Merchant Integration Guide

PHP API – Vault

v1.2.7

Moneris SOLUTIONS

Revision Number	Date	Changes
V1.2.0	May 12, 2009	Document edited for coherence
V1.2.1	July 18, 2011	-New download link updated in various locations **http://www.eselectplus.ca/en/downloadable-content** -Section 8. How do I test my Solution? –Added CA Root Certificate File.
V1.2.2	September 9, 2011	-Appendix D. Recur Fields - Added recurUnit End of Month.
V1.2.3	January 5, 2012	-Appendix F. Card Validation Digits (CVD) -Added American Express/JCB response codes -Appendix G. Address Verification Service (AVS) -Added American Express/JCB response codes -Appendix H. Additional Information for CVD and AVS -Added American Express/JCB response codes
V1.2.4	March 23, 2012	-Appendix D. Recur Fields – Corrected recur examples
V1.2.5	June 6, 2013	-New download link updated in various locations: **https://developer.moneris.com/** -Section 4. Transaction Types - Added ResTokenizeCC - Added ResAddToken -Section 5 – Added ResTokenizeCC example -Added Section 8. Hosted Tokenization -Added Section 9. How to Charge a Temporary Token -Added Section 10. How to Turn a Temporary Token into a Permanent Token Appendix F and G – Moved the AVS and CVD response codes to a separate document
V1.2.6	October 15, 2013	-Section 2. Skill and Requirements - Updated PCI-DSS description -Section 4. Transaction Types - Added ResMpiTxn -Section 6. Financial Transaction Examples - Added ResMpiTxn
V1.2.7	November 21, 2014	-Appendix A & B corrected datakey size from 23 to 25

Table of Contents

**** PLEASE READ CAREFULLY****

You have a responsibility to protect cardholder and merchant related confidential account information. Under no circumstances should ANY confidential information be sent via email while attempting to diagnose integration or production issues. When sending sample files or code for analysis by Moneris staff, all references to valid card numbers, merchant accounts and transaction tokens should be removed and or obscured. Under no circumstances should live cardholder accounts be used in the test environment.

1. About this Documentation

This document describes the basic information for using the PHP API for sending Vault transactions as well as outlining all administrative functions of the Vault. The Vault feature allows a merchant to create customer profiles, edit those profiles, and use them to process transactions without having to enter financial information each time. This document will outline all the steps required in order to fully utilize this functionality and will not describe basic transaction processing. To access basic transaction processing information without the Vault, for example Refund and Void, please refer to the PHP API Integration Guide available for download at: **https://developer.moneris.com**.

2. System and Skill Requirements

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

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

cURL CA Root Certificate File:

The default installation of PHP/cURL does not include the cURL CA root certificate file. In order for the eSelectPlus PHP API to connect to the eSelectPlus gateway during transaction processing, the 'mpgclasses.php' file that's included with the PHP API package needs to be modified to include a path to the CA root certificate file. Follow the instructions below to set this up.

1) If cURL was not installed separately from your PHP installation, libcurl is included in your PHP installation. You will need to download the 'cacert.pem' file from 'http://curl.haxx.se/docs/caextract.html' and save it to the necessary directory. Once downloaded, rename the file to 'curl-ca-bundle.crt' (e.g. 'C:\path\to\curl-ca-bundle.crt'). If cURL was installed separately from PHP, you may need to determine the path to the cURL CA root certificate bundle on your system (e.g. 'C:\path\to\curl-ca-bundle.crt').

2) Insert the code below into the 'mpgclasses.php' file as part of the cURL option setting, at approximately line 73 below the line 'curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, TRUE);'

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

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

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, validation 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 2.0 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.

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

You will need to follow these steps:
1. Do the required development as outlined in this document.
2. Refer to the main PHP API Integration Guide to develop all follow-on procedures (ex. Refund) **https://developer.moneris.com**.
3. Test your solution in the test environment.
4. Activate your store.
5. Make the necessary changes to move your solution from the test environment into production as outlined in this document.

Transaction Types

The Vault API supports both financial and administrative transactions. These transactions are outlined below.

Vault Transactions (Admin)

ResAddCC – Create a new credit card profile. The fields which may be sent in are outlined in the transaction examples which can be found in section 4 of this documentation.

ResTokenizeCC – Create a new credit card profile using the credit card number and expiry date submitted in a previous financial transaction. The tokenize feature allows you to take a transaction that was previously done in eSELECTplus and store the card date from that transaction in the Moneris Vault.

ResUpdateCC – This will update a profile to contain Credit Card information using a unique data_key. All information contained within a credit card profile will be updated as indicated by the submitted fields. The mandatory fields for creating a new Credit Card profile will be required. These are all outlined in the transaction examples found in section 4 of this documentation.

ResDelete – Delete an existing profile of any payment type using the unique data_key which was assigned when the profile was first added. *It is important to note that once a profile is deleted, the information which was saved within can no longer be retrieved.*

ResGetExpiring – Retrieve all Credit cards which are about to expire, as well as the Vault data which is associated with each profile. This transaction will retrieve cards which will expire within the current calendar month or one month following. This transaction will be limited to being performed a maximum of 2 times per calendar day.

