************************/
$cvdTemplate = array(     cvd_indicator => $cvd_indicator,
                          cvd_value => $cvd_value
                          );
/************************* AVS Object ********************************/
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************** CVD Object ********************************/
$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);
/************************ Transaction Array **********************************/
$txnArray=array(type=>'us_card_verification',
         order_id=>$orderid,
         cust_id=>'cust',
         pan=>$pan,
         expdate=>$expiry_date
           );
/************************ Transaction Object *******************************/
$mpgTxn = new mpgTransaction($txnArray);
/************************ Set AVS and CVD *****************************/
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgTxn->setCvdInfo($mpgCvdInfo);
/************************ Request Object **********************************/
$mpgRequest = new mpgRequest($mpgTxn);
/************************ mpgHttpsPost Object ******************************/
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);
/************************ Response Object **********************************/
$mpgResponse=$mpgHttpPost->getMpgResponse();

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

?>
```

Encrypted Card Verification

Similar to the regular Card Verification transactionmentioned above, the Encrypted Card Verification (us_enc_card_verification) 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 Encrypted Card Verification requires the card data to be keyed in via the Moneris provided encrypted MSR device. This transaction requires several variables (store_id, api_token, order_id, enc_track2 and device_type). Also, Address Verification (AVS) is required while the 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.

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

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

$store_id="monusqa002";
$api_token="qatoken";

/************************ Transaction Variables ******************************/

$orderid="enc_cardverification".date("dmy-G:i:s");
$amount="1.00";
$enc_track2="02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^****************************************?*;4924*****
        ***4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283DBEBB2C6B3FDEACF7B5B314219D76C00
        890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A18B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083
        239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B807
        60742A3506C31415939320000A000283C5E03";
$device_type="idtech";

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

$avs_street_number = '201';
$avs_street_name = 'Michigan Ave';
$avs_zipcode = 'M1M1M1';

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

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

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

$avsTemplate = array(
                     avs_street_number=>$avs_street_number,
                     avs_street_name =>$avs_street_name,
                     avs_zipcode => $avs_zipcode
                    );

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

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

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

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

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);

/************************ Transaction Array **********************************/

$txnArray=array(type=>'us_enc_card_verification',
         order_id=>$orderid,
         cust_id=>'cust',
         enc_track2=>$enc_track2,
         device_type=>$device_type
           );

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

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

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

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

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

$mpgRequest = new mpgRequest($mpgTxn);

/************************ mpgHttpsPost Object ******************************/

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

/************************ Response Object **********************************/

$mpgResponse=$mpgHttpPost->getMpgResponse();


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

?>
```

Recur Update

Recur Update allows a user to alter characteristics of a previously registered Recurring Billing transaction. This feature is commonly used to update a customer's credit card information and the number of times it is to be billed (recur). Only fields sent to the gateway will be updated. Please see Appendix A. Definition of Request Fields and Appendix D. Recur and Recur Update Fields for description of each of the fields.

Recur Update – Credit Card example

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

/**************************** Request Variables *******************************/
$store_id='monusqa002';
$api_token='qatoken';

/************************* Transactional Variables ***************************/
$type='us_recur_update';
$order_id='ORDER_ID_FROM_ORIGINAL_TXN';

//The following fields can be updated for a CC, ACH or Pinless Debit transaction
$cust_id='MY CUST ID';
$recur_amount='1.00';
$add_num='20';
$total_num='999';
$hold = 'false';
$terminate = 'false';

//The pan & expdate can be updated for a Credit Card or Pinless Debit transaction
$pan='5454545454545454';
$expdate='1111';

//The AVS details can only be updated for a Credit Card transaction
$avs_street_number = '112';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '123123';

//The p_account_number & presentation_type can only be updated for a Pinless Debit transaction
$p_account_number="Account a12345678 9876543";
$presentation_type = "X";
/*********************** Transactional Associative Array *********************/
$txnArray=array('type'=>$type,
                     'order_id'=>$order_id,
                     'cust_id'=>$cust_id,
                     'recur_amount'=>$recur_amount,
                     'pan'=>$pan,
                     'expdate'=>$expdate,
                     'p_account_number'=>$p_account_number,
                     'presentation_type'=>$presentation_type,
                     'add_num_recurs' => $add_num,
                     'total_num_recurs' => $total_num,
                     'hold' => $hold,
                     'terminate' => $terminate,
                     'avs_street_number' => $avs_street_number,
                     'avs_street_name' => $avs_street_name,
                     'avs_zipcode' => $avs_zipcode
                  );
/**************************** 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());

?>
```

Recur Update – Pinless Debit example

```php
<?php

require "../mpgClasses.php";

/**************************** Request Variables *******************************/
$store_id='monusqa002';
$api_token='qatoken';

/************************* Transactional Variables ****************************/
$type='us_recur_update';
$order_id='ORDER_ID_FROM_ORIGINAL_TXN';

//The following fields can be updated for a CC, ACH or Pinless Debit transaction
$cust_id='MY CUST ID';
$recur_amount='1.00';
$add_num='20';
$total_num='999';
$hold = 'false';
$terminate = 'false';

//The pan & expdate can be updated for a Credit Card or Pinless Debit transaction
$pan='5454545454545454';
$expdate='1111';

//The AVS details can only be updated for a Credit Card transaction
//$avs_street_number = '112';
//$avs_street_name = 'lakeshore blvd';
//$avs_zipcode = '123123';

//The p_account_number & presentation_type can only be updated for a Pinless Debit transaction
$p_account_number="Account a12345678 9876543";
$presentation_type = "X";

/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
                     'order_id'=>$order_id,
                     'cust_id'=>$cust_id,
                     'recur_amount'=>$recur_amount,
                     'pan'=>$pan,
                     'expdate'=>$expdate,
                     'p_account_number'=>$p_account_number,
                     'presentation_type'=>$presentation_type,
                     'add_num_recurs' => $add_num,
                     'total_num_recurs' => $total_num,
                     'hold' => $hold,
                     'terminate' => $terminate,
                     'avs_street_number' => $avs_street_number,
                     'avs_street_name' => $avs_street_name,
                     'avs_zipcode' => $avs_zipcode
                 );

/**************************** 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());

?>
```

Recur Update – ACH example

```
<?php

require "../mpgClasses.php";

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

$store_id='monusqa002';
$api_token='qatoken';

/************************* Transactional Variables ****************************/
$type='us_recur_update';
$order_id='ORDER_ID_FROM_ORIGINAL_TXN';

//The following fields can be updated for a CC, ACH or Pinless Debit transaction
$cust_id='MY CUST ID';
$recur_amount='1.00';
$add_num='20';
$total_num='999';
$hold = 'false';
$terminate = 'false';

//The pan & expdate can be updated for a Credit Card or Pinless Debit transaction
//$pan='5454545454545454';
//$expdate='1111';

//The AVS details can only be updated for a Credit Card transaction
//$avs_street_number = '112';
//$avs_street_name = 'lakeshore blvd';
//$avs_zipcode = '123123';

//The p_account_number & presentation_type can only be updated for a Pinless Debit transaction
//$p_account_number="Account a12345678 9876543";
//$presentation_type = "X";

/*********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
                     'order_id'=>$order_id,
                     'cust_id'=>$cust_id,
                     'recur_amount'=>$recur_amount,
                     'pan'=>$pan,
                     'expdate'=>$expdate,
                     'p_account_number'=>$p_account_number,
                     'presentation_type'=>$presentation_type,
                     'add_num_recurs' => $add_num,
                     'total_num_recurs' => $total_num,
                     'hold' => $hold,
                     'terminate' => $terminate,
                     'avs_street_number' => $avs_street_number,
                     'avs_street_name' => $avs_street_name,
                     'avs_zipcode' => $avs_zipcode
                  );

/**************************** 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());

?>
```

20. 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 at https://developer.moneris.com

21. 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
monusqa002***	qatoken	demouser	abc1234
monusqa003	qatoken	demouser	abc1234
monusqa004	qatoken	demouser	abc1234
monusqa005	qatoken	demouser	abc1234
monusqa006	qatoken	demouser	abc1234
monusqa024 *	qatoken	demouser	abc1234
monusqa025 **	qatoken	demouser	abc1234

* test store 'monusqa024' is intended for testing ACH transactions only
** test store 'monusqa025' is intended for testing both ACH and Credit Card transactions
*** test store 'monusqa002' is intended for testing the Pinless Debit transactions
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 or 4005554444444403
Amex	373599005095005
Pinless Debit	4496270000164824

When testing ACH transactions you may use the following test bank account details.

Test Bank Account Details			
Financial Institution	**Routing Number**	**Account Number**	**Check Number**
FEDERAL RESERVE BANK	011000015	Any number between 5-22 digits	Any number

To To access the Merchant Resource Centre in the test environment go to https://esplusqa.moneris.com/usmpg. 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 credit card transactions based on the penny value of the amount field.
For example, a credit card transaction made for the amount of $9.00 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 $11.00. 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 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.

The test environment will approve/register all ACH transactions as long as there is no error with the format.
For example, if all of the ACH variables are properly named and populated, all transactions will approve/register. If there is a format violation, such as invalid data in one of the fields (ex. cust_zip requires 'MI' but 'Michigan' is sent) then the ACH transaction will decline/fail to register.

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.

22. 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 fields vary depending on what type of transaction was performed:

- Basic Transaction (Non Track2 / Card Not Present) – please refer to Appendix N. Basic Transaction Receipt (Non Track2)
- Mag Swipe Transaction (Track2 / Card Present) – please refer to Appendix O. Mag Swipe Transaction Receipt (Track2)

In addition, for non credit card transactions, such as Pinless Debit and ACH, there are certain fields that are recommended to be returned in a receipt of registration of the transaction.

- Pinless Debit – please refer to Appendix P. Pinless Debit Transaction Receipt
- ACH Transaction (Check not present) – please refer to Appendix Q. ACH Transaction
- ACH Transaction (Check physically present) – please refer to Appendix R. ACH Transaction Receipt (Check Physically Present)

For a breakdown of all required fields, as well as a sample of the receipt, please refer to the appropriate Appendix listed above.

23. How Do I Activate My Store?

Once you have received your activation letter/fax go to https://esplus.moneris.com/usmpg/activate/ 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 top of the page you will locate your production API Token.

24. How Do I Configure My Store For Production?

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

PRODUCTION

