# eSelect PLUS

## Merchant Integration Guide
### Ruby API – Vault
### v 1.2.5

**Moneris SOLUTIONS**

| Revision Number | Date | Description |
|---|---|---|
| **V1.2.0** | May 12, 2009 | Document edited for coherence |
| **V1.2.1** | July 22, 2011 | -New download link updated in various locations<br>http://www.eselectplus.ca/en/downloadable-content<br>-Section 8. How do I test my Solution?<br>     –Added CA Root Certificate File. |
| **V1.2.2** | September 9, 2011 | -Appendix D. Recur Fields<br>     - Added recurUnit End of Month. |
| **V1.2.3** | January 5, 2012 | -Appendix F. Card Validation Digits (CVD)<br>     -Added American Express/JCB response codes<br>-Appendix G. Address Verification Service (AVS)<br>     -Added American Express/JCB response codes<br>-Appendix H. Additional Information for CVD and AVS<br>     -Added American Express/JCB response codes |
| **V1.2.4** | March 23, 2012 | -Appendix D. Recur Fields  – Corrected recur examples |
| **V1.2.5** | November 21, 2014 | -Appendix A & B corrected datakey size from 23 to 25 |

### Table of Contents

## **** <u>PLEASE READ CAREFULLY</u>****

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 Ruby 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 Ruby API Integration Guide available at:
https://www3.moneris.com/connect/en/documents/index.html

# 2.  System and Skill Requirements

In order to use Ruby your system will need to have the following:
1. Port 443 open
2. Web server with an SSL certificate
3. RUBY

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

# 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 Ruby API Integration Guide to develop all follow-on procedures (ex. Refund)
     https://www3.moneris.com/connect/en/documents/index.html.
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.

# 4. 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 5 of this documentation.

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 5 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.

**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.

## 5. Administrative Transactions

Included below is the sample code for the Administrative transactions that can be found in the "Examples" folder of the Vault Ruby 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 4 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.

```ruby
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResAddCC.new

p.cust_id = "john smith"
p.phone = "416-555-5555"
p.email = "bob@smith.com"
p.note = "this is my note"
p.pan = "4242424242424242"
p.expdate = "0909"
p.crypt_type = "7"

#AvsInfo Variables
avs = ReqMod::AVSInfo.new
avs.avs_street_number = "1"
avs.avs_street_name = "bloor st"
avs.avs_zipcode = "90210"

p.avs_info = avs

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_add_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpDate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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\***

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResDelete.new

p.data_key = "3h113py0D29DUuN44142n8lbp"

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_delete = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s


#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpDate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResUpdateCC.new

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.cust_id = "john smith"
p.phone = "416-555-5555"
p.email = "bob@smith.com"
p.note = "this is my note"
p.pan = "4242424242424242"
p.expdate = "0910"
p.crypt_type = "7"

#AvsInfo Variables
avs = ReqMod::AVSInfo.new
avs.avs_street_number = "1"
avs.avs_street_name = "bloor st"
avs.avs_zipcode = "90210"

p.avs_info = avs

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_update_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpDate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResLookupFull.new

p.data_key = "k8n3X9E2h79VNPf3888331tpt"

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_lookup_full = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nPan = " + response.receipt.resolvedata[0].pan.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpDate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResLookupMasked.new

p.data_key = "k8n3X9E2h79VNPf3888331tpt"

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_lookup_masked = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s


#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpDate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```ruby
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResGetExpiring.new

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_get_expiring = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

response.receipt.resolvedata.each do |x|
  print "\n\nData Key = " + x.data_key.to_s
  print "\n\nPayment Type = " + x.payment_type.to_s
  print "\nCust ID = " + x.cust_id.to_s
  print "\nPhone = " + x.phone.to_s
  print "\nEmail = " + x.email.to_s
  print "\nNote = " + x.note.to_s
  print "\nMasked Pan = " + x.masked_pan.to_s
  print "\nExp Date = " + x.expdate.to_s
  print "\nCrypt Type = " + x.crypt_type.to_s
  print "\nAvs Street Number = " + x.avs_street_number.to_s
  print "\nAvs Street Name = " + x.avs_street_name.to_s
  print "\nAvs Zipcode = " + x.avs_zipcode.to_s
end
```

