Merchant Integration Guide

US .NET API – Vault

V 1.1.8

Revision Number	**Date**	**Changes**
V1.1.1	Mary 5, 2009	-Document edited for coherence
V1.1.2	June 13, 2011	-New download link updated in various locations https://esplusqa.moneris.com/connect/en/download/index.php -Appendix A. Definition of Request Fields -Added new Variable Name (dynamic_descriptor) -Appendix D. Recur Fields - Added recur_unit End of Month.
V1.1.3	January 4, 2012	-Appendix G. Card Validation Digits (CVD) -Added American Express/JCB response codes -Appendix H. Address Verification Service (AVS) -Added American Express/JCB response codes -Appendix I. Additional Information for CVD and AVS -Added American Express/JCB response codes
V1.1.4	March 29, 2012	-Section 2. System and Skill Requirements -Added PCI & PA DSS note. -Appendix D. Recur Fields – Corrected recur examples
V1.1.5	September 6, 2012	-New download link updated in various locations: https://developer.moneris.com/ -Section 4. Transaction Types – Added ResTokenizeCC -Section 5. Administrative Transactions – Added ResTokenizeCC example
V1.1.6	November 13, 2012	-Section 4. Transaction Types – Added EncResAddCC & EncResUpdateCC -Section 5. Administrative Transactions - Added EncResAddCC example - Added EncResUpdateCC example -Section 7. Financial Transaction with Extra features: ResPurchaseCC with CVD and AVS (eFraud) -Added CVD note. -Appendix A. Definition of Request Fields – Added new variable names (enc_track2 & device_type) – Added CVD note. -Appendix G. Card Validation Digits (CVD) – Added CVD note.
V1.1.7	July 02, 2013	-Section 4. Transaction Types - Added ResAddToken -Section 8. Added Hosted Tokenization -Section 9. Added How to charge a Temporary Token -Section 10. Added How to Turn a Temporary Token into a Permanent Token Appendix G – Shifted AVS and CVD response codes to a separate document.
V1.1.8	May 26, 2014	-Section 9. How to charge a Temporary Token – Updated ResPurchaseCC

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 .NET 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 .NET API Integration Guide available at: https://developer.moneris.com

2. System and Skill Requirements

In order to use .NET your system will need to have the following:
1. A web server with an SSL certificate
2. .NET Framework
3. Port 443 open for bi-directional communication

As well, you will need to have the following knowledge and/or skill set:
1. Install a dll into the global assembly cache
2. Knowledge of the .NET Framework

Note:

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

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

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

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

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

You will need to follow these steps:
1. Do the required development as outlined in this document.
2. Refer to the main .NET 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.

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.

EncResAddCC – Similar to the regular ResAddCC, the Encrypted ResAddCC transaction type creates a new credit card profile. This transaction type requires the card data to be either swiped or manually keyed in via a Moneris provided encrypted mag swipe reader.

ResTokenizeCC - Create a new credit card profile using the credit card number and expiry date submitted in a previous financial transaction. The fields which may be sent in are outlined in the transaction examples which can be found in section 5 of this documentation.

ResAddAch – Create a new ACH profile. The fields which may be sent in are outlined in the transaction examples which can be found in section 5 of this documentation.

ResAddPinless – Create a new Pinless Debit 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. If the profile which is being updated was already a Credit Card profile, all information contained within it will simply be updated as indicated by the submitted fields. If however the profile was of a different payment type (ie: ACH or Pinless Debit), the old profile will be deactivated and the new Credit Card information will be associated with the data_key. As a result, 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.

EncResUpdateCC – Similar to the regular ResUpdateCC, the Encrypted ResUpdateCC transaction type will update a profile to contain Credit Card information using a unique data_key. This transaction type requires the card data to be either swiped or manually keyed in via a Moneris provided encrypted mag swipe reader.

ResUpdateAch – This will update a profile to contain ACH information using a unique data_key. If the profile which is being updated was already an ACH profile, all information contained within it will simply be updated as indicated by the submitted fields. If however the profile was of a different payment type (ie: Credit Card or Pinless Debit), the old profile will be deactivated and the new ACH information will be associated with the data_key. As a result, the mandatory fields for creating a new ACH profile will be required. These are all outlined in the transaction examples found in section 5 of this documentation.

ResUpdatePinless – This will update a profile to contain Pinless Debit Card information using a unique data_key. If the profile which is being updated was already a Pinless Debit Card profile, all information contained within it will simply be updated as indicated by the submitted fields. If however the profile was of a different payment type (ie: Credit Card or ACH), the old profile will be deactivated and the new Pinless Debit Card information will be associated with the data_key. As a result, the mandatory fields for creating a new Pinless Debit 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 and Pinless Debit 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, Pinless Debit Card or bank account 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, Pinless Debit Card or bank account number as well as the masked value.

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 only. 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.

ResPurchase(CC|ACH|Pinless) – This is a purchase transaction which can be used for all the payment types. For Credit Cards, this is processed as a USPurchase transaction. For ACH this is processed as a ACHDebit. For Pinless Debit, this is processed as a USPinlessDebitPurchase transaction. The name of the transaction (ex. ResPurchaseCC) must coincide with the payment type associated with the data_key (ex.Credit Card).

ResIndRefund(CC|ACH) – This is an independent refund transaction which can be used for Credit Card and ACH profiles only. For ACH transactions, this is processed as a ACHCredit. The name of the transaction (ex. ResIndRefundCC) must coincide with the payment type associated with the data_key (ex.Credit Card).