ResLookupMasked – Retrieve all Vault data that is associated with a unique data_key. The Credit Card number that will be returned will be masked.

ResLookupFull – Retrieve all Vault data that is associated with a unique data_key. Unlike ResLookupMasked, this transaction will return both the full unmasked Credit Card number as well as the masked value.

ResIscorporate – Determine if the credit card associated with an existing profile is a corporate card. The result of this transaction is returned as a Boolean value in the ‘CorporateCard’ response field.

ResAddToken – Convert a Hosted Tokenization temporary token into a permanent vault token.

Vault Transactions (Financial)

ResPreauthCC – This is a preauthorization transaction for Credit Card profiles. This transaction will use a unique data_key which will identify a previously registered Credit Card profile. The details within the profile will be submitted to perform the preauthorization transaction.

ResPurchaseCC – This is a purchase transaction for Credit Card profiles. This transaction will use a unique data_key which will identify a previously registered Credit Card profile. The details within the profile will be submitted to perform the purchase transaction.

ResIndRefundCC – This is an independent refund transaction which can be used for Credit Card profiles. This transaction will use a unique data_key which will identify a previously registered Credit Card profile. The details within the profile will be submitted to perform the independed refund transaction.

ResMpiTxn - Below is an example of a ResMpiTxn transaction. You can use the ResMpiTxn class to retrieve the vault transaction value to pass onto Visa and Mastercard.

4. Administrative Transactions

Included below is the sample code for the Administrative transactions that can be found in the "Examples" folder of the Vault PHP API download. Administrative transactions allow the user to perform such tasks as creating new Vault profiles, deleting existing profiles and updating profile information.

ResAddCC

This transaction is used to create a new Credit Card profile. A unique data_key will be generated and returned to the merchant in the response. This will be the identifier for this profile which all other Vault financial transactions will use in order to associate the transaction with the saved information. Please refer to sections 0 and 0 for examples of the financial transactions available.

The mandatory fields for this transaction are: pan, expdate, crypt_type (required to register a CC transaction but will not be used for any Vault financial transactions). Optional fields are: avsInfo, cust_id, email, phone, and note. The ResolveData that is returned in the response will indicate the fields registered for this profile.

```
<?php

## Example php -q TestResAddCC.php store3 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_add_cc';
$cust_id='';
$phone = '';
$email = 'bob@smith.com';
$note = 'this is my note';
$pan='5454545454545454';
$expiry_date='0812';
$crypt_type='1';
$avs_street_number = '';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '';

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

$txnArray=array('type'=>$type,
                'cust_id'=>$cust_id,
                'phone'=>$phone,
                'email'=>$email,
                'note'=>$note,
                'pan'=>$pan,
                'expdate'=>$expiry_date,
                'crypt_type'=>$crypt_type
                 );

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

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

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************** 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResTokenizeCC

The ResTokenizeCC transaction is used to create a new Credit Card profile, but using the credit card number, expiry date and crypt type from a previous financial transaction. Similarly to a ResAddCC, a unique data_key will be generated and returned to the merchant in the response. This will be the identifier for this profile which all other Vault financial transactions will use in order to associate the transaction with the saved information.

Transactions that can be tokenized include: Purchase, Preauthorization, Capture, Reauth, Refund, Purchase Correction, and Independent Refund

In order to tokenize a transaction you will need:

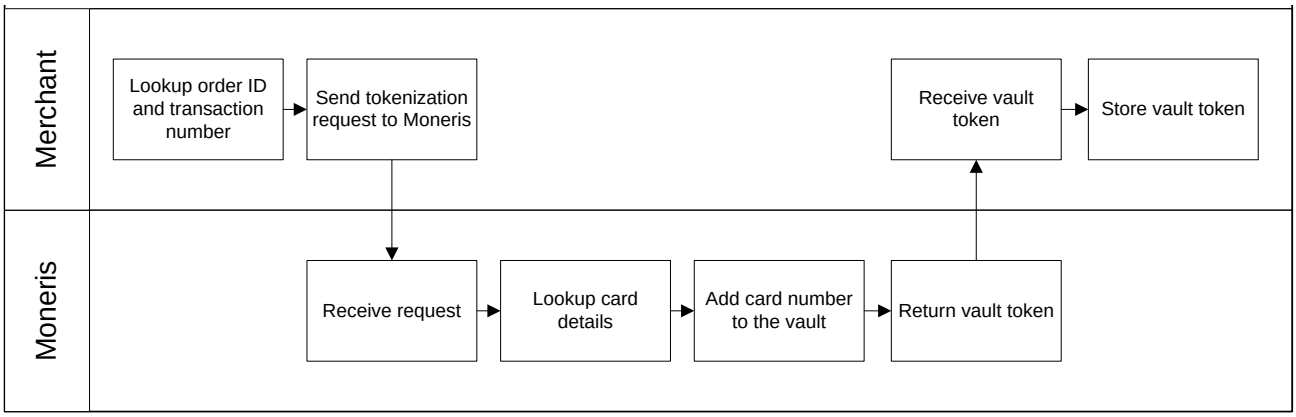
- Order ID
- Transaction Number (txn_num)

NOTE: The txn_num is returned in the response from the original transaction

These fields are required to reference a previously processed credit card financial transaction. The credit card number, expiry date, and crypt type from this transaction will be registered in the Vault for future Vault financial transactions.

Optional fields are: avs_info, cust_id, email, phone, and note. If the below optional fields are not sent in the tokenize request then they will not be stored with the transaction (eSELECTplus will not pull the optional transaction information from the previous transaction). The ResolveData that is returned in the response will indicate the fields registered for this profile.

Tokenize Process Diagram



Example Transaction Request:

```
<?php

require "../mpgClasses.php";

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

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

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

$type='res_tokenize_cc';
$order_id='res-purch-250113-20:01:11';
$txn_number='895697-0_8';
$cust_id='customer1';
$phone = '4165555555';
$email = 'bob@smith.com';
$note = 'this is my note';
```

```
$avs_street_number = '123';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '90210';

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

$txnArray=array('type'=>$type,
                                'order_id'=>$order_id,
                                'txn_number'=>$txn_number,
                                'cust_id'=>$cust_id,
                                'phone'=>$phone,
                                'email'=>$email,
                                'note'=>$note
                              );

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

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

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

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

$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResDelete

This transaction is used to delete an existing Vault profile. The data_key from the original profile will be required for this transaction. Within the ResolveData of the response, all details that were associated with the profile will be returned. Please note, the full card number will not be returned. Please refer to the ResLookupFull transaction to see how to retrieve these details prior to deleting the profile.

Please note: Once a profile is deleted, the details can no longer be retrieved

```
<?php

## Example php -q TestResDelete.php store3 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_delete';
$data_key='89p999Hx2VKa92E13867YLOgl';

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

$txnArray=array('type'=>$type,
                            'data_key'=>$data_key
                          );

/************************** 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResUpdateCC

This transaction is used to update an existing Vault profile. The ResUpdateCC transaction pertains to Credit Cards and will update the existing profile accordingly. All fields are optional besides the data_key. If a field is submitted, it will be updated. For example, if a blank field is submitted in cust_id, the cust_id will be deleted. The ResolveData will return all the details that are associated with the profile *after* the update.

```
<?php

## Example php -q TestResUpdateCC.php store3 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_update_cc';
$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$cust_id='customer1';
$phone = '5555555555';
$email = 'bob@smith.com';
$note = 'stuff';
$pan='5454545454545454';
$expiry_date='0909';
$crypt_type='7';

$avs_street_number = '123';
$avs_street_name = 'stuff dr';
$avs_zipcode = '90210';

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

$txnArray=array('type'=>$type,
                'data_key'=>$data_key,
                'cust_id'=>$cust_id,
                'phone'=>$phone,
                'email'=>$email,
                'note'=>$note,
                'pan'=>$pan,
                'expdate'=>$expiry_date,
                'crypt_type'=>$crypt_type
                 );

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

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

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

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);

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

$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResLookupFull

This transaction is used to verify what is currently saved under a given Vault profile. The data_key for the profile will need to be provided for this transaction. The response will return the latest active data for the given data_key. The ResLookupFull transaction returns the full pan as well as the masked_pan.

```
<?php

## Example php -q TestResLookupFull.php store3 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_lookup_full';      //will return both the full & masked card number
$data_key='79e34qjex6w1Z1bN3Q0F392H2';

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

$txnArray=array('type'=>$type,
                'data_key'=>$data_key
                 );

/************************** 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

print("\n\nCust ID = " . $mpgResponse->getResDataCustId());
print("\nPhone = " . $mpgResponse->getResDataPhone());
print("\nEmail = " . $mpgResponse->getResDataEmail());
print("\nNote = " . $mpgResponse->getResDataNote());
print("\nPan = " . $mpgResponse->getResDataPan());
print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
print("\nExp Date = " . $mpgResponse->getResDataExpDate());
print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());

?>
```

ResLookupMasked

This transaction is used to verify what is currently saved under a given Vault profile. The data_key for the profile will need to be provided for this transaction. The response will return the latest active data for the given data_key. The ResLookupMasked transaction returns the card number however only first 4 and last 4 digits. Please refer to the ResLookupFull transaction to retrieve the un-masked details from the profile.

```
<?php

## Example php -q TestResLookupMasked.php store3 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_lookup_masked';    //will only return the masked card number
$data_key='79e34qjex6w1Z1bN3Q0F392H2';

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

$txnArray=array('type'=>$type,
                'data_key'=>$data_key
               );

/************************** 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResGetExpiring

This transaction is used to verify which profiles will be expiring within the current month and one month following. For example, if processing this transaction on September 2nd 2008, then it will return all cards expiring in September and October of 2008. This particular transaction can only be performed a maximum of 2 times in any given calendar day, and it only applies to Credit Card profiles. The response will provide all expiring cards as well as the details registered in their profile for the specified store_id.