```
var $Globals=array(
    MONERIS_PROTOCOL => 'https',
    MONERIS_HOST => 'esplus.moneris.com',
    MONERIS_PORT =>'443',
    MONERIS_FILE => '/gateway_us/servlet/MpgRequest',
    API_VERSION  =>'MPG version 1.0',
    CLIENT_TIMEOUT => '60'
);
```

DEVELOPMENT

```
var $Globals=array(
    MONERIS_PROTOCOL => 'https',
    MONERIS_HOST => 'esplusqa.moneris.com',
    MONERIS_PORT =>'443',
    MONERIS_FILE => '/gateway_us/servlet/MpgRequest',
    API_VERSION  =>'MPG version 1.0',
    CLIENT_TIMEOUT => '60'
 );
```

Once you are in production you will access the Merchant Resource Centre at https://esplus.moneris.com/usmpg. 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 HELP button found in the top left corner of the website.

25. How Do I Get Help?

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

For financial support:
Phone: 1-800-471-9511
Email: supportinfo@moneris.com

For technical and integration support:
Phone: 1-866-696-0488
Email: eselectplus@moneris.com

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

26. 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. Characters allowed for Order ID: **a-z A-Z 0-9 _ - : . @ spaces**
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 *'expdate' is optional for Pinless Debit Purchase
amount	9 / decimal	Amount of the transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
crypt_type	1 / an	E-Commerce Indicator: 1 - Mail Order / Telephone Order - Single 2 - Mail Order / Telephone Order - Recurring 3 - Mail Order / Telephone Order - Instalment 4 - Mail Order / Telephone Order - Unknown Classification 5 - Authenticated E-commerce Transaction (VBV/MCSC) 6 – Non Authenticated E-commerce Transaction (VBV/MCSC) 7 - SSL enabled merchant 8 - Non Secure Transaction (Web or Email Based) 9 - SET non - Authenticated transaction
pos_code	2 / num	Under normal presentment situations the value should be '00'. In the case of a PreAuth/Completion, if the PreAuth was card present keyed-in then the 'pos_code' for the completion should be '71' (meaning that a 'us_track2_preauth' transaction was submitted where the 'pan' and 'expdate' variables were populated while the 'track2' was left blank). In an unmanned kiosk environment, where the card is present, the value should be '27'. If the solution is not "merchant and cardholder present" please call the support desk and we will provide the proper POS Code.
txn_number	255 / varchar	Used when performing follow on transactions - this must be filled with the value that was returned as the Txn_number in the response of the original transaction. When performing a Capture this must reference the PreAuth. When performing a Refund or a Void this must reference the Capture or the Purchase.
cust_id	50/an	This is an optional field that can be sent as part of a Purchase or PreAuth request. It is searchable from the Moneris Merchant Resource Centre. It is commonly used for policy number, membership number, student ID or invoice number.
track2		This is a string that is retrieved from the mag swipe of a credit card by swiping the credit card through a card reader. It is part of a mag swipe/track2 transaction.

enc_track2		This is a string that is retrieved by swiping or keying in a credit card through a Moneris provided encrypted mag swipe card reader. It is part of an encrypted keyed or swiped transaction only. This string must be retrieved by a specific device. Please refer to device_type for the list of current available devices.
device_type	an	Defines the encrypted mag swipe reader that was used for swiping the credit card. Plesase note, this device must be provided by Moneris Solutions so that the values are properly encrypted and decrypted. This field is case sensitive. Available values are: device_type="idtech"
cavv		This is a value that is provided by the Moneris MPI or by a third party MPI. It is part of a VBV/MCSC 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	9 / 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. Refer to Appendix I. Card Validation Digits (CVD) for further details. 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 Appendix I. Card Validation Digits (CVD) for values)
commcard_invoice	17 / an	Level 2 Invoice Number for the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). Characters allowed for commcard_invoice: **a-z A-Z 0-9 spaces**
commcard_tax_amount	9 / decimal	Level 2 Tax Amount of the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). This must contain 3 digits with two penny values. The minimum value passed can be 0.00 and the maximum is 9999999.99
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.
dynamic_descriptor	25 / an	Merchant defined description sent on a per-transaction basis that will appear on the credit card statement appended to the merchant's business name. Please note, the combined length of the merchant's business name and dynamic_descriptor may not exceed 25 characters.
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 (Check Appendix B. Definitions of Response Fields) ▪ If the transaction is not found then the gateway will respond with a not found message (Check Appendix B. Definitions of Response Fields) Once it is set to "false" the transaction will process as a new transaction

NOTE

The order_id allows the following characters: **a-z A-Z 0-9 _ - : . @ spaces**

The commcard_invoice allows the following characters: **a-z A-Z 0-9 spaces**

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

27. 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 "640123450010690030" is the reference number returned in the message, "64012345" 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 at https://developer.moneris.com
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
MaskedPan	####********#### ####*******####	Indicates the first 4 last 4 digits of the credit card number that was swiped or keyed in using the encrypted mag swipe reader so that it may be displayed on a receipt.
RecurSucess	true/false	Indicates whether the transaction successfully registered.
AvsResultCode	1/alpha	Indicates the address verification result. Refer to Appendix J. Address Verification Service (AVS).