5. Administrative Transactions

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

ResAddCC

The ResAddCC 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 6 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: avs_info, cust_id, email, phone, and note. The ResolveData that is returned in the response will indicate the fields registered for this profile.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;
        public class TestResAddCC
        {
          public static void Main(string[] args)
          {
               string host = "esplusqa.moneris.com";
               string store_id = "monusqa002";
               string api_token = "qatoken";
               string pan = "5454545454545454";
               string expdate = "0909";
               string phone = "0000000000";
               string email = "bob@smith.com";
               string note = "my note";
               string cust_id = "customer1";
               string crypt_type = "7";
               AvsInfo avsCheck = new AvsInfo();
               avsCheck.SetAvsStreetNumber("212");
               avsCheck.SetAvsStreetName("Payton Street");
               avsCheck.SetAvsZipCode("M1M1M1");

               USResAddCC usResAddCC = new USResAddCC(cust_id, phone, email, note, pan, expdate, crypt_type);
               //************************OPTIONAL VARIABLES***************************
               usResAddCC.SetAvsInfo(avsCheck);
               HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResAddCC);
        /**********************   REQUEST  ************************/
        try
        {
             Receipt receipt = mpgReq.GetReceipt();
             Console.WriteLine("DataKey = " + receipt.GetDataKey());
             Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
             Console.WriteLine("Message = " + receipt.GetMessage());
             Console.WriteLine("TransDate = " + receipt.GetTransDate());
             Console.WriteLine("TransTime = " + receipt.GetTransTime());
             Console.WriteLine("Complete = " + receipt.GetComplete());
             Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
             Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
             Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
             //ResolveData
             Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
             Console.WriteLine("Phone = " + receipt.GetResDataPhone());
             Console.WriteLine("Email = " + receipt.GetResDataEmail());
             Console.WriteLine("Note = " + receipt.GetResDataNote());
             Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
             Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
             Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
             Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
             Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
             Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
        }
         catch (Exception e)
         {
              Console.WriteLine(e);
         }
          }
        }
}
```

EncResAddCC

Similar to the standard ResAddCC transaction type, the EncResAddCC transaction is used to create a new Credit Card profile. The EncResAddCC transaction allows the merchant to swipe or manually key in the credit card details using a Moneris provided encrypted reader and submit the encrypted track2 details. 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 6 for examples of the financial transactions available.

The mandatory fields for this transaction are: enc_track2, device_type, and 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.

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

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestEncResAddCC
         {
           public static void Main(string[] args)
           {
                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa002";
                  string api_token = "qatoken";

                  string enc_track2 =
                          "02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^**************************************
                          **?*;4924********4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283D
                          BEBB2C6B3FDEACF7B5B314219D76C00890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A1
                          8B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F
                          1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C31415939320000A00028
                          3C5E03";
                  string device_type = "idtech";
                  string phone = "0000000000";
                  string email = "bob@smith.com";
                  string note = "my note";
                  string cust_id = "customer1";
                  string crypt_type = "1";

                  AvsInfo avsCheck = new AvsInfo();
                  avsCheck.SetAvsStreetNumber("212");
                  avsCheck.SetAvsStreetName("Payton Street");
                  avsCheck.SetAvsZipCode("M1M1M1");


                  USEncResAddCC usEncResAddCC = new USEncResAddCC(enc_track2, device_type, crypt_type);

                  //***********************OPTIONAL VARIABLES***************************

                  usResAddCC.SetCustId(cust_id);
                  usResAddCC.SetPhone(phone);
                  usResAddCC.SetEmail(email);
                  usResAddCC.SetNote(note);
                  usResAddCC.SetAvsInfo(avsCheck);

                  HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usEncResAddCC);

                  /**********************   REQUEST  ***********************/

                 try
                 {
                          Receipt receipt = mpgReq.GetReceipt();

                          Console.WriteLine("DataKey = " + receipt.GetDataKey());
                          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                          Console.WriteLine("Message = " + receipt.GetMessage());
                          Console.WriteLine("TransDate = " + receipt.GetTransDate());
                          Console.WriteLine("TransTime = " + receipt.GetTransTime());
                          Console.WriteLine("Complete = " + receipt.GetComplete());
```

```
                              Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                              Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                              Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                              //ResolveData
                              Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                              Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                              Console.WriteLine("Email = " + receipt.GetResDataEmail());
                              Console.WriteLine("Note = " + receipt.GetResDataNote());
                              Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                              Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                              Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                              Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                              Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                              Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                     }
                     catch (Exception e)
                     {
                              Console.WriteLine(e);
                     }
           }

         }
}
```

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.

The mandatory fields for this transaction are: order_id, txn_number (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. The ResolveData that is returned in the response will indicate the fields registered for this profile.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResAddCC
         {
           public static void Main(string[] args)
           {
                  string host = "esplusqa.moneris.com";
                 string store_id = "monusqa002";
                 string api_token = "qatoken";
                 string order_id = "dotnettokenize1";
                 string txn_number = "565534-0_10";
                 string phone = "0000000000";
                 string email = "bob@smith.com";
                 string note = "my note";
                 string cust_id = "customer1";

                 AvsInfo avsCheck = new AvsInfo();
                 avsCheck.SetAvsStreetNumber("212");
                 avsCheck.SetAvsStreetName("Payton Street");
                 avsCheck.SetAvsZipCode("M1M1M1");

                 USResTokenizeCC usResTokenizeCC = new USResTokenizeCC(order_id, txn_number);

                 //***********************OPTIONAL VARIABLES***************************
                 usResTokenizeCC.SetCustId(cust_id);
                 usResTokenizeCC.SetPhone(phone);
                 usResTokenizeCC.SetEmail(email);
                 usResTokenizeCC.SetNote(note);
                 usResTokenizeCC.SetAvsInfo(avsCheck);