```
<?php

## Example php -q TestResGetExpiring.php store3 yesguy

//There is a max number of attempts set for this transaction per calendar day
//Can not surpass or will receive Invalid Transaction error

require "../mpgClasses.php";

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

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

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

$type='res_get_expiring';

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

$txnArray = array( 'type'=>$type );

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

$mpgTxn = new mpgTransaction($txnArray);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

$DataKeys = $mpgResponse->getDataKeys();

for($i=0; $i < count($DataKeys); $i++)
{
        $mpgResponse->setResolveData($DataKeys[$i]);

        print("\n\nData Key = " . $DataKeys[$i]);
        print("\nCust ID = " . $mpgResponse->getResDataCustId());
        print("\nPhone = " . $mpgResponse->getResDataPhone());
        print("\nEmail = " . $mpgResponse->getResDataEmail());
        print("\nNote = " . $mpgResponse->getResDataNote());
        print("\nMasked Pan = " . $mpgResponse->getResDataMaskedPan());
        print("\nExp Date = " . $mpgResponse->getResDataExpDate());
        print("\nCrypt Type = " . $mpgResponse->getResDataCryptType());
        print("\nAvs Street Number = " . $mpgResponse->getResDataAvsStreetNumber());
        print("\nAvs Street Name = " . $mpgResponse->getResDataAvsStreetName());
        print("\nAvs Zipcode = " . $mpgResponse->getResDataAvsZipcode());
}
?>
```

ResIscorporatecard

This transaction is used to verify if a profile has a corporate card registered within it. This transaction will return a 'true' value if it is determined that the card is corporate, 'false' otherwise. The response is returned within the getCorporateCard response method.

```
<?php

## Example php -q TestResIscorporatecard.php store5 yesguy

require "../mpgClasses.php";

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

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

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

$type='res_iscorporatecard';
$data_key='9A143sx23Y2Sb426J45GXYYM8';

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

$txnArray=array('type'=>$type,
                'data_key'=>$data_key
                );

/************************** 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nCorporateCard = " . $mpgResponse->getCorporateCard());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());
?>
```

5. Financial Transaction Examples

Included below is the sample code that can be found in the "Examples" folder of the Vault PHP API download. Vault transactions are very similar to regular financial transactions. The main difference is the use of a data_key which is used as a reference to all the mandatory financial information normally found in a regular transaction. Once the transaction is complete, the response will also include all the fields which are currently saved under the profile which was used. It is also important to note that cust_id is not a mandatory variable. If it is passed in, it will be used for the current transaction. If cust_id is not passed in and there is a cust_id saved in the customer profile, the profile cust_id will then be used. If no cust_id is passed in and there is none saved in the customer profile, the transaction will be completed without a cust_id.

ResPreauthCC (basic)

The ResPreauthCC transaction will process a preauth transaction for a Credit Card using saved information in a Vault profile. In the ResPreauthCC example we require several variables (store_id, api_token, data_key, order_id, amount, and crypt_type). If avs_info is registered in the profile, it will be submitted with the preauth as well as returned in the ResolveData portion of the response. EFraud is outlined in greater detail in the following section. Please refer to Appendix A: for variable definitions.

```
<?php

## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPreauthCC.php store3 yesguy unique_order_id cust_id 15.00 1

require "../mpgClasses.php";

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

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

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

$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-preauth-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';        //if sent will be submitted, otherwise cust_id from profile will be used
$crypt_type='1';

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

$txnArray =array(type=>'res_preauth_cc',
                      data_key=>$data_key,
                      order_id=>$orderid,
                      cust_id=>$custid,
                      amount=>$amount,
                      crypt_type=>$crypt_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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResPurchaseCC (basic)

The ResPurchaseCC transaction will process a purchase transaction for a Credit Card using saved information in a Vault profile. In the ResPurchaseCC example we require several variables (store_id, api_token, data_key, order_id, amount, and crypt_type). Cust_id is an optional variable. If avs_info is registered in the profile, it will be submitted with the purchase as well as returned in the ResolveData portion of the response. EFraud is outlined in greater detail in the following section. Please refer to Appendix A: for variable definitions.

```
<?php
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
## Example php -q TestResPurchaseCC.php store3 yesguy unique_order_id 1.00

require "../mpgClasses.php";

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

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

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

$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';

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

$txnArray=array(type=>'res_purchase_cc',
                      data_key=>$data_key,
                      order_id=>$orderid,
                      cust_id=>$custid,
                      amount=>$amount,
                      crypt_type=>$crypt_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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
```

```
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

ResIndRefundCC

The ResIndRefundCC will credit a specified amount to the cardholder's Credit Card. This transaction will process a regular Independent Refund on a card using the card information found in the Vault profile referenced by the data_key. Required fields for this transaction are: store_id, api_token, data_key, order_id, amount, and crypt_type. The optional variable is cust_id.

```
<?php

## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResIndRefundCC.php store3 yesguy unique_order_id cust_id 15.00 1

require "../mpgClasses.php";

/************************ Request Variables *********************************/
$store_id='store5';
$api_token='yesguy';
$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-ind-refund-'.date("dmy-G:i:s");
$amount='1.00';
$custid='';
$crypt_type='1';
$txnArray =array(type=>'res_ind_refund_cc',
                     data_key=>$data_key,
                     order_id=>$orderid,
                     cust_id=>$custid,
                     amount=>$amount,
                     crypt_type=>$crypt_type);
$mpgTxn = new mpgTransaction($txnArray);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgHttpPost = new mpgHttpsPost($store_id,$api_token,$mpgRequest);