CvdResultCode	2/an	Indicates the CVD validation result. Refer to Appendix I. Card Validation Digits (CVD).
RecurUpdateSuccess	true/false	Indicates whether the transaction successfully updated.
NextRecurDate	yyyy-mm-dd	Indicates when the transaction will be billed again (recur).
RecurEndDate	yyyy-mm-dd	Indicates when the Recurring Billing Transaction will end.
CardLevelResult	3/an	Please refer to Appendix L. Card Level Result Value for a list of all Visa and MasterCard Card Level Result values.

CavvResultCode	1 / an	The CAVV result code indicates the result of the CAVV validation. Note this is only applicable to Visa VBV transactions. 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; information only, no liability shift Please refer to Appendix M. CAVV Result Code 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
StatusMessage	found/not found	The StatusMessage is populated when status_check is set to “true” in the request

28. 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 8 - Purchase (with Customer and Order details)for an example.		
first_name	30 / an	
last_name	30 / an	
company_name	30 / an	
address	30 / an	
city	30 / an	
province	30 / an	
postal_code	30 / an	
country	30 / an	
phone	30 / an	
fax	30 / an	
tax1	30 / an	
tax2	30 / an	
tax3	30 / an	
shipping_cost	30 / an	
Item Information		
NOTE: You may send multiple items. Please refer to section 8 - Purchase (with Customer and Order details) for an example.		
item_description	30 / an	
item_quantity	10 / num	You must send a quanitity > 0 or the item will not be added to the item list (ie. minimum 1, maximum 9999999999)
item_product_code	30 / 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	50 / an	
instructions	50 / an	

NOTE

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

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

Also, the data sent in Billing and Shipping Address fields will not be used for any address verification. Please refer to the section 8 – Purchase (with CVD and AVS - eFraud).

29. Appendix D. Recur and Recur Update Fields

Recur Request Fields

Variable Name	Size/Type	Description
recur_unit	day, week, month, eom	The unit that you wish to use as a basis for the Interval. This can be set as day, week, month or end of 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 = 45, recur_unit=day -> Card will be billed every 45 days. period = 4, recur_unit=weeks -> Card will be billed every 4 weeks. period = 3, recur_unit=month -> Card will be billed every 3 months. period = 3, recur_unit=eom -> Card will be billed every 3 months (on the last day of the month) 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 Examples	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);

$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);

$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. 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.

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 and Pinless Debit transactions.
expdate	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 bank 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. **PLEASE NOTE TERMINATED RECURRING TRANSACTION CAN NO LONGER BE REACTIVATED.**

NOTE

When completing the Recur Update portion please keep in mind that the profile cannot be changed to have a new end date greater than 10 years from today. Also the new end date cannot be today or earlier.

Once a Recurring Billing profile has been terminated it 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

30. Appendix E. Pinless Debit Fields

Pinless Debit Request Fields		
Variable Name	**Size/Type**	**Description**
presentation_type	1 / alpha	Identifies how merchants obtain the Pinless Debit account. - 'X' for Telephone/VRU - 'W' for Internet
intended_use	1 / num	Identifies the party who initiated the transaction. - "0" = Merchant initiated the payment - "1" = Customer initiated the payment
p_account_number	25 / num	The billing invoice number – no spaces or dashes. The length of the account number varies with a maximum length of 25 digits.
Pinless Debit Customer Information		
NOTE: The following Account Holder information fields are optional.		
first_name	50 / an	
last_name	50 / an	
address	50 / an	
address2	50 / an	
city	50 / an	
state	2 / alpha	The state must be submitted as exactly 2 characters (ex. MI – Michigan)
zip_code	15 / an	

NOTE

If you send characters that are not included in the allowed list, the Pinless Debit transaction may not be properly registered.

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

Also, the data sent in the Pinless Debit Customer Information fields will not be used for any address verification.

31. Appendix F. AchInfo Fields

AchInfo Request Fields		
Variable Name	**Size/Type**	**Description**
sec	3 / an	ACH sec Code: The following sec codes apply only if check not physically present. ppd - Prearranged Payment and Deposit ccd - Cash Concentration or Disbursement web - Internet Initiated Entry The following SEC codes apply only if ckeck present. pop – Point of Sale Purchase boc – Back Office Conversion arc – Account Receivable Conversion Please refer to Appendix G. ACH Sec Codes and Process Flow for full sec code description
routing_num	9 / num	The first number in the MICR, or magnetic ink character recognition, line at the bottom of a check is the bank's check routing number. It is exactly nine digits long and always starts with 0, 1, 2 or 3.
account_num	50 / num	The account number may appear before or after the check number in the check's MICR line at the bottom of the check. The length of the account number varies with a maximum length of 50 digits.
check_num	16 / num	The sequential number for checks appears in both the MICR line at the bottom of the check and the upper right corner of the check. The check number length may vary; the maximum length is 16 digits. This is an optional field.
account_type	savings / checking	Identifies the type of bank account. The account type must be submitted as either ‘savings’ or ‘checking’. This field is case sensitive.
micr	200 / alpha	The check’s raw MICR number obtained from the scanner. Do not modify the micr data value after it has already been scanned. e.g. micr = "t071000013t742941347o129";