                 HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResTokenizeCC);

                 try
                 {
                          Receipt receipt = mpgReq.GetReceipt();
                         Console.WriteLine("DataKey = " + receipt.GetDataKey());
                         Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                         Console.WriteLine("Message = " + receipt.GetMessage());
                         Console.WriteLine("TransDate = " + receipt.GetTransDate());
                         Console.WriteLine("TransTime = " + receipt.GetTransTime());
                         Console.WriteLine("Complete = " + receipt.GetComplete());
                         Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                         Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                         Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
                         //ResolveData
                         Console.WriteLine("Cust ID = " + receipt.GetResDataCustId());
                         Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                         Console.WriteLine("Email = " + receipt.GetResDataEmail());
                         Console.WriteLine("Note = " + receipt.GetResDataNote());
                         Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                         Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                         Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                         Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                         Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                         Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                 }
                 catch (Exception e)
                 {
                     Console.WriteLine(e);
                 }
           }
         }
}
```

ResAddAch

The ResAddAch transaction is used to create a new ACH profile. A 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 note, only the following SEC codes are currently supported: PPD, CCD, and WEB. The SEC code, as well as the rest of the ACHInfo data, that is registered will be submitted with all future Vault transactions unless it is later updated. Mandatory fields are: sec, routing_num, account_num, account_type. Optional fields are: phone, email, note, cust_id and all other ACHInfo fields.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResAddAch
        {
          public static void Main(string[] args)
          {
               string host = "esplusqa.moneris.com";
               string store_id = "monusqa002";
               string api_token = "qatoken";
               string phone = "0000000000";
               string email = "bob@smith.com";
               string note = "my note";
               string cust_id = "customer1";

               //ACHInfo Variables
               string sec = "ppd";
               string cust_first_name = "Christian";
               string cust_last_name = "M";
               string cust_address1 = "3300 Bloor St W";
               string cust_address2 = "4th floor west tower";
               string cust_city = "Toronto";
               string cust_state = "ON";
               string cust_zip = "M1M1M1";
               string routing_num = "490000018";
               string account_num = "222222";
               string check_num = "11";
               string account_type = "checking";

               ACHInfo achinfo = new ACHInfo(sec, cust_first_name, cust_last_name, cust_address1, cust_address2,
               cust_city, cust_state, cust_zip, routing_num, account_num, check_num, account_type);

               USResAddAch usResAddAch = new USResAddAch(cust_id, phone, email, note);

               usResAddAch.SetAchInfo(achinfo);

               //************************OPTIONAL VARIABLES***************************
               HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResAddAch);

        /**********************   REQUEST  ************************/
        try
        {
            Receipt receipt = mpgReq.GetReceipt();

            Console.WriteLine("DataKey = " + receipt.GetDataKey());
            Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
            Console.WriteLine("Message = " + receipt.GetMessage());
            Console.WriteLine("TransDate = " + receipt.GetTransDate());
            Console.WriteLine("TransTime = " + receipt.GetTransTime());
            Console.WriteLine("Complete = " + receipt.GetComplete());
            Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
            Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
            Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

            //ResolveData
            Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
            Console.WriteLine("Phone = " + receipt.GetResDataPhone());
            Console.WriteLine("Email = " + receipt.GetResDataEmail());
            Console.WriteLine("Note = " + receipt.GetResDataNote());
            Console.WriteLine("Sec = " + receipt.GetResDataSec());
            Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
            Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
            Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
            Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
```

```
             Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                 Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                 Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                 Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                 Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                 Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                 Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
         }
          catch (Exception e)
         {
                 Console.WriteLine(e);
         }
           }
         } // end TestDrive Item
}
```

ResAddPinless

The ResAddPinless transaction is used to create a new Pinless Debit profile. A data_key will be generated and returned to the merchant in the response. This will be the unique identifier for this profile which all other Vault financial transactions will use in order to associate the transaction with the saved information. The presentation_type that is registered will be submitted with all future Vault financial transactions, unless it is later updated. Mandatory fields are pan and presentation_type. Optional fields are email, note, phone, cust_id, expdate, and p_account_number.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

    public class TestResAddPinless
    {
      public static void Main(string[] args)
      {
        string host = "esplusqa.moneris.com";
        string store_id = "monusqa002";
        string api_token = "qatoken";
        string phone = "0000000000";
        string email = "bob@smith.com";
        string note = "my note";
        string cust_id = "customer1";
        string pan = "4242424242424242";
        string expdate = "1111";
        string presentation_type = "W";
        string p_account_number = "1231232131232132l3123123";

        USResAddPinless usResAddPinless = new USResAddPinless(cust_id, phone, email, note, pan, expdate,
        presentation_type, p_account_number);

        /**********************   REQUEST  ************************/

        HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResAddPinless);

        try
        {
                Receipt receipt = mpgReq.GetReceipt();

                Console.WriteLine("DataKey = " + receipt.GetDataKey());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
        }
        catch (Exception e)
        {
                Console.WriteLine(e);
        }
      }
    }
}
```

ResDelete

The ResDelete 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 or account 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

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResDelete
         {
           public static void Main(string[] args)
           {
                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa002";
                  string api_token = "qatoken";
                  string data_key = "F9X268oHsX115o521OJ11t9";

                  USResDelete usResDelete = new USResDelete(data_key);
                  HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResDelete);

         try
         {
                   Receipt receipt = mpgReq.GetReceipt();

                  Console.WriteLine("DataKey = " + receipt.GetDataKey());
                  Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                  Console.WriteLine("Message = " + receipt.GetMessage());
                  Console.WriteLine("TransDate = " + receipt.GetTransDate());
                  Console.WriteLine("TransTime = " + receipt.GetTransTime());
                  Console.WriteLine("Complete = " + receipt.GetComplete());
                  Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                  Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                  Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                  //ResolveData
                  Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                  Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                  Console.WriteLine("Email = " + receipt.GetResDataEmail());
                  Console.WriteLine("Note = " + receipt.GetResDataNote());
                  Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                  Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                  Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                  Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                  Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                  Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                  Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                  Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
                  Console.WriteLine("Sec = " + receipt.GetResDataSec());
                  Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                  Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                  Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                  Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                  Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                  Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                  Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                  Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                  Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                  Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                  Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
         }
         catch (Exception e)
         {
                  Console.WriteLine(e);
         }
           }
         }
}
```