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

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

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

ResMpiTxn

This transaction type supports the utilization of a Vault Data Key as opposed to a PAN, in a VBV/SecureCode Txn MPI transaction. This new transaction type allows the merchant to use a card currently stored within the vault by referencing the data key. The merchant may pass the data key to the MPI TXN request to verify if the card is enrolled in Verified by Visa or MasterCard SecureCode. Once the validation is completed the merchant may initiate the forming of the validation form(getMpiInLineForm()).For more information on integrating with Moneris MPI please refer to our MPI integration guides available in our Developer Portal.

```
<?php

require "../mpgClasses.php";

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

$store_id='moneris';
$api_token='hurgle';

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

$data_key=' FnUlvWe56jgVPvG4xaW2ynRwa';
$amount='1.00';
$xid = "12345678910111213116";
$MD = $xid."mycardinfo".$amount;
$merchantUrl = "www.mystoreurl.com";
$accept = "true";
$userAgent = "Mozilla";

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

$txnArray =array(type=>'res_mpitxn',
                                data_key=>$data_key,
                                amount=>$amount,
                                xid=>$xid,
                                MD=>$MD,
                                merchantUrl=>$merchantUrl,
                                accept=>$accept,
                                userAgent=>$userAgent
                                );

/************************ 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("\nMpiSuccess = " . $mpgResponse->getMpiSuccess());

if($mpgResponse->getMpiSuccess() == "true")
{
        print($mpgResponse->getMpiInLineForm());
}
else
{
        print("\nMpiMessage = " . $mpgResponse->getMpiMessage());
}

?>
```

6. Financial Transactions with Extra Features - Examples

In the previous section the instructions were provided for the financial transaction set. eSELECTplus also provides several extra features/functionalities for the financial transactions. These features include storing customer and order details, card verification digit check (CVD), address verification (AVS), and the Recurring Billing feature. AVS, CVD, and Recurring Billing must be added to your account, please call the Service Centre at 1-866-319-7450 to have your profile updated.

ResPurchaseCC (with Customer and Order details)

Below is an example of sending a ResPurchaseCC with the customer and order details. If one piece of CustInfo 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. It can be used in conjunction with other extra features such as AVS, CVD and Recurring Billing. ***Please note that the CustInfo fields are not used for any type of address verification or fraud check.***

```
<?php
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC-CustInfo.php store3 yesguy unique_order_id 1.00 1

require "../mpgClasses.php";

/************************ Request Variables *********************************/
$store_id='store5';
$api_token='yesguy';
$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';
$txnArray=array(type=>'res_purchase_cc',
                      data_key=>$data_key,
                      order_id=>$orderid,
                      cust_id=>$custid,
                      amount=>$amount,
                      crypt_type=>$crypt_type);

/********************* Set Customer info **************/
$mpgCustInfo = new mpgCustInfo();

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

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

/********************* Create Billing and Shipping 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);
```

```
$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 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);

$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCustInfo($mpgCustInfo);

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

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

ResPurchaseCC (with Recurring Billing)

Recurring Billing is a feature that allows the transaction information to be sent once and then re-billed on a specified interval for a certain number of times. This is a feature commonly used for memberships, subscriptions, or any other charge that is re-billed on a regular basis. The transaction is split into two parts; the recur information and the transaction information. Please see Appendix D. Recur Fields for description of each of the fields. The optional customer and order details can be included in the transaction using the method outlined above –*ResPurchaseCC (with Customer and Order Details).* This transaction allows the merchant to use the data registered within a profile to setup a customer for recurring billing. Once a recurring billing transaction has been initiated, it will no longer be linked to the Vault profile. All changes to the recurring billing details will need to be made using the Recurring Billing tools, for example, using the Merchant Resource Centre. Recurring Billing must be added to your account, please call the Service Centre at 1-866-319-7450 to have your profile updated.

```
<?php
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC-Recur.php store3 yesguy unique_order_id 1.00

require "../mpgClasses.php";

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

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

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

$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';

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

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

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

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

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);

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

$recurUnit = 'day';
$startDate = '2009/11/30';
$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
                        );

$mpgRecur = new mpgRecur($recurArray);

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

$txnArray=array(type=>'res_purchase_cc',
                        data_key=>$data_key,
                        order_id=>$orderid,
                        cust_id=>$custid,
                        amount=>$amount,
                        crypt_type=>$crypt_type
                        );

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

$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCvdInfo($mpgCvdInfo);
$mpgTxn->setRecur($mpgRecur);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

print("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nRecurSuccess = " . $mpgResponse->getRecurSuccess());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

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.

ResPurchaseCC (with CVD and AVS - eFraud)

Below is an example of a ResPurchaseCC transaction with CVD and AVS information. These values can be sent in conjunction with other additional variables such as Recurring Billing or customer information. It is important to note that if AVS details are sent, they will be submitted with the purchase but not stored. If they are not sent but avs_info is stored in the Vault profile, it will be submitted instead. If they are not sent and there was no stored avs_info found, no address verification will take place. To have the eFraud feature added to your profile, please call the Service Center at 1-866-319-7450 to have your profile updated.