dl_num	an	The first two characters of this field should be the State Code of the Driver’s License, followed by a dash (ASCII 45), then the ID Data. e.g. Colorado: dl_num = “CO-12312312";
magstripe	yes / no	Fixed value. The magstripe data is obtained when scanning the Driver’s License.
image_front	an	The front image of the check obtained from the scanner, base64 encoded.
image_back	an	The back image of the check obtained from the scanner, base64 encoded.
ACH Customer Information		
NOTE: The following Account Holder information fields are optional.		
cust_first_name	50 / an	
cust_last_name	50 / an	
cust_address1	50 / an	
cust_address2	50 / an	
cust_city	50 / an	
cust_state	2 / alpha	The state must be submitted as exactly 2 characters (ex. MI – Michigan)
cust_zip	15 / an	

NOTE

If you send characters that are not included in the allowed list, the ACH transaction may not be properly registered.

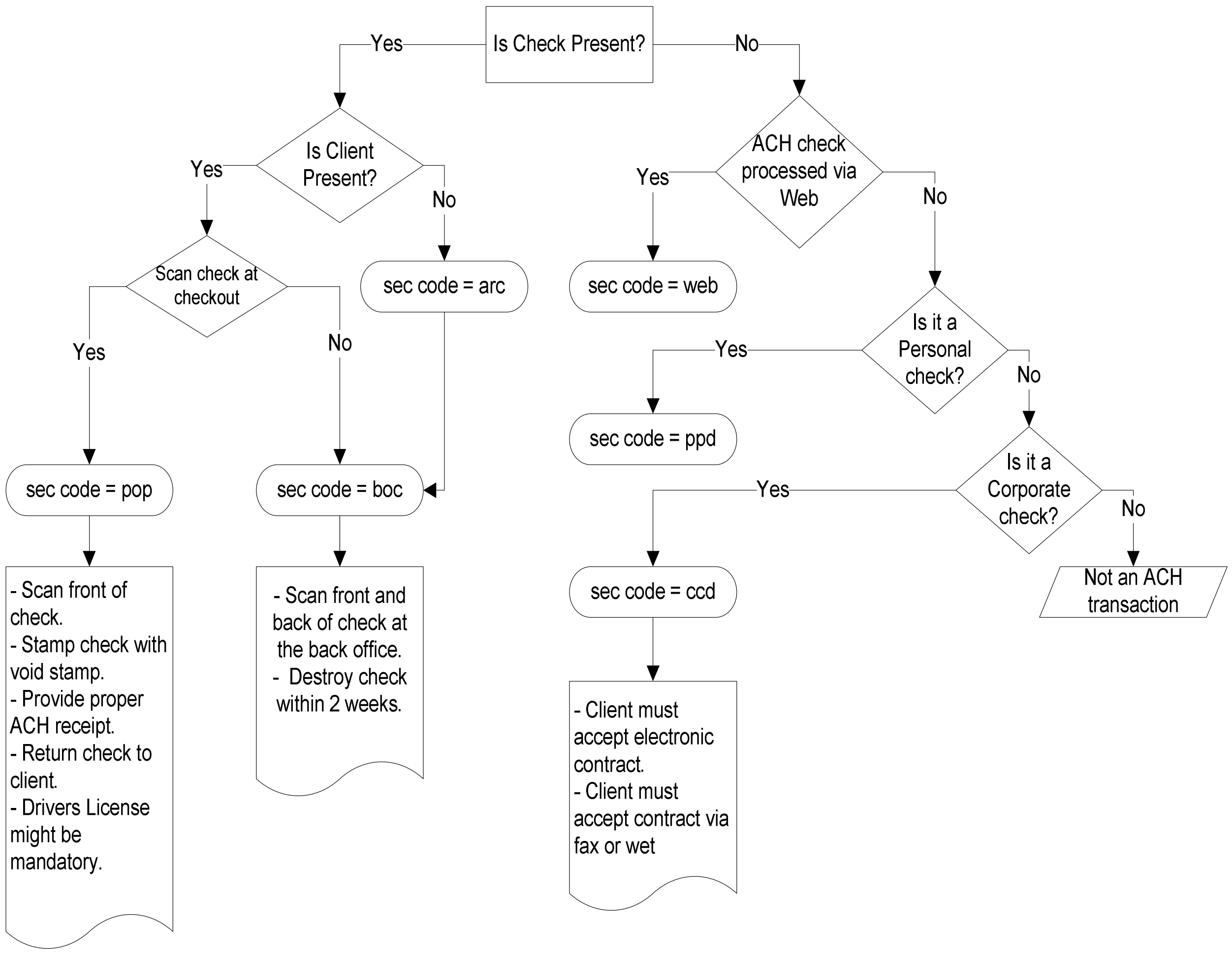
All alphanumeric fields allow the following characters: **a-z A-Z 0-9 _ - : . @ $ =** /

Also, the data sent in the ACH Customer Information fields will not be used for any address verification.

32. Appendix G. ACH Sec Codes and Process Flow

ACH Sec Codes		
Variable Name	**Size/Type**	**Description**
sec	3 / an	*The following SEC codes apply only if check is not physically present:* **PPD** – (Prearranged Payment and Deposit) Debit (Sale): Consumer grants the merchant the right to initiate a one time or recurring charge(s) to his or her account as bills become due. Credit (Refund): Transfers funds into a consumer's bank account. The funds being deposited can represent a variety of financial transactions, such as payroll, interest, pension, dends, etc. **CCD** – (Cash Concentration or Disbursement) Debit (Sale): Client grants the merchant the right to initiate a one time or recurring charge(s) to a business bank account. Credit (Refund): Transfer funds to a client's business bank account. **WEB** – (Internet Initiated Entry) Debit (Sale): A Debit entry to a consumer's bank account initiated by a merchant. The consumer's authorization is obtained via the Internet. Credit (Refund): N/A. *The following SEC codes apply only if ckeck is present:* **POP** – (Point of Sale Purchase) Client presents check to the merchant at time of purchase. **BOC** – (Back Office Conversion) Client presents check to the merchant at time of purchase and check is converted to electronic format. **ARC** – (Account Receivable Conversion) Client's check is received through mail and processed by merchant. * only 'ppd' and 'ccd' apply to the ACH Credit transaction

Process Flow for ACH Transactions


Is Check Present?
Yes
No
Is Client Present?
Yes
No
ACH check processed via Web
Yes
No
Scan check at checkout
Yes
No
sec code = arc
sec code = web
Is it a Personal check?
Yes
No
sec code = ppd
sec code = pop
sec code = boc
Is it a Corporate check?
Yes
No
sec code = ccd
Not an ACH transaction
- Scan front of check.
- Stamp check with void stamp.
- Provide proper ACH receipt.
- Return check to client.
- Drivers License might be mandatory.
- Scan front and back of check at the back office.
- Destroy check within 2 weeks.
- Client must accept electronic contract.
- Client must accept contract via fax or wet

33. Appendix H. 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

34. Appendix I. 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. The CvdInfo parameter is broken down into two elements. The first element is the CVD Value itself.

The second element is the CVD Indicator. This value indicates the possible scenarios when collecting CVD information. This is a 1 digit value which can have any of the following values:

CVD INDICATOR	
VALUE	**DEFINITION**
0	CVD value is deliberately bypassed or is not provided by the merchant.
1	CVD value is present.
2	CVD value is on the card, but is illegible.
9	Cardholder states that the card has no CVD imprint.

CVD Response codes:

The CVD response is an alphanumeric 2 byte variable. The first byte is the numeric CVD indicator sent in the request; the second byte would be the response code. The following is a list of all possible responses once a CVD value has been passed in.

CVD RESPONSE CODES	
RESULT VALUE	**DEFINITION**
M	Match
Y	Match for AmEx
N	No Match
P	Not Processed
S	CVD should be on the card, but Merchant has indicated that CVD is not present
R	Retry for AmEx
U	Issuer is not a CVD participant
Other	Invalid Response Code

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.

***For additional information on how to handle these responses, please refer to Appendix K. Additional Information for CVD and AVS.**

35. Appendix J. 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. AvsInfo is broken down into three elements:

Element	Type	Length
Street Number	Numeric	19 characters combined.
Street Name	Alphanumeric	
Zip/Postal Code	Alphanumeric	9 characters

The following table outlines the possible responses when passing in AVS information.

AVS RESPONSE CODES		
VALUE	**VISA/DISCOVER / JCB**	**MASTERCARD**
A	Address matches, ZIP does not. Acquirer rights not implied.	Address matches, zip code does not.
B	Street addresses match. Zip code not verified due to incompatible formats. (Acquirer sent both street address and zip code.)	N/A
C	Street addresses not verified due to incompatible formats. (Acquirer sent both street address and zip code.)	N/A
D	Street addresses and zip codes match.	N/A
F	Street address and zip code match. Applies to U.K. only	N/A
G	Address information not verified for international transaction. Issuer is not an AVS participant, or AVS data was present in the request but issuer did not return an AVS result, or Visa performs AVS on behalf of the issuer and there was no address record on file for this account.	N/A
I	Address information not verified.	N/A
K	N/A	N/A
L	N/A	N/A
M	Street address and zip code match.	N/A
N	No match. Acquirer sent postal/ZIP code only, or street address only, or both zip code and street address. Also used when acquirer requests AVS but sends no AVS data.	Neither address nor zip code matches.
O	N/A	N/A