## 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.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResIscorporatecard.new

p.data_key = "k8n3X9E2h79VNPf3888331tpt"

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

r.res_iscorporatecard = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nCorporateCard = " + response.receipt.corporatecard.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nISO = " + response.receipt.iso.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s
```

# 6. Financial Transaction Examples

Included below is the sample code that can be found in the "Examples" folder of the Vault Ruby 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. Definition of Request Fields for variable definitions.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResPreauthCC.new

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubytest21"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

r.res_preauth_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s
print "\nAVSResponse = " + response.receipt.avsresultcode.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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. Definition of Request Fields for variable definitions.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResPurchaseCC.new

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubypurchasecctest5"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

r.res_purchase_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s
print "\nAVSResponse = " + response.receipt.avsresultcode.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResIndRefundCC.new

r = ReqMod::Request.new

r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubyindrefundcctest1"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

r.res_ind_refund_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

# 7.  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 1-866-319-8475 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.**

```
#!c:\ruby\bin\rubyw.exe
require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResPurchaseCC.new

r = ReqMod::Request.new
c = ReqMod::CustInfo.new
b = ReqMod::Billing.new
s = ReqMod::Shipping.new
i = ReqMod::Item.new

p.cust_info=c
p.cust_info.billing=b
p.cust_info.shipping=s
p.cust_info.item=i


r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubyrespurchasecctest4"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

#Customer Information Variables
#E-mail
p.cust_info.email="Joe@widgets.com"

#Instructions
p.cust_info.instructions="Make it fast"

#Billing Information
p.cust_info.billing.first_name = "Joe"
p.cust_info.billing.last_name = "Thompson"
p.cust_info.billing.company_name = "Widget Company Inc."
p.cust_info.billing.address = "111 Bolts Ave."
p.cust_info.billing.city = "Toronto"
p.cust_info.billing.province = "Ontario"
p.cust_info.billing.country = "Canada"
p.cust_info.billing.postal_code = "M8T 1T8"
p.cust_info.billing.phone_number = "416-555-5555"
p.cust_info.billing.fax = "416-555-5555"
p.cust_info.billing.tax1 = "123.45"
p.cust_info.billing.tax2 = "123.45"
p.cust_info.billing.tax3 = "15.45"
p.cust_info.billing.shipping_cost = "456.23"

#Shipping Information
p.cust_info.shipping.first_name = "Joe"
p.cust_info.shipping.last_name = "Thompson"
```

```
p.cust_info.shipping.company_name = "Widget Company Inc."
p.cust_info.shipping.address = "111 Bolts Ave."
p.cust_info.shipping.city = "Toronto"
p.cust_info.shipping.province = "Ontario"
p.cust_info.shipping.country = "Canada"
p.cust_info.shipping.postal_code = "M8T 1T8"
p.cust_info.shipping.phone_number = "416-555-5555"
p.cust_info.shipping.fax = "416-555-5555"
p.cust_info.shipping.tax1 = "123.45"
p.cust_info.shipping.tax2 = "123.45"
p.cust_info.shipping.tax3 = "15.45"
p.cust_info.shipping.shipping_cost = "456.23"

#Item Information
i1 = ReqMod::Item.new
i1.name = "item1 name"
i1.quantity = "53"
i1.product_code = "item1 product code"
i1.extended_amount = "1.00"

i2 = ReqMod::Item.new
i2.name = "item2 name"
i2.quantity = "24"
i2.product_code = "item2 product code"
i2.extended_amount = "1.00"

p.cust_info.item = Array.new
p.cust_info.item << i1 << i2

r.res_purchase_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s
print "\nAVSResponse = " + response.receipt.avsresultcode.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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.

```
#!c:\ruby\bin\rubyw.exe

require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResPurchaseCC.new

r = ReqMod::Request.new
rec = ReqMod::RecurInfo.new
p.recur=rec


r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubypurchasecctest12"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

#recur variables
p.recur.recur_unit = "day"
p.recur.start_date = "2009/11/30"
p.recur.num_recurs = "4"
p.recur.period = "10"
p.recur.recur_amount = "31.00"
p.recur.start_now = "true"


r.res_purchase_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s

#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
```

```
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

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.