When testing eFraud (AVS and CVD) you **must only use** the Visa test card numbers, 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at **https://developer.moneris.com**. These tests will only work with "store5" as the store_id.

```
<?php
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC-Efraud.php store3 yesguy unique_order_id 1.00 1

require "../mpgClasses.php";

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

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

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

$data_key='79e34qjex6w1Z1bN3Q0F392H2';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';

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

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

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

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

$mpgCvdInfo = new mpgCvdInfo ($cvdTemplate);

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

//The AVS portion is optional if AVS details are already stored in this profile
//If AVS details are resent in Purchase transaction, they will replace stored details

$avs_street_number = '';
$avs_street_name = 'bloor st';
$avs_zipcode = '111111';

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

$avsTemplate = array(
                      'avs_street_number' => $avs_street_number,
                      'avs_street_name' => $avs_street_name,
                      'avs_zipcode' => $avs_zipcode
                      );
$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
/************************ Transaction Array **********************************/

$txnArray=array(type=>'res_purchase_cc',
                      data_key=>$data_key,
                      order_id=>$orderid,
                      cust_id=>$custid,
```

```
                        amount=>$amount,
                        crypt_type=>$crypt_type
                        );

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

$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setCvdInfo($mpgCvdInfo);
$mpgTxn->setAvsInfo($mpgAvsInfo);

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

$mpgRequest = new mpgRequest($mpgTxn);

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

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

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

$mpgResponse=$mpgHttpPost->getMpgResponse();

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

//----------------- ResolveData ------------------------------

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

?>
```

As part of the eFraud response there will be two additional methods called getAvsResultCode( ) and getCvdResultCode( ). In the ResolveData, the AVS fields will be returned if avs_info is stored in the profile. If no avs_info was submitted with the purchase, then these details would have been used for verification.

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

7. Hosted Tokenization

Moneris Hosted Tokenization (HT) was designed as a solution for online e-commerce merchants that do not wish to handle credit card numbers directly on their websites and also have the ability to fully customize their check-out webpage's appearance. When a HT transaction is initiated the Moneris eSELECTplus payment gateway will present and display, on the merchant's behalf, a single text-box on the merchant's check-out page. The cardholder can then securely enter their credit card information into the text-box. Upon submission of the payment information on the check-out page the eSELECTplus gateway will return a temporary token representing the credit card number, to the merchant. This token would then be used in an API call to process a financial transaction directly with Moneris to charge the card. Upon receiving a response to the financial transaction, the merchant would then generate a receipt and allow the cardholder to continue on with the online shopping experience.

For more details on how to implement the Moneris Hosted Tokenization feature please see the Hosted Tokenization Integration Guide. The guide can be downloaded from the Moneris Developer Portal: **https://developer.moneris.com**

8. How to Charge a Temporary Token

The only difference between charging a temporary token and charging a normal Vault token is the expiry date. With the Vault token, the expiry date is stored with the card number as part of the Vault profile so there's no need to send the expiry date again with each transaction. However a temporary token only stores the card number so the expiry date must be sent when you charge the card via the API.

To charge a Temporary Token you may use the following transaction types:

- ResPurchaseCC
- ResPreauthCC
- ResIndRefundCC

Please refer to the section 5 above for these transaction examples.

Example Temporary Token Purchase

```
<?php
##
## This program takes 3 arguments from the command line:
## 1. Store id
## 2. api token
## 3. order id
##
## Example php -q TestResPurchaseCC.php store3 yesguy unique_order_id 1.00
##

require "../mpgClasses.php";

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

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

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

$temp_data_key='duZ2UOuosdySQPbfvNAvOUcDE';
$orderid='res-purch-'.date("dmy-G:i:s");
$amount='1.00';
$custid='cust';
$crypt_type='1';

$expdate='1511';
```

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

$txnArray=array(type=>'res_purchase_cc',
                       data_key=>$temp_data_key,
                        order_id=>$orderid,
                        cust_id=>$custid,
                        amount=>$amount,
                        crypt_type=>$crypt_type
                        expdate=>$expdate
                        );


/************************ 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("\nDataKey = " . $mpgResponse->getDataKey());
print("\nReceiptId = " . $mpgResponse->getReceiptId());
print("\nReferenceNum = " . $mpgResponse->getReferenceNum());
print("\nResponseCode = " . $mpgResponse->getResponseCode());
print("\nISO = " . $mpgResponse->getISO());
print("\nAuthCode = " . $mpgResponse->getAuthCode());
print("\nMessage = " . $mpgResponse->getMessage());
print("\nTransDate = " . $mpgResponse->getTransDate());
print("\nTransTime = " . $mpgResponse->getTransTime());
print("\nTransType = " . $mpgResponse->getTransType());
print("\nComplete = " . $mpgResponse->getComplete());
print("\nTransAmount = " . $mpgResponse->getTransAmount());
print("\nCardType = " . $mpgResponse->getCardType());
print("\nTxnNumber = " . $mpgResponse->getTxnNumber());
print("\nTimedOut = " . $mpgResponse->getTimedOut());
print("\nAVSResponse = " . $mpgResponse->getAvsResultCode());
print("\nResSuccess = " . $mpgResponse->getResSuccess());
print("\nPaymentType = " . $mpgResponse->getPaymentType());