P	Zip code match. Acquirer sent both zip code and street address but street address not verified due to incompatible formats.	N/A
R	Retry: system unavailable or timed out. Issuer ordinarily performs AVS but was unavailable. The code R is used by Visa when issuers are unavailable. Issuers should refrain from using this code.	Retry; system unable to process.
S	N/A	AVS currently not supported.
U	Address not verified for domestic transaction. Issuer is not an AVS participant, or AVS data was present in the request but issuer did not return an AVS result, or Visa performs AVS on behalf of the issuer and there was no address record on file for this account.	No data from Issuer/Authorization system.
W	Not applicable. If present, replaced with 'Z' by Visa. Available for U.S. issuers only.	For U.S. Addresses, nine-digit zip code matches, address does not; for address outside the U.S. postal code matches, address does not.
X	N/A	For U.S. addresses, nine-digit zip code and addresses matches; for addresses outside the U.S., postal code and address match.
Y	Street address and zip code match.	For U.S. addresses, five-digit zip code and address matches.
Z	Postal/Zip matches; street address does not match or street address not included in request.	For U.S. addresses, five digit zip code matches, address does not.

VALUE	AMERICAN EXPRESS
A	Billing address matches, zip code does not
D	Customer name incorrect, zip code matches
E	Customer name incorrect, billing address and zip code match
F	Customer name incorrect, billing address matches
K	Customer name matches
L	Customer name and zip code match
M	Customer name, billing address, and zip code match
N	Billing address and zip code do not match
O	Customer name and billing address match
R	System unavailable; retry
S	AVS not currently supported
U	Information is unavailable
W	Customer name, billing address, and zip code are all incorrect
Y	Billing address and zip code both match
Z	Zip code matches, billing address does not

36. Appendix K. 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 completion 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 and AVS verification is only applicable towards Visa, MasterCard, Discover, JCB and American Express transactions.

37. Appendix L. Card Level Result Value

The Card Level Result value refers to the issuer-supplied data on file in the Cardholder Database. Visa and MasterCard will populate this field with an appropriate product identification value which can be used to track card-level activity by an individual account number. These details will be populated within the CardLevelResult response field when returned by the associations.

The following table outlines the possible Card Level Result responses.

CARD LEVEL RESULT VALUE			
VALUE	**VISA**	**VALUE**	**VISA**
A	Visa Classic/Traditional	L	Electron
AX	American Express	M	MasterCard/Euro Card and Diners
B	Visa Gold/Platinum/Traditional Rewards	N	Visa Platinum
C	Visa Signature	N1	TBA
D	Visa Signature Preferred/Visa Infinite	O	Reserved
DI	Discover	P	Visa Gold
E	Reserved	Q	Private Label
F	Visa Classic	Q1	Private Label Prepaid
G	Visa Business	R	Proprietary
G1	Visa Signature Business	S	Visa Purchasing
G2	Visa Business Check Card	S1	Visa Purchasing with Fleet
G3	Visa Enhanced Business	S2	Visa GSA Purchasing
H	Visa Check Card/Debit	S3	Visa GSA Purchasing with Fleet
I	Visa Infinite	S4	Commercial Business Loan
J	Reserved	S5	Commercial Transport EBT
J1	Visa General Prepaid	S6	Business Loan
J2	Visa Prepaid Gift	S7	Visa Distribution
J3	Visa Prepaid Healthcare	T	Reserved/Interlink
J4	Visa Prepaid Commercial	U	Visa Travel Money
K	Visa Corporate	V	Reserved
K1	Visa GSA Corporate T&E		
VALUE	**MASTERCARD**	**VALUE**	**MASTERCARD**
CIR	Cirrus	MPZ	MasterCard Prepaid Debit Standard- Gov. Consumer
DAG	Gold Debit MasterCard Salary	MRC	Electronic Consumer Pre-Paid (Non U.S.)
DAP	Platinum Debit MasterCard Salary	MRF	Standard Deferred
DAS	Standard Debit MasterCard Salary	MRG	Standard Pre-Paid (Non U.S.)
DLG	Debit Gold - Delayed Debit	MRH	Platinum Prepaid Travel Card
DLH	Debit World Embossed - Delayed Debit	MRJ	Pre-Paid Gold Card
DLP	Debit Platinum - Delayed Debit	MRK	Pre-Paid Public Sector Commercial Card
DLS	Debit Standard - Delayed Debit	MRO	MasterCard Rewards Only
DOS	Standard Debit MasterCard Social	MRP	Standard Retailer Centric Payments
MAB	World Elite For Business	MRW	Prepaid Business Card (Non U.S.)
MAC	Corporate World Elite	MSA	Pre-Paid Maestro Payroll Card
MAV	MasterCard Activation Verification	MSB	Maestro Small Business Card
MBD	MasterCard Professional Debit Business Card	MSF	Pre-Paid Maestro Gift Card
MBE	Electronic Business Card	MSG	Pre-Paid Maestro Consumer Reloadable Card
MBK	Black Card	MSI	Maestro
MBP	MasterCard Corporate Prepaid	MSJ	Prepaid Maestro Gold
MBT	MasterCard Corporate Prepaid Travel	MSM	Pre-Paid Maestro Consumer Promotion Card
MCB	BusinessCard Card	MSN	Pre-Paid Maestro Insurance Card
MCC	Credit (Mixed BIN)	MSO	Pre-Paid Maestro Other Card
MCD	Debit MasterCard	MSQ	Reserved
MCE	Electronic Card	MSR	Pre-Paid Maestro Travel Card
MCF	Fleet Card	MST	Pre-Paid Maestro Teen Card
MCG	Gold Card	MSV	Pre-Paid Maestro Government Benefit Card
MCH	MasterCard Premium Charge	MSW	Pre-Paid Maestro Corporate Card
MCO	Global Certified Corporate Card	MSX	Pre-Paid Maestro Flex Benefit Card
MCP	Purchasing Card	MSY	Pre-Paid Maestro Employee Incentive Card
MCS	Standard Card	MSZ	Pre-Paid Maestro Emergency Assistance Card
MCT	Titanium Card	MUW	World Domestic Affluent
MCV	Merchant Branded Program	MWB	World MasterCard For Business
MCW	World MasterCard	MWD	World Deferred
MDB	Debit MasterCard Business Card	MWE	World Elite Card
MDG	Gold Debit MasterCard	MWO	World Elite Corporate Card

MDH	World Debit Card	MWR	World Retailer Centric Payments
MDJ	Debit World Elite	OLB	Maestro Small Business - Delayed Debit
MDL	Business Debit Other Embossed	OLG	Maestro Gold - Delayed Debit
MDO	Debit Other	OLP	Maestro Platinum - Delayed Debit
MDP	Debit MasterCard Platinum	OLS	Maestro - Delayed Debit
MDR	Debit Brokerage	OLW	World Maestro - Delayed Debit
MDS	Debit MasterCard	PMC	Proprietary Credit Card (Sweden)
MDT	Commercial Debit Card	PMD	Proprietary Debit Card (Sweden)