ResUpdateCC

The ResUpdateCC transaction is used to update an existing Vault profile. The ResUpdateCC transaction pertains to Credit Cards and will update the existing profile accordingly. If the profile is of a different payment type, it will be automatically deactivated and a new Credit Card profile will be created and assigned to the data_key. The only data that will remain from the prior profile is the cust_id, phone, email and note associated with this data_key. For example, if data_key 'abc' refers to an ACH profile but it is submitted in the ResUpdateCC, the ACHInfo details will be deactivated and the new CC details will be registered. In this example, because the payment type is being changed, the following fields would be mandatory: pan, expdate, crypt_type. Otherwise, if the payment type is not being changed, 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.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResUpdateCC
        {
          public static void Main(string[] args)
          {
            string host = "esplusqa.moneris.com";
            string store_id = "monusqa002";
            string api_token = "qatoken";
            string data_key = "180298J12ku85m2u2gNC7Tk";
            string pan = "4242424242424242";
            string expdate = "1111";
                string phone = "0000000000";
            string email = "bob@smith.com";
            string note = "my note";
            string cust_id = "customer1";
            string crypt_type = "7";

            AvsInfo avsCheck = new AvsInfo();

            avsCheck.SetAvsStreetNumber("212");
            avsCheck.SetAvsStreetName("Payton Street");
            avsCheck.SetAvsZipCode("M1M1M1");

            USResUpdateCC usResUpdateCC = new USResUpdateCC(data_key);

            usResUpdateCC.SetAvsInfo(avsCheck);
            usResUpdateCC.SetCustId(cust_id);
            usResUpdateCC.SetPan(pan);
            usResUpdateCC.SetExpdate(expdate);
            usResUpdateCC.SetPhone(phone);
            usResUpdateCC.SetEmail(email);
            usResUpdateCC.SetNote(note);
            usResUpdateCC.SetCryptType(crypt_type);

            /**********************   REQUEST  ************************/
            HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResUpdateCC);

        try
        {
                Receipt receipt = mpgReq.GetReceipt();
                Console.WriteLine("DataKey = " + receipt.GetDataKey());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
```

```
                 Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                 Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                 Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                 Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                         Console.WriteLine(e);
                }
          }

        }
}
```

EncResUpdateCC

Similar to the standard ResUpdateCC transaction, the EncResUpdateCC is used to update an existing Vault profile. The EncResUpdateCC transaction pertains to Credit Cards and will update the existing profile accordingly. The EncResAddCC transaction allows the merchant to swipe or manually key in the credit card details using a Moneris provided encrypted reader and submit the encrypted track2 details.

If the profile is of a different payment type, it will be automatically deactivated and a new Credit Card profile will be created and assigned to the data_key. The only data that will remain from the prior profile is the cust_id, phone, email and note associated with this data_key. For example, if data_key 'abc' refers to an ACH profile but it is submitted in the EncResUpdateCC, the ACHInfo details will be deactivated and the new CC details will be registered. In this example, because the payment type is being changed, the following fields would be mandatory: enc_track2, device_type, crypt_type and data_key. Otherwise, if the payment type is not being changed, only the enc_track2, device_type and data_key are mandatory and all other fields are optional. 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.

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

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestEncResUpdateCC
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";

                string data_key = "32b55Q428ySjNBpbXw24128p5";
                string enc_track2 =
                        "02D901801F4F2800039B%*4924********4030^TESTCARD/MONERIS^***************************************
                        **?*;4924********4030=*******************?*A7150C78335A5024949516FDA9A68A91C4FBAB1279DD1DE2283D
                        BEBB2C6B3FDEACF7B5B314219D76C00890F347A9640EFE90023E31622F5FD95C14C0362DD2EAB28ADEB46B8B577DA1A1
                        8B707BCC7E48068EFF1882CFB4B369BDC4BB646C870D6083239860B23837EA91DB3F1D8AD066DAAACE2B2DA18D563E4F
                        1EF997696337B8999E9C707DEC4CB0410B887291CAF2EE449573D01613484B80760742A3506C31415939320000A00028
                        3C5E03";
                string device_type = "idtech";
                string phone = "0000000000";
                string email = "bob@smith.com";
                string note = "my note";
                string cust_id = "customer1";
                string crypt_type = "1";

                AvsInfo avsCheck = new AvsInfo();

                avsCheck.SetAvsStreetNumber("212");
                avsCheck.SetAvsStreetName("Payton Street");
                avsCheck.SetAvsZipCode("M1M1M1");

                USEncResUpdateCC usEncResUpdateCC = new USEncResUpdateCC(data_key);

                usEncResUpdateCC.SetAvsInfo(avsCheck);
                usEncResUpdateCC.SetCustId(cust_id);
                usEncResUpdateCC.SetEncTrack2(enc_track2);
                usEncResUpdateCC.SetDeviceType(device_type);
                usEncResUpdateCC.SetPhone(phone);
                usEncResUpdateCC.SetEmail(email);
                usEncResUpdateCC.SetNote(note);
                usEncResUpdateCC.SetCryptType(crypt_type);

                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usEncResUpdateCC);
```

```
                 /**********************   REQUEST  ************************/

                 try
                 {
                          Receipt receipt = mpgReq.GetReceipt();

                          Console.WriteLine("DataKey = " + receipt.GetDataKey());
                          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                          Console.WriteLine("Message = " + receipt.GetMessage());
                          Console.WriteLine("TransDate = " + receipt.GetTransDate());
                          Console.WriteLine("TransTime = " + receipt.GetTransTime());
                          Console.WriteLine("Complete = " + receipt.GetComplete());
                          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                          Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                          Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                          //ResolveData
                          Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                          Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                          Console.WriteLine("Email = " + receipt.GetResDataEmail());
                          Console.WriteLine("Note = " + receipt.GetResDataNote());
                          Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                          Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                          Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                          Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                          Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                          Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                 }
                 catch (Exception e)
                 {
                          Console.WriteLine(e);
                 }
           }

         }
}
```