```
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

## 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 form CvdInfo please refer to Appendix F.  Card Validation Digits (CVD), to form AvsInfo please refer to Appendix G.  Address Verification Service (AVS).  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 http://www.eselectplus.ca/en/downloadable-content.

```
#!c:\ruby\bin\rubyw.exe

require './mpgapi4r.rb' #path to usmpgapi4r.rb
p = ReqMod::ResPurchaseCC.new

r = ReqMod::Request.new
c = ReqMod::CVDInfo.new
a = ReqMod::AVSInfo.new

p.cvd_info=c
p.avs_info=a

r.store_id = 'moneris'
r.api_token = 'hurgle'

p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "rubypurchasetest26b"
p.cust_id = "john smith"
p.amount = "1.00"
p.crypt_type = "1"

#Efraud Variables
p.cvd_info.cvd_indicator="1"
p.cvd_info.cvd_value="099"

p.avs_info.avs_street_number="123"
p.avs_info.avs_street_name="East Street"
p.avs_info.avs_zipcode="M1M2M2"

r.res_preauth_cc = p

response = HttpsPoster.post(r)
print "\nDataKey = " + response.receipt.datakey.to_s
print "\nReceiptId = " + response.receipt.receiptid.to_s
print "\nReferenceNum = " + response.receipt.referencenum.to_s
print "\nResponse Code = " + response.receipt.responsecode.to_s
print "\nAuthCode = " + response.receipt.authcode.to_s
print "\nMessage = " + response.receipt.message.to_s
print "\nTransaction Date = " + response.receipt.transdate.to_s
print "\nTransaction Time = " + response.receipt.transtime.to_s
print "\nTransaction Type = " + response.receipt.transtype.to_s
print "\nComplete = " + response.receipt.complete.to_s
print "\nTransaction Amount = " + response.receipt.transamount.to_s
print "\nCard Type = " + response.receipt.cardtype.to_s
print "\nTxn Number = " + response.receipt.txnnumber.to_s
print "\nTimedOut = " + response.receipt.timedout.to_s
print "\nResSuccess = " + response.receipt.ressuccess.to_s
print "\nPaymentType = " + response.receipt.paymenttype.to_s
print "\nAVSResponse = " + response.receipt.avsresultcode.to_s
print "\nCVDResponse = " + response.receipt.cvdresultcode.to_s
```

```
#ResolveData
print "\n\nCust ID = " + response.receipt.resolvedata[0].cust_id.to_s
print "\nPhone = " + response.receipt.resolvedata[0].phone.to_s
print "\nEmail = " + response.receipt.resolvedata[0].email.to_s
print "\nNote = " + response.receipt.resolvedata[0].note.to_s
print "\nMasked Pan = " + response.receipt.resolvedata[0].masked_pan.to_s
print "\nExpdate = " + response.receipt.resolvedata[0].expdate.to_s
print "\nCrypt Type = " + response.receipt.resolvedata[0].crypt_type.to_s
print "\nAvs Street Number = " + response.receipt.resolvedata[0].avs_street_number.to_s
print "\nAvs Street Name = " + response.receipt.resolvedata[0].avs_street_name.to_s
print "\nAvs Zipcode = " + response.receipt.resolvedata[0].avs_zipcode.to_s
```

As part of the eFraud response there will be two additional methods called avsresultcode and cvdresultcode.  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 a list of possible CVD responses please refer to Appendix F.  Card Validation Digits (CVD) and for a list of AVS responses, please refer to Appendix G. Address Verification Service (AVS).

# 8.  How Do I Test My Solution?

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

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

| Test IDs | | | |
|---|---|---|---|
| store_id | api_token | Username | Password |
| store1 | yesguy | demouser | password |
| store2 | yesguy | demouser | password |
| store3 | yesguy | demouser | password |
| 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 http://www.eselectplus.ca/en/downloadable-content.

---

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

---

**Ruby CA Root Certificate File:**

The default installation of Ruby does not include the Net::HTTP CA root certificate file.  In order for the eSelectPlus Ruby API to connect to the eSelectPlus gateway during transaction processing, the 'mpgapi4r.rb' file that's included with the Ruby 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) You will need to download the 'carcert.pem' file from 'http://curl.haxx.se/docs/caextract.html' and save it to the necessary directory. Once downloaded, rename the file to 'ca-certificate.crt and determine the file path (e.g. 'C:\path\to\ ca-certificate.crt).