MEC	Electronic Commercial	PSC	Common Proprietary Credit Card (Sweden)
MEF	Electronic Payment Account	PSD	Common Proprietary Debit Card (Sweden)
MFB	Flex World Elite	PVA	Private Label A
MFD	Flex Platinum	PVB	Private Label B
MFE	Flex Charge World	PVC	Private Label C
MFH	Flex World	PVD	Private Label D
MFL	Flex Charge Platinum	PVE	Private Label E
MFW	Flex Charge World	PVF	Private Label F
MGF	Government Commercial Card	PVG	Private Label G
MHA	MasterCard Healthcare Prepaid (Non Tax)	PVH	Private Label H
MIA	Prepaid MasterCard Unembossed Student Card	PVI	Private Label I
MIP	Prepaid MasterCard Student Card	PVJ	Private Label J
MIU	Debit MasterCard Unembossed (Non US)	PVL	Private Label L
MNF	Public Sector Commercial Card	SAG	Gold MasterCard Debit - Immediate Debit
MNW	New World	SAL	Standard Maestro Salary
MOC	Standard Maestro Social	SAP	Platinum MasterCard Salary - Immediate Debit
MOG	Maestro Gold	SAS	Standard MasterCard Salary - Immediate Debit
MOP	Maestro Platinum	SOL	UK Domestic Solo Brand
MOW	World Maestro	SOS	Standard MasterCard Social - Immediate Debit
MPA	Prepaid Debit Standard-Payroll	SWI	UK Domestic Switch Brand
MPB	Preferred Business Card	TBE	Electronic Business - Immediate Debit
MPF	Prepaid Debit Standard- Gift	TCB	Business Card - Immediate Debit
MPG	Debit Standard Prepaid - General Spend	TCC	Mixed Bin - Immediate Debit
MPH	MasterCard Cash	TCE	Electronic - Immediate Debit
MPJ	Prepaid Debit Card Gold	TCF	Fleet Card - Immediate Debit
MPK	Prepaid Government Commercial Card	TCG	Gold Card - Immediate Debit
MPL	Platinum Card	TCO	Corporate - Immediate Debit
MPM	MC Prepaid Debit Standard- Consumer Incentive	TCP	Purchasing Card - Immediate Debit
MPN	MC Prepaid Debit Standard- Insurance	TCS	Standard Card - Immediate Debit
MPO	MC Prepaid Debit Standard- Other	TCW	World Signia Card - Immediate Debit
MPP	Prepaid Card	TEC	Electronic Commercial - Immediate Debit
MPR	MC Prepaid Debit Standard- Travel	TNF	Public Sector Commercial Card - Immediate Debit
MPT	MC Prepaid Debit Standard- Teen	TNW	New World - Immediate Debit
MPV	MC Prepaid Debit Standard- Government	TPB	Preferred Business Card - Immediate Debit
MPW	Debit MC Business Prepaid Business To Business	TPL	Platinum - Immediate Debit
MPX	MasterCard Prepaid Debit Standard- Flex Benefit	VIS	VisaNet
MPY	MasterCard Prepaid Debit Standard - Employee	WBE	World MasterCard Black Edition

38. Appendix M. 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 'us_cavv_purchase' or 'us_cavv_preauth' transaction type.
For more information on sending VBV/SecureCode transactions, please refer to our "Moneris MPI - Verified By Visa / MasterCard SecureCode PHP API" document.

NOTE Please note that the CAVV Result Code is only applicable towards Visa transactions.

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

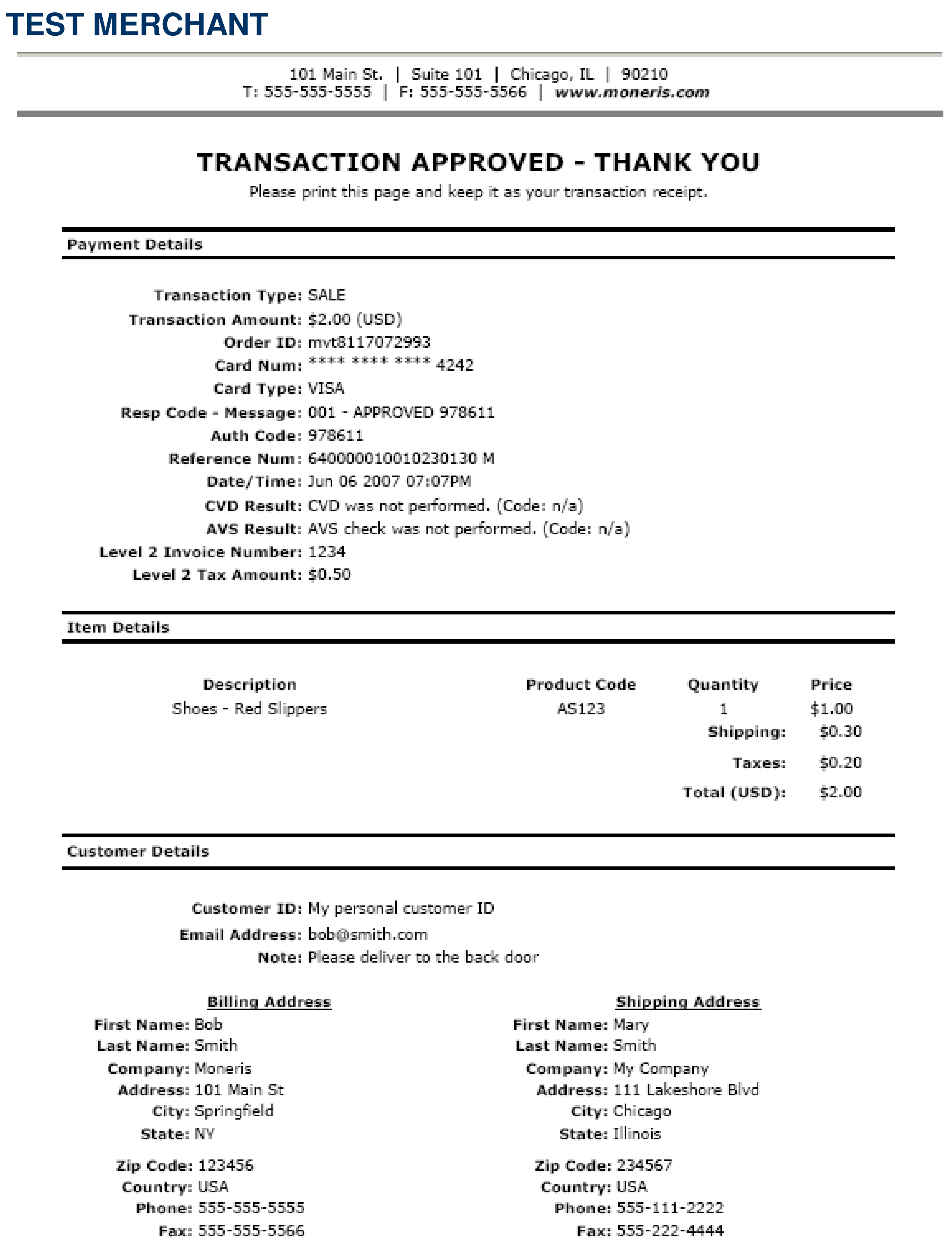
Table of CAVV result codes

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; information only, no liability shift	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.

39. Appendix N. Basic Transaction Receipt (Non Track2)

For all regular credit card transactions (card not present), the credit card associations expect certain fields to be presented to the cardholder on a receipt.

	Field	Description
1	Merchant Name	The name of the store / business.
2	Merchant URL	Web site address of the store / business.
3	Transaction Type	The type of transaction that was performed: - Sale (Purchase) - Offline Sale (Force Post) - Authorization (PreAuth) - Authorization Completion (Completion) - Sale Void (Correction / Purchase Correction) - Refund *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Transaction Amount	The total amount being paid by credit card.
5	AVS / CVD Result	The result for AVS and CVD verifications are one alpha character. This character will indicate if the verification was performed or not by the merchant or the Card Associations.
6	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
7	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
8	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined, the title is printed but the field is blank.
9	Response Code	The 3 digit response code returned by the issuer (ex. 000 – 999)
10	Response Message	Message indicating whether the transaction was Approved or Declined.
11	Cardholder Name	Display both First and Last Name as submitted by the cardholder.
12	Goods and Services Order	A list of all items/services that are being paid for in this transaction.
13	Return Policy	The refund policy is only a requirement for e-commerce transactions.



TEST MERCHANT

101 Main St. | Suite 101 | Chicago, IL | 90210
T: 555-555-5555 | F: 555-555-5566 | ***www.moneris.com***

TRANSACTION APPROVED - THANK YOU

Please print this page and keep it as your transaction receipt.

Payment Details

Transaction Type:	SALE
Transaction Amount:	$2.00 (USD)
Order ID:	mvt8117072993
Card Num:	**** **** **** 4242
Card Type:	VISA
Resp Code - Message:	001 - APPROVED 978611
Auth Code:	978611
Reference Num:	640000010010230130 M
Date/Time:	Jun 06 2007 07:07PM
CVD Result:	CVD was not performed. (Code: n/a)
AVS Result:	AVS check was not performed. (Code: n/a)
Level 2 Invoice Number:	1234
Level 2 Tax Amount:	$0.50

Item Details

Description	**Product Code**	**Quantity**	**Price**
Shoes - Red Slippers	AS123	1	$1.00
		Shipping:	$0.30
		Taxes:	$0.20
		Total (USD):	$2.00

Customer Details

Customer ID:	My personal customer ID
Email Address:	bob@smith.com
Note:	Please deliver to the back door

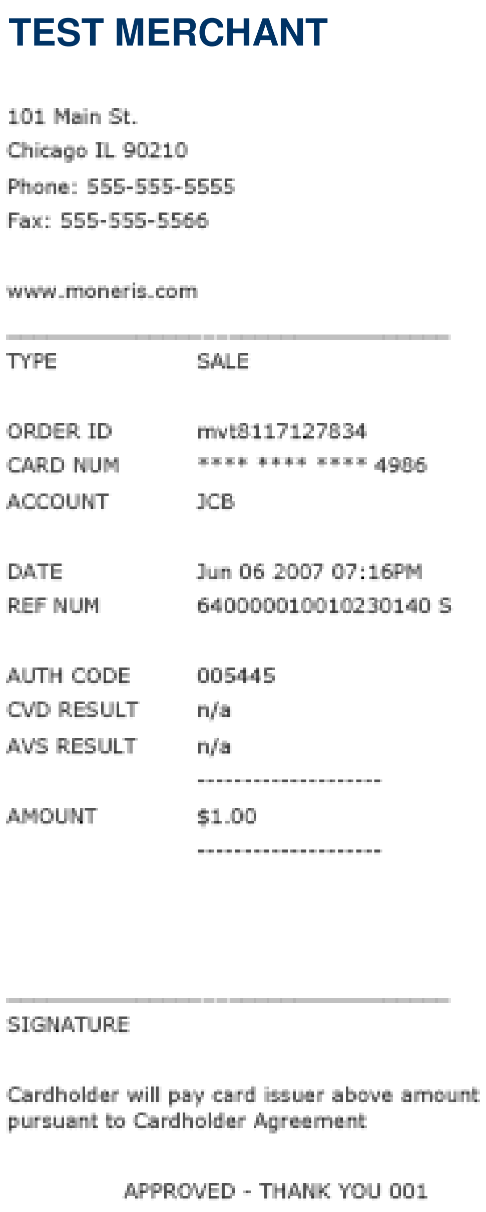
	Billing Address		**Shipping Address**
First Name:	Bob	**First Name:**	Mary
Last Name:	Smith	**Last Name:**	Smith
Company:	Moneris	**Company:**	My Company
Address:	101 Main St	**Address:**	111 Lakeshore Blvd
City:	Springfield	**City:**	Chicago
State:	NY	**State:**	Illinois
Zip Code:	123456	**Zip Code:**	234567
Country:	USA	**Country:**	USA
Phone:	555-555-5555	**Phone:**	555-111-2222
Fax:	555-555-5566	**Fax:**	555-222-4444

40. Appendix O. Mag Swipe Transaction Receipt (Track2)

For all mag swipe (card present) transactions, the credit card associations expect certain mandatory fields to be presented to the cardholder on a receipt.

	Field	Description