ResUpdateAch

The ResUpdateAch transaction is used to update an existing Vault profile. The ResUpdateAch transaction pertains to ACH details and will update the existing profile accordingly. If the profile is of a different payment type, it will be automatically deactivated and a new ACH profile will be created and assigned to the data_key. A full explanation of how this update will behave can be found in the ResUpdateCC example above.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResUpdateAch
         {
           public static void Main(string[] args)
           {
                  string host = "esplusqa.moneris.com";
                 string store_id = "monusqa002";
                 string api_token = "qatoken";
                 string data_key = "180298J12ku85m2u2gNC7Tk";
                  string phone = "0000000000";
                 string email = "bob@smith.com";
                 string note = "my note";
                 string cust_id = "customer1";

                 //ACHInfo Variables
                  string sec = "ppd";
                  string cust_first_name = "Christian";
                  string cust_last_name = "M";
                 string cust_address1 = "3300 Bloor St W";
                 string cust_address2 = "4th floor west tower";
                 string cust_city = "Toronto";
                 string cust_state = "ON";
                 string cust_zip = "M1M1M1";
                 string routing_num = "490000018";
                 string account_num = "222222";
                 string check_num = "11";
                 string account_type = "checking";

                 ACHInfo achinfo = new ACHInfo();

                 achinfo.SetSec(sec);
                 achinfo.SetCustFirstName(cust_first_name);
                 achinfo.SetCustLastName(cust_last_name);
                 achinfo.SetCustAddress1(cust_address1);
                 achinfo.SetCustAddress2(cust_address2);
                 achinfo.SetCustCity(cust_city);
                 achinfo.SetCustState(cust_state);
                 achinfo.SetCustZip(cust_zip);
                 achinfo.SetRoutingNum(routing_num);
                 achinfo.SetAccountNum(account_num);
                 achinfo.SetCheckNum(check_num);
                 achinfo.SetAccountType(account_type);

                 USResUpdateAch usResUpdateAch = new USResUpdateAch(data_key);

                 usResUpdateAch.SetAchInfo(achinfo);
                 usResUpdateAch.SetCustId(cust_id);
                 usResUpdateAch.SetPhone(phone);
                 usResUpdateAch.SetEmail(email);
                 usResUpdateAch.SetNote(note);

                 HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResUpdateAch);

                  try
                  {
                 Receipt receipt = mpgReq.GetReceipt();

                  Console.WriteLine("DataKey = " + receipt.GetDataKey());
                  Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                  Console.WriteLine("Message = " + receipt.GetMessage());
                  Console.WriteLine("TransDate = " + receipt.GetTransDate());
                  Console.WriteLine("TransTime = " + receipt.GetTransTime());
                  Console.WriteLine("Complete = " + receipt.GetComplete());
                  Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                  Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                  Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
```

```
                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("Sec = " + receipt.GetResDataSec());
                Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
                }
                catch (Exception e)
                {
                        Console.WriteLine(e);
                }
          }
        }
}
```

ResUpdatePinless

The ResUpdatePinless transaction is used to update an existing Vault profile. The ResUpdatePinless transaction pertains to Pinless Debit and will update the existing profile accordingly. If the profile is of a different payment type, it will be automatically deactivated and a new Pinless Debit profile will be created and assigned to the data_key. A full explanation of how this update will behave can be found in the ResUpdateCC example above.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResUpdatePinless
        {
          public static void Main(string[] args)
          {
            string host = "esplusqa.moneris.com";
            string store_id = "monusqa002";
            string api_token = "qatoken";
            string data_key = "180298J12ku85m2u2gNC7Tk";
            string pan = "4242424242424242";
            string expdate = "1111";
                string phone = "0000000000";
            string email = "bob@smith.com";
            string note = "my note";
            string cust_id = "customer1";
            string presentation_type = "W";
            string p_account_number = "23456789";

            USResUpdatePinless usResUpdatePinless = new USResUpdatePinless(data_key);

            usResUpdatePinless.SetCustId(cust_id);
            usResUpdatePinless.SetPan(pan);
            usResUpdatePinless.SetExpdate(expdate);
            usResUpdatePinless.SetPhone(phone);
            usResUpdatePinless.SetEmail(email);
            usResUpdatePinless.SetNote(note);
            usResUpdatePinless.SetPresentationType(presentation_type);
            usResUpdatePinless.SetPAccountNumber(p_account_number);

            HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResUpdatePinless);

        try
        {
            Receipt receipt = mpgReq.GetReceipt();

            Console.WriteLine("DataKey = " + receipt.GetDataKey());
            Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
            Console.WriteLine("Message = " + receipt.GetMessage());
            Console.WriteLine("TransDate = " + receipt.GetTransDate());
            Console.WriteLine("TransTime = " + receipt.GetTransTime());
            Console.WriteLine("Complete = " + receipt.GetComplete());
            Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
            Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
            Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

            //ResolveData
            Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
            Console.WriteLine("Phone = " + receipt.GetResDataPhone());
            Console.WriteLine("Email = " + receipt.GetResDataEmail());
            Console.WriteLine("Note = " + receipt.GetResDataNote());
            Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
            Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
            Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
            Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
        }
        catch (Exception e)
        {
                Console.WriteLine(e);
        }
          }
        }
}
```

ResLookupFull

The ResLookupFull 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 or account_num as well as the masked_pan or masked_account_num.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestResLookupFull
        {
          public static void Main(string[] args)
          {
                 string host = "esplusqa.moneris.com";
                 string store_id = "monusqa002";
                 string api_token = "qatoken";
                 string data_key = "6sEdhp45QjY21795P12866a";

                 USResLookupFull usResLookupFull = new USResLookupFull(data_key);
                 HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResLookupFull);

        try {
                Receipt receipt = mpgReq.GetReceipt();