2) Insert the code below into the 'mpgapi4r.rb' file as part of the option setting, at approximately line 16 below the line 'http.verify_mode = OpenSSL::SSL::VERIFY_PEER'
**http.ca_file = "c:\path\to\ca-certificate.crt"**

For more information regarding the OpenSSL::SSL::VERIFY_PEER option, please refer to your Ruby manual.


# 9.  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

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

# 10.    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. <br> 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. <br> 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: <br> 1 - Mail Order / Telephone Order - Single <br> 2 - Mail Order / Telephone Order - Recurring <br> 3 - Mail Order / Telephone Order - Instalment <br> 4 - Mail Order / Telephone Order - Unknown Classification <br> 7 - SSL enabled merchant <br> 8 - Non Secure Transaction (Web or Email Based) <br> 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 <br> 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 Appendix F.  Card Validation Digits (CVD) for further details. |
| cvd_indicator | 1 / num | CVD presence indicator (1 digit – refer to Appendix F.  Card Validation Digits (CVD) 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 _ - : . @ $ = /**

## 11.    Appendix B. Definitions of Response Fields

| Response Fields | | |
|---|---|---|
| **Variable Name** | **Size/Type** | **Description** |
| order_id | 50 / an | order_id specified in request |
| ReferenceNum | 18 / num | The reference number is an 18 character string that references the terminal used to process the transaction as well as the shift, batch and sequence number, This data is typically used to reference transactions on the host systems and must be displayed on any receipt presented to the customer. This information should be stored by the merchant. The following illustrates the breakdown of this field where "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.<br><br>Moneris Host Transaction identifier. |
| ReponseCode | 3 / num | Transaction Response Code<br>Financial Transaction Responses (i.e. ResPurchase)<br>  < 50     Transaction approved<br>  >= 50    Transaction declined<br>  NULL    Transaction was not sent for authorization<br>* If you would like further details on the response codes that are returned please see the Response Codes document for download at http://www.eselectplus.ca/en/downloadable-content<br><br>Vault Admin Responses (i.e. ResAdd or ResDelete)<br>  001     Successfully registered CC details.<br>           Successfully updated CC details.<br>           Successfully deleted CC details.<br>           Successfully located CC details.<br>           Successfully located # expiring cards.<br>           (NOTE: # = the number of cards located)<br>  983     Can not find previous<br>  986     Incomplete: timed out<br>  987     Invalid transaction<br>  988     Can not find expiring cards<br>  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 Appendix G.  Address Verification Service (AVS). |
| CvdResultCode | 2/an | Indicates the CVD validation result.  Refer to Appendix F.  Card Validation Digits (CVD). |
| 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. |

## 12.    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 7 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 7 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 7 for further details about Address Verification Service (AVS).

## 13.    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 : <br> period = 45, recur_unit=day -> Card will be billed every 45 days. <br> period = 4, recur_unit=weeks -> Card will be billed every 4 weeks. <br> period = 3, recur_unit=month -> Card will be billed every 3 months. <br> period = 3, recur_unit=eom -> Card will be billed every 3 months (on the last day of the month) <br> 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 Example** | **Description** |
| ```
p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "monthly_purchase"
p.amount = "15.00"
p.crypt_type = "7"

p.recur.recur_unit = "month"
p.recur.start_date = "2007/01/02"
p.recur.num_recurs = "12"
p.recur.period = "2"
p.recur.recur_amount = "30.00"
p.recur.start_now = "false"

r.res_purchase_cc = p
``` | 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. |
| ```
p.data_key = "k8n3X9E2h79VNPf3888331tpt"
p.order_id = "bi-weekly_purchase"
p.amount = "15.00"
p.crypt_type = "7"

#recur variables
p.recur.recur_unit = "week"
p.recur.start_date = "2007/01/02"
p.recur.num_recurs = "26"
p.recur.period = "2"
p.recur.recur_amount = "30.00"
p.recur.start_now = "true"

r.res_purchase_cc = p
``` | 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".