1	Merchant Name	The name of the store / business.
2	Store Address	The civic address of the store / business, which must include the street, town/city, state, and ZIP code.
3	Transaction Type	The type of transaction that was performed: - Sale (Mag Swipe Purchase) - Offline Sale (Mag Swipe Force Post) - Authorization (Mag Swipe PreAuth) - Authorization Completion (Mag Swipe Completion) - Sale Void (Mag Swipe Correction / Purchase Correction) - Refund (Mag Swipe Refund) *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Account Type	The type of credit card: VISA, MC, AMEX, etc.
5	Transaction Amount	The total amount being paid by credit card.
6	AVS / CVD Result	The result for AVS and CVD verifications are one alpha character. This character will indicate if the verification was performed or not by the merchant or the Card Associations.
7	Primary Account Number (PAN)	Cardholder's credit card number. The customer's copy of the receipt must have all but the last 4 digits of PAN masked out.
8	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
9	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
10	Card Entry Indicator	Credit cards can be manually keyed or swiped; if manual, the indicator is "M"; if swiped, the indicator is "S"
11	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined, the title is printed but the field is blank.
12	Response Code	The 3 digit response code returned by the issuer (ex. 000 – 999)

	Field	Description
13	Response Message	Message indicating whether the transaction was Approved or Declined. **Format:** message rrr (where message = defined below, rrr = response code) **Message Definition:** If the Response Code is between 00 and 49 (inclusive) (‘0’ <= Response Code =< ‘49’) Message = *“APPROVED - THANK YOU”* Any other response code (including ‘null’ and empty) Message = *“TRANSACTION NOT APPROVED”*
14	Signature	The signature forms the cardholder’s authority for the Sale transaction. **NOTE: Only the merchant’s copy requires the cardholder’s signature.** For Refund and Sale Void transactions, the merchant must sign the Cardholder’s copy of the cardholder receipt.
15	Cardholder Agreement	This text is required on cardholder transaction receipts for the following types of transactions: *Cardholder will pay card issuer above amount pursuant to Cardholder Agreement* - Sale - Offline Sale - Authorization - Authorization Completion This text is NOT required on cardholder transaction receipts for the following types of transactions: - Refund - Sale Void
16		Customer Copy Or Merchant Copy



TEST MERCHANT

101 Main St.
Chicago IL 90210
Phone: 555-555-5555
Fax: 555-555-5566

www.moneris.com

TYPE	SALE
ORDER ID	mvt8117127834
CARD NUM	**** **** **** 4986
ACCOUNT	JCB
DATE	Jun 06 2007 07:16PM
REF NUM	640000010010230140 S
AUTH CODE	005445
CVD RESULT	n/a
AVS RESULT	n/a

AMOUNT	$1.00

SIGNATURE

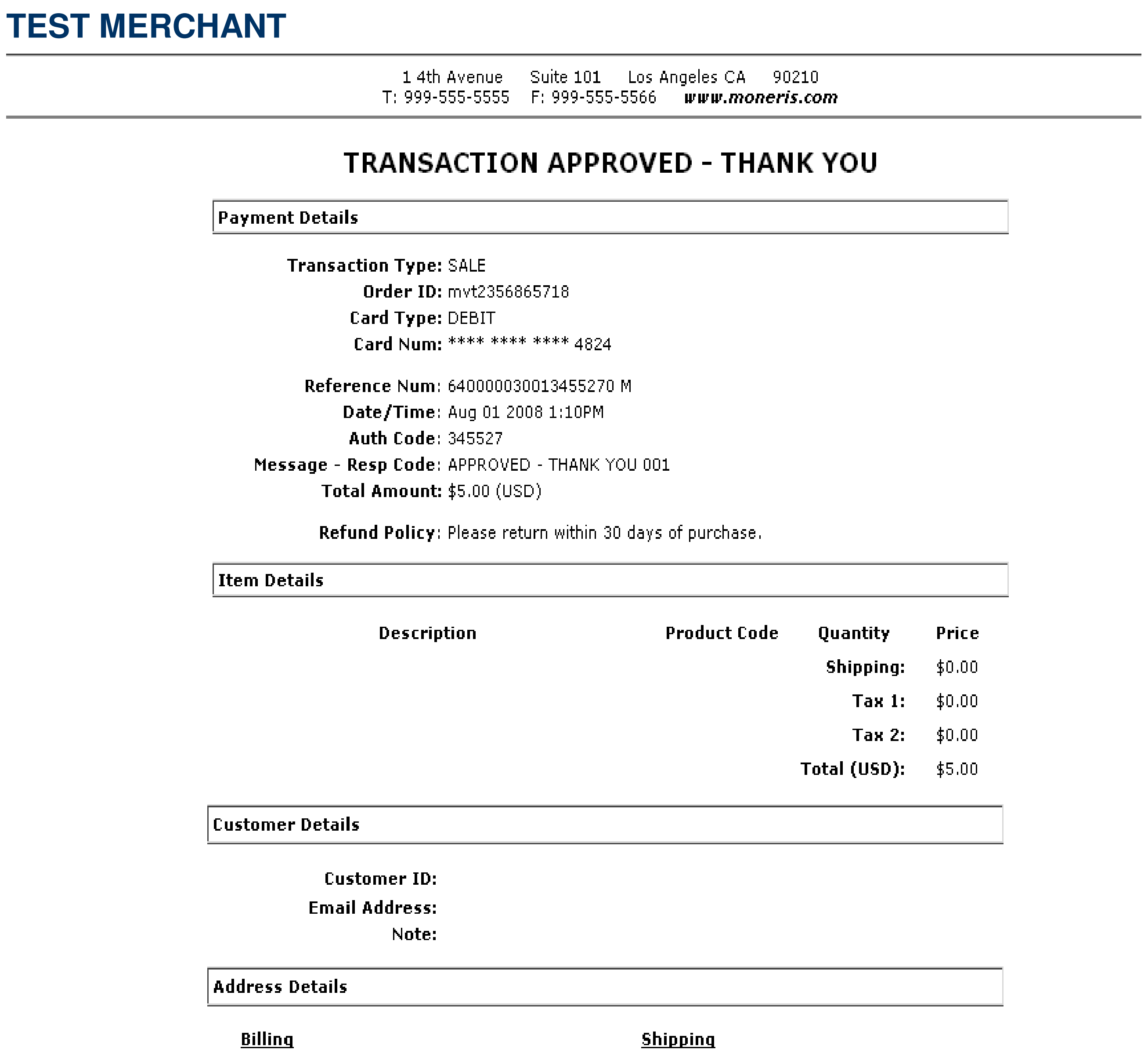
Cardholder will pay card issuer above amount
pursuant to Cardholder Agreement

APPROVED - THANK YOU 001

41. Appendix P. Pinless Debit Transaction Receipt

For all Pinless Debit transactions the credit card associations expect certain fields to be presented to the cardholder on a receipt

	Field	Description
1	Merchant Name	The name of the store / business.
2	Merchant URL	Web site address of the store / business.
3	Transaction Type	The type of transaction that was performed: - Sale (Purchase) - Refund *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Transaction Amount	The total amount being paid by Pinless Debit.
5	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
6	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
7	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined, the title is printed but the field is blank.
8	Response Code	The 3 digit response code returned by the issuer (ex. 000 – 999)
9	Response Message	Message indicating whether the transaction was Approved or Declined.
10	Cardholder Name	Display both First and Last Name as submitted by the cardholder.
11	Goods and Services Order	A list of all items/services that are being paid for in this transaction.
12	Return Policy	The refund policy is only a requirement for e-commerce transactions.



TEST MERCHANT

1 4th Avenue Suite 101 Los Angeles CA 90210
T: 999-555-5555 F: 999-555-5566 ***www.moneris.com***

TRANSACTION APPROVED - THANK YOU

Payment Details

Transaction Type:	SALE
Order ID:	mvt2356865718
Card Type:	DEBIT
Card Num:	**** **** **** 4824
Reference Num:	640000030013455270 M
Date/Time:	Aug 01 2008 1:10PM
Auth Code:	345527
Message - Resp Code:	APPROVED - THANK YOU 001
Total Amount:	$5.00 (USD)
Refund Policy:	Please return within 30 days of purchase.

Item Details

Description	Product Code	Quantity	Price
		Shipping:	$0.00
		Tax 1:	$0.00
		Tax 2:	$0.00
		Total (USD):	$5.00

Customer Details

Customer ID:
Email Address:
Note:

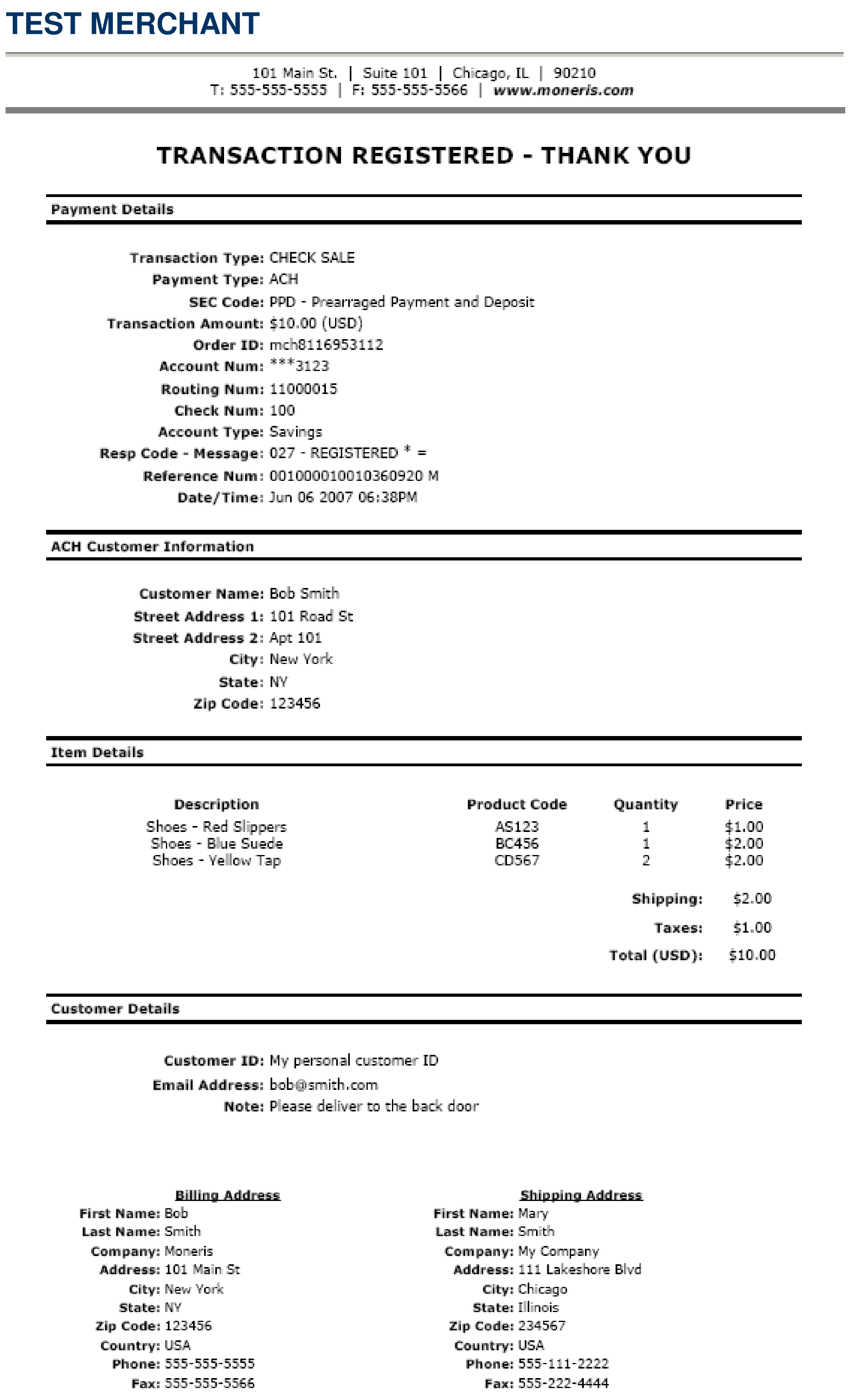
Address Details

Billing **Shipping**

42. Appendix Q. ACH Transaction Receipt (Check Not Present)

For an ACH transaction, a transaction confirmation is not mandatory; though eSELECTplus does recommend that a receipt of registration of the transaction be provided to the customer. Below is a list of recommended fields and the format they are to be displayed in.

	Field	Description
1	Merchant Name	The name of the store / business.
2	Merchant URL	Web site address of the store / business.
3	Transaction Type	The type of transaction that was performed: - Check Sale (ACH Debit) - Check Refund (ACH Credit) - Check Reversal (ACH Reversal) *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Payment Type	Indicate that this is an ACH processed transaction.
5	SEC Code	Specify which SEC Code was submitted. Identifies how the bank account information was collected. i.e. WEB – Internet Initiated Entry
6	Transaction Amount	The total amount being debited or credited to the bank account.
7	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
8	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
9	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined the field may be omitted.
10	Response Code	The 3 digit response code returned in the transaction (ex. 000 – 999)