                 Console.WriteLine("DataKey = " + receipt.GetDataKey());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                 Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                 //ResolveData
                 Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("Pan = " + receipt.GetResDataPan());
                 Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                 Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                 Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                 Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                 Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                 Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                 Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                 Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
                 Console.WriteLine("Sec = " + receipt.GetResDataSec());
                 Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                 Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                 Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                 Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                 Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                 Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                 Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                 Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                 Console.WriteLine("Account Num = " + receipt.GetResDataAccountNum());
                 Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                 Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                 Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
            }
                catch (Exception e)
                 {
                         Console.WriteLine(e);
                }
          }
        }
}
```

ResLookupMasked

The ResLookupMasked 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, or the masked account number depending on the payment type. Please refer to the ResLookupFull transaction to retrieve the un-masked details from the profile.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResLookupMasked
        {
          public static void Main(string[] args)
          {
                 string host = "esplusqa.moneris.com";
                 string store_id = "monusqa002";
                 string api_token = "qatoken";
                 string data_key = "6sEdhp45QjY21795P12866a";

                 USResLookupMasked usResLookupMasked = new USResLookupMasked(data_key);
                 HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token,  usResLookupMasked);

              try
              {
                  Receipt receipt = mpgReq.GetReceipt();

                 Console.WriteLine("DataKey = " + receipt.GetDataKey());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                 Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                 //ResolveData
                 Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                 Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                 Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                 Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                 Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                 Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                 Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                 Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
                 Console.WriteLine("Sec = " + receipt.GetResDataSec());
                 Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                 Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                 Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                 Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                 Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                 Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                 Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                 Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                 Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                 Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                 Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
              }
                catch (Exception e)
                 {
                    Console.WriteLine(e);
                 }
          }
        }
}
```

ResGetExpiring

The ResGetExpiring 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 and Pinless Debit profiles. Please note, any Pinless Debit profile which does not have an expiry date registered will not be returned in the ResGetExpiring transaction. The response will provide all expiring cards as well as the details registered in their profile for the specified store_id.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResGetExpiring
        {
          public static void Main(string[] args)
          {
               string host = "esplusqa.moneris.com";
               string store_id = "monusqa002";
               string api_token = "qatoken";

               USResGetExpiring usResGetExpiring = new USResGetExp  iring();
               HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResGetExpiring);
               try
               {
                 Receipt receipt = mpgReq.GetReceipt();

                 Console.WriteLine("DataKey = " + receipt.GetDataKey());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                 Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                 //ResolveData
                 foreach (string dataKey in receipt.GetDataKeys())
                 {
                         Console.WriteLine("\nDataKey = " + dataKey);
                         Console.WriteLine("Payment Type = " + receipt.GetExpPaymentType(dataKey));
                         Console.WriteLine("Cust ID = " + receipt.GetExpCustId(dataKey));
                         Console.WriteLine("Phone = " + receipt.GetExpPhone(dataKey));
                         Console.WriteLine("Email = " + receipt.GetExpEmail(dataKey));
                         Console.WriteLine("Note = " + receipt.GetExpNote(dataKey));
                         Console.WriteLine("Masked Pan = " + receipt.GetExpMaskedPan(dataKey));
                         Console.WriteLine("Exp Date = " + receipt.GetExpExpdate(dataKey));
                         Console.WriteLine("Crypt Type = " + receipt.GetExpCryptType(dataKey));
                         Console.WriteLine("Avs Street Number = " + receipt.GetExpAvsStreetNumber(dataKey));
                         Console.WriteLine("Avs Street Name = " + receipt.GetExpAvsStreetName(dataKey));
                         Console.WriteLine("Avs Zipcode = " + receipt.GetExpAvsZipCode(dataKey));
                         Console.WriteLine("Presentation Type = " + receipt.GetExpPresentationType(dataKey));
                         Console.WriteLine("P Account Number = " + receipt.GetExpPAccountNumber(dataKey));
                 }
               }
              catch (Exception e)
              {
                  Console.WriteLine(e);
              }
          }
        }
}
```

6. Financial Transaction Examples

Included below is the sample code that can be found in the "Examples" folder of the Vault .NET 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. It is important to note that the transaction type used must match the payment type which is saved in the profile. For example, a ResPurchaseCC transaction may not use a data_key which references an ACH profile. 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 USPreAuth 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 USPreAuth 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.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

      public class TestResPreauthCC
      {
       public static void Main(string[] args)
       {

         string host = "esplusqa.moneris.com";
         string store_id = "monusqa002";
         string api_token = "qatoken";
         string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
         string order_id = "qa31050031";
         string amount = "1.00";
         string cust_id = "customer1";  //if sent will be submitted, otherwise cust_id from profile will be used
         string crypt_type = "1";

         USResPreauthCC usResPreauthCC = new USResPreauthCC(data_key, order_id, cust_id, amount, crypt_type);

         usResPreauthCC.SetDynamicDescriptor("FromQA");

         HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPreauthCC);

         /**********************   REQUEST  ************************/

         try
         {
                 Receipt receipt = mpgReq.GetReceipt();

                 Console.WriteLine("DataKey = " + receipt.GetDataKey());
                 Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                 Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("TransType = " + receipt.GetTransType());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                 Console.WriteLine("CardType = " + receipt.GetCardType());
                 Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                 Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                 //ResolveData
                 Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                 Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                 Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                 Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                 Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                 Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
         }
         catch (Exception e)
         {
                 Console.WriteLine(e);
         }
        }

      } // end TestDrive Item
}
```

ResPurchaseCC (basic)

The ResPurchaseCC transaction will process a USPurchase 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). Optional variables are cust_id, commcard_invoice and commcard_tax_amount. The commcard_* variables should be passed in blank if not used.). If avs_info is registered in the profile, it will be submitted with the USPurchase 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.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;
         public class TestResPurchaseCC
         {
           public static void Main(string[] args)
           {
                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa002";
                  string api_token = "qatoken";
                  string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
                  string order_id = "qa31050023";
                  string amount = "1.00";
                  string cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be used
                  string crypt_type = "1";
                  string commcard_invoice = "invoice";
                  string commcard_tax_amount = "1.00";