# 14.    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.

# 15.    Appendix F.  Card Validation Digits (CVD)

The Card Validation Digits (CVD) value refers to the numbers appearing on the back of the credit card which are not imprinted on the front.  The exception to this is with American Express cards where this value is indeed printed on the front.  The mpgCvdInfo parameter is broken down into two elements.  The first element is the CVD Value itself.

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

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

## CVD Response codes:

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

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

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

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

## 16.    Appendix G.  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.  mpgAvsInfo is broken down into three elements:

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

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

| Table of possible Visa and MC AVS result codes | | | |
|---|---|---|---|
| **VALUE** | **DESCRIPTION** | **Domestic** | **International** |
| A | Street addresses match.  The street addresses match but the postal/ZIP codes do not, or the request does not include the postal/ZIP code. | √ | √ |
| B | Street addresses match.  Postal code not verified due to incompatible formats.  (Acquirer sent both street address and postal code.) | √ | √ |
| C | Street address and postal code not verified due to incompatible formats. (Acquirer sent both street address and postal code.) | √ | √ |
| D | Street addresses and postal codes match. | | √ |
| G | Address information not verified for international transaction. | | √ |
| I | Address information not verified. | | √ |
| M | Street address and postal code match. | | √ |
| N | No match.  Acquirer sent postal/ZIP code only, or street address only, or both postal code and street address. | √ | √ |
| P | Postal code match.  Acquirer sent both postal code and street address, but street address not verified due to incompatible formats. | √ | √ |
| R | Retry:  System unavailable or timed out.  Issuer ordinarily performs its own AVS but was unavailable.  Available for U.S. issuers only. | √ | |
| S | Not applicable.  If present, replaced with G (for international) or U (for domestic) by V.I.P.  Available for U.S. Issuers only. | √ | |
| U | Address not verified for domestic transaction.  Visa tried to perform check on issuer's behalf but no AVS information was available on record, issuer is not an AVS participant, or AVS data was present in the request but issuer did not return an AVS result. | √ | |
| W | Not applicable.  If present, replaced with "Z" by V.I.P.  Available for U.S. issuers only. | √ | |
| X | Not applicable.  If present, replaced with "Y" by V.I.P.  Available for U.S. issuers only. | √ | |
| Y | Street address and postal code match. | √ | |
| Z | Postal/ZIP matches; street address does not match or street address not included in request. | √ | √ |

| Table of possible AVS response codes from Discover ||
|---|---|
| **VALUE** | **DESCRIPTION** |
| **X** | All digits match, nine-digit zip code. |
| **A** | All digits match, five-digit zip code. |
| **Y** | Address matches, zip code does not match. |
| **T** | Nine-digit zip code matches, address does not match. |
| **Z** | Five-digit zip codes matches, address does not match. |
| **N** | Nothing matches. |
| **W** | No data from issuer/authorization system. |
| **U** | Retry, system unable to process. |
| **S** | AVS not supported at this time. |
| Table of possible AVS response codes from American Express or JCB ||
| **VALUE** | **DESCRIPTION** |
| **A** | Billing address matches, postal code does not |
| **D** | Customer name incorrect, postal code matches |
| **E** | Customer name incorrect, billing address and postal code match |
| **F** | Customer name incorrect, billing address matches |
| **K** | Customer name matches |
| **L** | Customer name and postal code match |
| **M** | Customer name, billing address, and postal code match |
| **N** | Billing address and postal code do not match |
| **O** | Customer name and billing address match |
| **R** | System unavailable; retry |
| **S** | AVS not currently supported |
| **U** | Information is unavailable |
| **W** | Customer name, billing address, and postal code are all correct |
| **Y** | Billing address and postal code both match |
| **Z** | Postal code matches, billing address does not |

## 17.    Appendix H. Additional Information for CVD and AVS

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

Please note that all responses coming back from these verification methods are not direct indicators of whether a merchant should complete any particular transaction.  The responses should <u>not</u> 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. <br><br> Also, please note that AVS verification is only applicable towards Visa, MasterCard, Discover and American Express transactions.  This verification method is <u>not</u> applicable towards any other card type. |

## 18.    Appendix I. 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 Ruby API Integration Guide available at:
https://www3.moneris.com/connect/en/documents/index.html

# eSELECTplus™

## Copyright Notice

Copyright © 2012 Moneris Solutions, 3300 Bloor Street West, Toronto, Ontario, M8X 2X2

## Trademarks