//----------------- ResolveData ------------------------------

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

?>
```

9. How to Turn a Temporary Token into a Permanent Token

A temporary token will only be valid for 15 minutes from when it was created. However it's still possible to convert a temporary token into a permanent Vault token by using the Moneris Vault API's ResAddToken function.

The below example code demonstrates how to convert a Hosted Tokenization temporary token into a permanent Vault token for future use.

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

/************************** Request Variables *****************************/
$store_id='store5';
$api_token='yesguy';
$type='res_add_token';
$temp_data_key='ot-psW2riouTA03hXtDBBRJ2dZa7';
$cust_id='customer1';
$phone = '5555551234';
$email = 'bob@smith.com';
$note = 'this is my note';
$expiry_date='1512';
$crypt_type='1';
$avs_street_number = '123';
$avs_street_name = 'lakeshore blvd';
$avs_zipcode = '90210';

/********************** Transactional Associative Array **********************/
$txnArray=array('type'=>$type,
                      'data_key'=>$temp_data_key,
                      'cust_id'=>$cust_id,
                      'phone'=>$phone,
                      'email'=>$email,
                      'note'=>$note,
                      'expdate'=>$expiry_date,
                      'crypt_type'=>$crypt_type);
$avsTemplate = array('avs_street_number' => $avs_street_number,
                     'avs_street_name' => $avs_street_name,
                     'avs_zipcode' => $avs_zipcode);

$mpgAvsInfo = new mpgAvsInfo ($avsTemplate);
$mpgTxn = new mpgTransaction($txnArray);
$mpgTxn->setAvsInfo($mpgAvsInfo);
$mpgRequest = new mpgRequest($mpgTxn);
$mpgHttpPost  =new mpgHttpsPost($store_id,$api_token,$mpgRequest);

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

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

10. How Do I Test My Solution?

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

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

Test IDs			
store_id	**api_token**	**Username**	**Password**
store1	yesguy	demouser	password
store2	yesguy	demouser	password
store3	yesguy	demouser	password
store4	yesguy	demouser	password
store5	yesguy	demouser	password

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

Test Card Numbers	
Card Plan	**Card Number**
MasterCard	5454545454545454
Visa	4242424242424242 or 4005554444444403
Amex	373599005095005

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

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

The test environment will approve and decline credit card transactions based on the penny value of the amount field.
For example, a 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 for download at **https://developer.moneris.com**.

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

---

11. How Do I Get Help?

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

For Technical Support:
Phone: 1-866-319-7450
Email: **eselectplus@moneris.com**

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

When sending an email support request please be sure to mention that this is in reference to a Vault transaction, your name, 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.**

12. 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 ResPurchase, ResPreAuth and ResIndRefund attempt. Characters allowed for Order ID: **a-z A-Z 0-9 _ - : . @ spaces**
data_key	25 / an	An alphanumeric identifier used in Vault transactions to uniquely identify a Vault profile. The data_key is generated by Moneris Solutions and returned to the merchant when the profile is first registered using ResAddCC.
pan	20 / variable	Credit Card Number - no spaces or dashes. Most credit card numbers today are 16 digits in length but some 13 digits are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and/or potential support of private label card ranges.
expdate	4 / num	Expiry Date - format **YYMM** no spaces or slashes. PLEASE NOTE THAT THIS IS REVERSED FROM THE DATE DISPLAYED ON THE PHYSICAL CARD WHICH IS MMYY
amount	9 / decimal	Amount of the transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
crypt_type	1 / an	E-Commerce Indicator: 1 - Mail Order / Telephone Order - Single 2 - Mail Order / Telephone Order - Recurring 3 - Mail Order / Telephone Order - Instalment 4 - Mail Order / Telephone Order - Unknown Classification 7 - SSL enabled merchant 8 - Non Secure Transaction (Web or Email Based) 9 - SET non - Authenticated transaction
cust_id	50 / an	This is an optional field that can be either registered in a profile or sent as part of a ResPurchase, ResPreauth or ResIndRefund request. It is searchable from the Moneris Merchant Resource Centre. It is commonly used for policy number, membership number, student ID or invoice number.
phone	30 / an	Phone number of the customer. This is an optional field which can be sent in when creating or updating a Vault profile.
email	30 / an	Email of the customer. This is an optional field which can be sent in when creating or updating a Vault profile.
note	30 / an	This field can be used for supplementary information which is to be sent in with the transaction. This is an optional field which can be sent in when creating or updating a Vault profile.
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 bank has on file.
cvd_value	4 / num	Credit Card CVD value – this number accommodates either 3 or 4 digit CVD values. Refer to section 17 for further details.
cvd_indicator	1 / num	CVD presence indicator (1 digit – refer to section 17 for values)

NOTE

The order_id 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 _ - : . @ $ = /**

13. 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 Financial Transaction Responses (i.e. ResPurchase) < 50 Transaction approved >= 50 Transaction declined NULL Transaction was not sent for authorization * If you would like further details on the response codes that are returned please see the Response Codes document available for download at **https://developer.moneris.com** Vault Admin Responses (i.e. ResAdd or ResDelete) 001 Successfully registered CC details. Successfully updated CC details. Successfully deleted CC details. Successfully located CC details. Successfully located # expiring cards. (NOTE: # = the number of cards located) 983 Can not find previous 986 Incomplete: timed out 987 Invalid transaction 988 Can not find expiring cards Null Error: Malformed XML
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