                  USResPurchaseCC usResPurchaseCC = new USResPurchaseCC(data_key, order_id, cust_id, amount, crypt_type);

                  /**********************   OPTIONAL VARIABLES  ***********************/
                  usResPurchaseCC.SetCommcardInvoice(commcard_invoice);
                  usResPurchaseCC.SetCommcardTaxAmount(commcard_tax_amount);
                  usResPurchaseCC.SetDynamicDescriptor("FromQA");
                  HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseCC);

                  /**********************   REQUEST  ***********************/
                 try
                 {
                          Receipt receipt = mpgReq.GetReceipt();
                          Console.WriteLine("DataKey = " + receipt.GetDataKey());
                          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                          Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                          Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                          Console.WriteLine("Message = " + receipt.GetMessage());
                          Console.WriteLine("TransDate = " + receipt.GetTransDate());
                          Console.WriteLine("TransTime = " + receipt.GetTransTime());
                          Console.WriteLine("TransType = " + receipt.GetTransType());
                          Console.WriteLine("Complete = " + receipt.GetComplete());
                          Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                          Console.WriteLine("CardType = " + receipt.GetCardType());
                          Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                          Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                          Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
                          //ResolveData
                          Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                          Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                          Console.WriteLine("Email = " + receipt.GetResDataEmail());
                          Console.WriteLine("Note = " + receipt.GetResDataNote());
                          Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                          Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                          Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                          Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                          Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                          Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                 catch (Exception e)
                 {
                          Console.WriteLine(e);
                 }
           }
         } // end TestDrive Item
}
```

ResPurchaseAch (basic)

In the ResPurchaseAch example we require several variables (store_id, api_token, data_key, order_id, amount). The optional variable is cust_id. This transaction will be processed as an ACHDebit. The ACHInfo registered for this profile will be used. The details submitted within 'ACHInfo' will be returned in the response within ResolveData. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResPurchaseAch
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "1sZH2p215HdQV616909x217";
                string order_id = "res_puchase_ach_1";
                string amount = "1.00";
                string cust_id = "customer1";

                USResPurchaseAch usResPurchaseAch = new USResPurchaseAch(data_key, order_id, cust_id, amount);
                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseAch);
        try {
                Receipt receipt = mpgReq.GetReceipt();
                Console.WriteLine("DataKey = " + receipt.GetDataKey());
                Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("TransType = " + receipt.GetTransType());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                Console.WriteLine("CardType = " + receipt.GetCardType());
                Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("Sec = " + receipt.GetResDataSec());
                Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }
        }
}
```

ResPurchasePinless (basic)

In the ResPurchasePinless example we require several variables (store_id, api_token, data_key, order_id, amount, intended_use). Optional variables are cust_id and p_account_number. If p_account_number is sent, it will be submitted with the purchase but not stored in the profile. If however it is not sent, the p_account_number will be pulled from the profile. If no p_account_number is sent or found in the profile, an error will be returned. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResPurchasePinless
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "UIV4907Qf1Er2198R951h0B";
                string order_id = "res_purchase_pinless_1";
                string amount = "1.00";
                string cust_id = "customer1";
                string intended_use = "1";
                string p_account_number = "23456789";

                USResPurchasePinless usResPurchasePinless = new USResPurchasePinless(data_key, order_id, cust_id,
                amount, intended_use, p_account_number);

                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchasePinless);

                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

                    Console.WriteLine("DataKey = " + receipt.GetDataKey());
                    Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                    Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                    Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                    Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                    Console.WriteLine("Message = " + receipt.GetMessage());
                    Console.WriteLine("TransDate = " + receipt.GetTransDate());
                    Console.WriteLine("TransTime = " + receipt.GetTransTime());
                    Console.WriteLine("TransType = " + receipt.GetTransType());
                    Console.WriteLine("Complete = " + receipt.GetComplete());
                    Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                    Console.WriteLine("CardType = " + receipt.GetCardType());
                    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                    Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                    Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                    Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                    //ResolveData
                    Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                    Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                    Console.WriteLine("Email = " + receipt.GetResDataEmail());
                    Console.WriteLine("Note = " + receipt.GetResDataNote());
                    Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                    Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                    Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                    Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());
                 }
                catch (Exception e)
                 {
                   Console.WriteLine(e);
                 }
          }
        }
}
```

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.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResIndRefundCC
         {
           public static void Main(string[] args)
           {

             string host = "esplusqa.moneris.com";
             string store_id = "monusqa002";
             string api_token = "qatoken";
             string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
             string order_id = "qa31050034";
             string amount = "1.00";
             string cust_id = "customer1";
             string crypt_type = "1";

             USResIndRefundCC usResIndRefundCC = new USResIndRefundCC(data_key, order_id, cust_id, amount, crypt_type);

             usResIndRefundCC.SetDynamicDescriptor("FromQA");

             HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResIndRefundCC);

                 /**********************   REQUEST  ************************/

                try
                {
                         Receipt receipt = mpgReq.GetReceipt();

                         Console.WriteLine("DataKey = " + receipt.GetDataKey());
                         Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                         Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                         Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                         Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                         Console.WriteLine("Message = " + receipt.GetMessage());
                         Console.WriteLine("TransDate = " + receipt.GetTransDate());
                         Console.WriteLine("TransTime = " + receipt.GetTransTime());
                         Console.WriteLine("TransType = " + receipt.GetTransType());
                         Console.WriteLine("Complete = " + receipt.GetComplete());
                         Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                         Console.WriteLine("CardType = " + receipt.GetCardType());
                         Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                         Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                         Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                         Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                         //ResolveData
                         Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                         Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                         Console.WriteLine("Email = " + receipt.GetResDataEmail());
                         Console.WriteLine("Note = " + receipt.GetResDataNote());
                         Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                         Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                         Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                         Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                         Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                         Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
           }

         } // end TestDrive Item
}
```