11	Response Message / Result	Message indicating whether the transaction was Registered or Failed to Register.
12	Account Number	Customer's bank account number. All but the last 4 digits of the account number must be masked out.
13	Routing Number	Check routing number to identify the Financial Institution.
14	Check Number	Used for check tracking purposes.
15	Account Type	Indicate whether this is a Savings or Checking account.



TEST MERCHANT

101 Main St. | Suite 101 | Chicago, IL | 90210
T: 555-555-5555 | F: 555-555-5566 | ***www.moneris.com***

TRANSACTION REGISTERED - THANK YOU

Payment Details

Transaction Type:	CHECK SALE
Payment Type:	ACH
SEC Code:	PPD - Prearraged Payment and Deposit
Transaction Amount:	$10.00 (USD)
Order ID:	mch8116953112
Account Num:	***3123
Routing Num:	11000015
Check Num:	100
Account Type:	Savings
Resp Code - Message:	027 - REGISTERED * =
Reference Num:	001000010010360920 M
Date/Time:	Jun 06 2007 06:38PM

ACH Customer Information

Customer Name:	Bob Smith
Street Address 1:	101 Road St
Street Address 2:	Apt 101
City:	New York
State:	NY
Zip Code:	123456

Item Details

Description	Product Code	Quantity	Price
Shoes - Red Slippers	AS123	1	$1.00
Shoes - Blue Suede	BC456	1	$2.00
Shoes - Yellow Tap	CD567	2	$2.00

Shipping:	$2.00
Taxes:	$1.00
Total (USD):	$10.00

Customer Details

Customer ID:	My personal customer ID
Email Address:	bob@smith.com
Note:	Please deliver to the back door

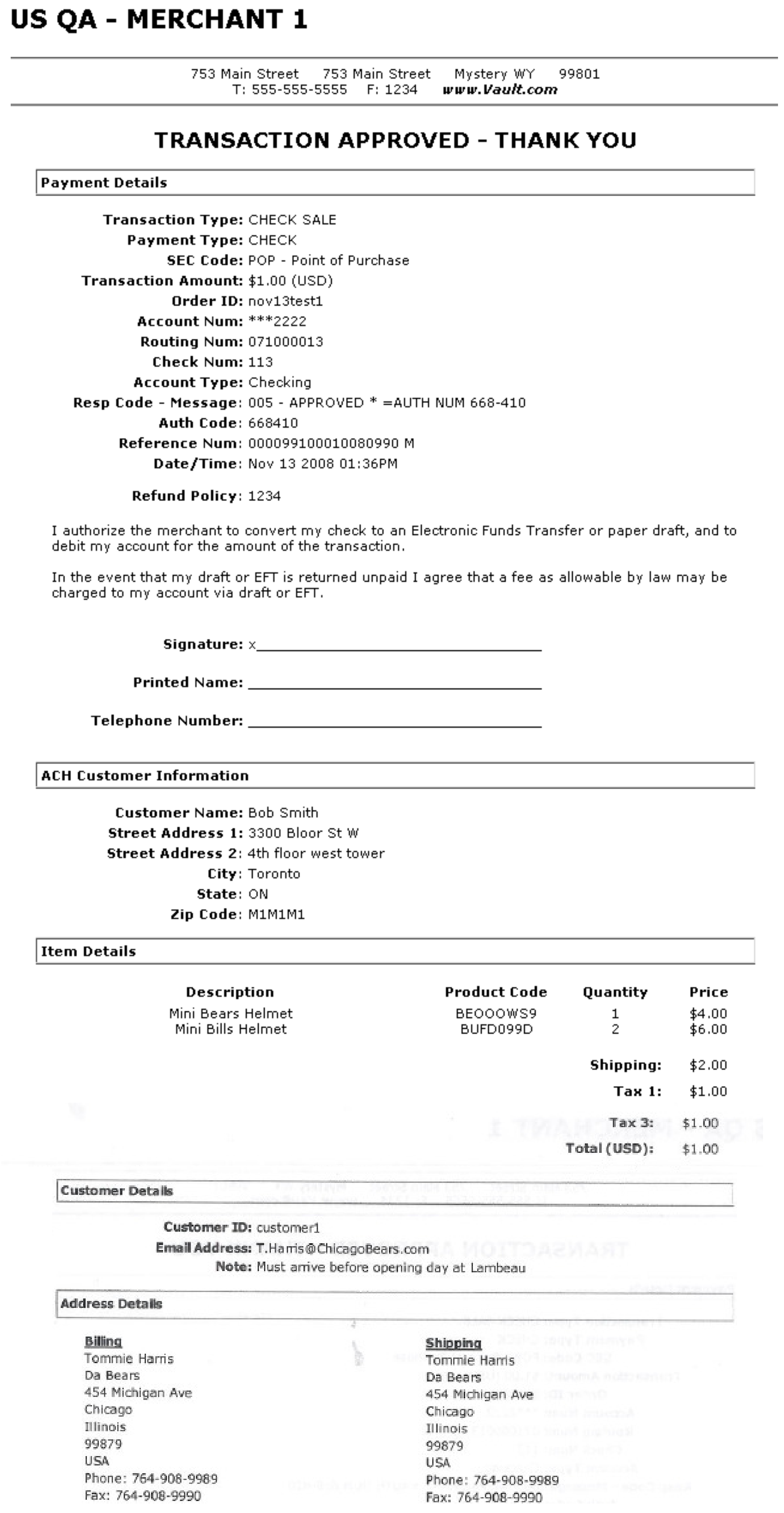
	Billing Address		**Shipping Address**
First Name:	Bob	**First Name:**	Mary
Last Name:	Smith	**Last Name:**	Smith
Company:	Moneris	**Company:**	My Company
Address:	101 Main St	**Address:**	111 Lakeshore Blvd
City:	New York	**City:**	Chicago
State:	NY	**State:**	Illinois
Zip Code:	123456	**Zip Code:**	234567
Country:	USA	**Country:**	USA
Phone:	555-555-5555	**Phone:**	555-111-2222
Fax:	555-555-5566	**Fax:**	555-222-4444

43. Appendix R. ACH Transaction Receipt (Check Physically Present)

For an ACH transaction, a transaction confirmation is not mandatory; though eSELECTplus does recommend that a receipt of registration of the transaction be provided to the customer. Below is a list of recommended fields and the format they are to be displayed in.

	Field	Description
1	Merchant Name	The name of the store / business.
2	Merchant URL	Web site address of the store / business.
3	Transaction Type	The type of transaction that was performed: - Check Sale (ACH Debit) - Check Reversal (ACH Reversal) *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Payment Type	Indicate that this is an ACH processed transaction.
5	SEC Code	Specify which SEC Code was submitted. Identifies how the bank account information was collected. i.e. POP – Point of Purchase
6	Transaction Amount	The total amount being debited or credited to the bank account.
7	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
8	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
9	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined the field may be omitted.
10	Response Code	The 3 digit response code returned in the transaction (ex. 000 – 999)
11	Response Message / Result	Message indicating whether the transaction was Registered or Failed to Register.
12	Account Number	Customer's bank account number. All but the last 4 digits of the account number must be masked out.
13	Routing Number	Check routing number to identify the Financial Institution.
14	Check Number	Used for check tracking purposes.
15	Account Type	Indicate whether this is a Savings or Checking account.

16	Signature	The signature forms the customer's authority for the ACH Debit transaction. It is only required for POP – Point of Purchase transactions. **NOTE: Only the merchant's copy requires the signature.** For Reversal transactions, the merchant must sign the customer's copy of the receipt.
17	Printed Name	The customer's name, as it appears on the check. Only required for POP – Point of Purchase transactions.
18	Telephone Number	The telephone number of the check holder. Only required for POP transactions.
19	Check Holder Agreement	This text is required on ACH Debit transaction when the SEC code is POP – Point of Purchase or BOC – Back Office Conversion. *I authorize the merchant to convert my check to an Electronic Funds Transfer or paper draft, and to debit my account for the amount of the transaction.* *In the event that my draft or EFT is returned unpaid I agree that a fee as allowable by law may be charged to my account via draft or EFT.*



US QA - MERCHANT 1

753 Main Street 753 Main Street Mystery WY 99801
T: 555-555-5555 F: 1234 ***www.Vault.com***

TRANSACTION APPROVED - THANK YOU

Payment Details

Transaction Type:	CHECK SALE
Payment Type:	CHECK
SEC Code:	POP - Point of Purchase
Transaction Amount:	$1.00 (USD)
Order ID:	nov13test1
Account Num:	***2222
Routing Num:	071000013
Check Num:	113
Account Type:	Checking
Resp Code - Message:	005 - APPROVED * =AUTH NUM 668-410
Auth Code:	668410
Reference Num:	000099100010080990 M
Date/Time:	Nov 13 2008 01:36PM
Refund Policy:	1234

I authorize the merchant to convert my check to an Electronic Funds Transfer or paper draft, and to debit my account for the amount of the transaction.

In the event that my draft or EFT is returned unpaid I agree that a fee as allowable by law may be charged to my account via draft or EFT.

Signature: x______________________________

Printed Name: ______________________________

Telephone Number: ______________________________

ACH Customer Information

Customer Name:	Bob Smith
Street Address 1:	3300 Bloor St W
Street Address 2:	4th floor west tower
City:	Toronto
State:	ON
Zip Code:	M1M1M1

Item Details

Description	Product Code	Quantity	Price
Mini Bears Helmet	BEOOOWS9	1	$4.00
Mini Bills Helmet	BUFD099D	2	$6.00
		Shipping:	$2.00
		Tax 1:	$1.00
		Tax 3:	$1.00
		Total (USD):	$1.00

Customer Details

Customer ID:	customer1
Email Address:	T.Harris@ChicagoBears.com
Note:	Must arrive before opening day at Lambeau

Address Details

Billing	**Shipping**
Tommie Harris	Tommie Harris
Da Bears	Da Bears
454 Michigan Ave	454 Michigan Ave
Chicago	Chicago
Illinois	Illinois
99879	99879
USA	USA
Phone: 764-908-9989	Phone: 764-908-9989
Fax: 764-908-9990	Fax: 764-908-9990