RecurSuccess	true/false	Indicates whether the recurring billing transaction successfully registered.
AvsResultCode	1/alpha	Indicates the address verification result. Refer to section 17.
CvdResultCode	2/an	Indicates the CVD validation result. Refer to section 17.
ResSuccess	true/false	Indicates if Vault transaction was successful.
PaymentType	cc	Indicates the payment type associated with a Vault profile.
DataKey	25 / an	The data_key specified in the request. If processing a ResAdd transaction, then this will indicate the newly generated unique data_key associated with the new profile.
CorporateCard	true/false	Indicates whether the card associated with the vault profile is a corporate card or not.
ResolveData		The fields returned within ResolveData will coincide with the registered profile details. Please refer to the examples. Fields found in ResolveData are: data_key, payment_type, cust_id, phone, email, note, masked_pan, pan, expdate, crypt_type, avs_street_number, avs_street_name, avs_zipcode.

14. Appendix C. CustInfo Fields

Field Definitions		
Field Name	**Size/Type**	**Description**
Billing and Shipping Information		
NOTE: The fields for billing and shipping information are identical. Please refer to section 6 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 6 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	

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

NOTE

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

Also, the data sent in Billing and Shipping Address fields will not be used for any address verification. Please refer to the section 6 for further details about Address Verification Service (AVS).

15. Appendix D. Recur 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 Exampl	**Description**

`$data_key='k8n3X9E2h79VNPf3888331tpt';`
`$orderid='monthly_purchase';`
`$amount='15.00';`
`$crypt_type='7';`
 	
`$recurUnit = 'month';`
`$startDate = '2007/01/02';`
`$numRecurs = '12';`
`$recurInterval = '2';`
`$recurAmount = '30.00';`
`$startNow = 'false';`
 	
`$recurArray =`
`array(recur_unit=>$recurUnit,`
`start_date=>$startDate,`
`num_recurs=>$numRecurs,`
`start_now=>$startNow, period =>`
`$recurInterval, recur_amount=>`
`$recurAmount);`
	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.

`$data_key='k8n3X9E2h79VNPf3888331tpt';`
`$orderid='bi-weekly_purchase';`
`$amount='15.00';`
`$crypt_type='7';`
 	
`$recurUnit = 'week';`
`$startDate = '2007/01/02';`
`$numRecurs = '26';`
`$recurInterval = '2';`
`$recurAmount = '30.00';`
`$startNow = 'true';`
 	
`$recurArray =`
`array(recur_unit=>$recurUnit,`
`start_date=>$startDate,`
`num_recurs=>$numRecurs,`
`start_now=>$startNow, period =>`
`$recurInterval, recur_amount=>`
`$recurAmount);`
	In the example on the left the first charge will be billed immediately. The initial charge will be for $15.00. Then starting on Jan 2nd 2007 the credit card will be billed $30.00 every 2 weeks for 26 recurring charges. The card will be billed a total of 27 times. (1 x $15.00 (immediate) and 26 x $30.00 (recurring))

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

16. Appendix E. Error Messages

Global Error Receipt – You are not connecting to our servers. This can be caused by a firewall or your internet connection.

Response Code = NULL – The response code can be returned as null for a variety of reasons. A majority of the time the explanation is contained within the Message field. When a 'NULL' response is returned it can indicate that the Issuer, the credit card host, or the gateway is unavailable, either because they are offline or you are unable to connect to the internet. A 'NULL' can also be returned when a transaction message is improperly formatted.

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

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

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

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

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

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

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

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

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

Vault Specific Responses

Message: Can not find previous
Cause: data_key provided was not found in our records or profile is no longer active.

Message: Invalid Transaction
Cause: -Transaction can not be performed due to improper data being sent in.
- Mandatory field is missing or an invalid SEC code is sent in.

Message: Malformed XML
Cause: Parse error.

Message: Incomplete
Cause: - Timed out.
- Can not find expiring cards.

17. Appendix F. Card Validation Digits (CVD) and Address Verification Service (AVS)

Card Validation Digits (CVD)

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

Address Verification Service (AVS)

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

Additional Information for CVD and AVS

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

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

NOTE

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

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

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

18. Appendix G. Vault Receipts

When completing Vault financial transactions (ResPreauth, ResPurchase, ResIndRefund), a receipt will need to be presented to the customer. Receipt requirements depend on the type of transaction which was. For further details on all receipt requirements, please refer to the full PHP API Integration Guide available for download at: **https://developer.moneris.com**.

eSELECTplus™

Copyright Notice

Copyright © 2013 Moneris Solutions, 3300 Bloor Street West, Toronto, Ontario, M8X 2X2

All Rights Reserved. This manual shall not wholly or in part, in any form or by any means, electronic, mechanical, including photocopying, be reproduced or transmitted without the authorized, written consent of Moneris Solutions.

This document has been produced as a reference guide to assist Moneris client's hereafter referred to as merchants. Every effort has been made to the make the information in this reference guide as accurate as possible. The authors of Moneris Solutions shall have neither liability nor responsibility to any person or entity with respect to any loss or damage in connection with or arising from the information contained in this reference guide.

Trademarks

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

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

Printed in Canada.

10 9 8 7 6 5 4 3 2 1