ResIndRefundAch

The ResIndRefundAch will credit a specified amount directly to the customer's bank account. Required fields for this transaction are: store_id, api_token, data_key, order_id, and amount. Optional variable is cust_id. This transaction will be processed as an ACHCredit. The ACHInfo registered for this profile will be used. The details submitted within 'ACHInfo' will be returned in the response within ResolveData.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResIndRefundAch
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "A9C7T91wnf43114R7T30BT2";
                string order_id = "res_ind_refund_1";
                string amount = "1.00";
                string cust_id = "customer1";

                USResIndRefundAch usResIndRefundAch = new USResIndRefundAch(data_key, order_id, cust_id, amount);
                HttpsPostRequest mpgReq=new HttpsPostRequest(host, store_id, api_token, usResIndRefundAch);

                try {
                      Receipt receipt = mpgReq.GetReceipt();
                      Console.WriteLine("DataKey = " + receipt.GetDataKey());
                      Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                      Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                      Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                      Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                      Console.WriteLine("Message = " + receipt.GetMessage());
                      Console.WriteLine("TransDate = " + receipt.GetTransDate());
                      Console.WriteLine("TransTime = " + receipt.GetTransTime());
                      Console.WriteLine("TransType = " + receipt.GetTransType());
                      Console.WriteLine("Complete = " + receipt.GetComplete());
                      Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                      Console.WriteLine("CardType = " + receipt.GetCardType());
                      Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                      Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                      Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                      Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
                      //ResolveData
                      Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                      Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                      Console.WriteLine("Email = " + receipt.GetResDataEmail());
                      Console.WriteLine("Note = " + receipt.GetResDataNote());
                      Console.WriteLine("Sec = " + receipt.GetResDataSec());
                      Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                      Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                      Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                      Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                      Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                      Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                      Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                      Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                      Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                      Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                      Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
               }
               catch (Exception e)
                {
                        Console.WriteLine(e);
                }
          }
        }
}
```

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, verifying the card verification digit (CVD), verifying the address via address verification (AVS), and providing details for the Recurring Billing feature. AVS, CVD, and Recurring Billing must be added to your account, please call the Service Centre at 1-866-423-8475 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.***

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestResPurchaseCC
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
                string order_id = "qa31050021";
                string amount = "1.00";
                string cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be used
                string crypt_type = "1";
                string commcard_invoice = "invoice";
                string commcard_tax_amount = "1.00";

                USResPurchaseCC usResPurchaseCC = new USResPurchaseCC(data_key, order_id, cust_id, amount, crypt_type);

                /**********************   OPTIONAL VARIABLES  ************************/
                usResPurchaseCC.SetCommcardInvoice(commcard_invoice);
                usResPurchaseCC.SetCommcardTaxAmount(commcard_tax_amount);

                //CustInfo Variables
                CustInfo custInfo = new CustInfo();

                custInfo.SetEmail("nick@widget.com");
                custInfo.SetInstructions("Make it fast!");


                Hashtable b = new Hashtable();

                b.Add("first_name", "Bob");
                b.Add("last_name", "Smith");
                b.Add("company_name", "Widget Company Inc.");
                b.Add("address", "111 Bolts Ave.");
                b.Add("city", "Toronto");
                b.Add("province", "Ontario");
                b.Add("postal_code", "M8T 1T8");
                b.Add("country", "Canada");
                b.Add("phone", "416-555-5555");
                b.Add("fax", "416-555-5555");
                b.Add("tax1", "123.45");       //federal tax
                b.Add("tax2", "12.34");        //prov tax
                b.Add("tax3", "15.45");        //luxury tax
                b.Add("shipping_cost", "456.23");   //shipping cost

                custInfo.SetBilling(b);

                /* OR you can pass the individual args.
                custInfo.SetBilling(
                              "Bob",                   //first name
                              "Smith",                 //last name
                              "Widget Company Inc.",   //company name
                              "111 Bolts Ave.",        //address
                              "Toronto",               //city
```

```
                   "Ontario",               //province
                   "M8T 1T8",               //postal code
                   "Canada",                //country
                   "416-555-5555",          //phone
                   "416-555-5555",          //fax
                   "123.45",                //federal tax
                   "12.34",                 //prov tax
                   "15.45",                 //luxury tax
                   "456.23"                 //shipping cost
    );
    */

    Hashtable s = new Hashtable();

    s.Add("first_name", "Bob");
    s.Add("last_name", "Smith");
    s.Add("company_name", "Widget Company Inc.");
    s.Add("address", "111 Bolts Ave.");
    s.Add("city", "Toronto");
    s.Add("province", "Ontario");
    s.Add("postal_code", "M8T 1T8");
    s.Add("country", "Canada");
    s.Add("phone", "416-555-5555");
    s.Add("fax", "416-555-5555");
    s.Add("tax1", "123.45");       //federal tax
    s.Add("tax2", "12.34");        //prov tax
    s.Add("tax3", "15.45");        //luxury tax
    s.Add("shipping_cost", "456.23");   //shipping cost

    custInfo.SetShipping(s);

    /* OR you can pass the individual args.
    custInfo.SetShipping(
                   "Bob",                   //first name
                   "Smith",                 //last name
                   "Widget Company Inc.",   //company name
                   "111 Bolts Ave.",        //address
                   "Toronto",               //city
                   "Ontario",               //province
                   "M8T 1T8",               //postal code
                   "Canada",                //country
                   "416-555-5555",          //phone
                   "416-555-5555",          //fax
                   "123.45",                //federal tax
                   "12.34",                 //prov tax
                   "15.45",                 //luxury tax
                   "456.23"                 //shipping cost
    );
    */

    Hashtable i1 = new Hashtable();

    i1.Add("name", "item1's name");
    i1.Add("quantity", "5");
    i1.Add("product_code", "item1's product code");
    i1.Add("extended_amount", "1.01");

    custInfo.SetItem(i1);

    /* OR you can pass the individual args.
    custInfo.SetItem(
        "item1's name",         //name
        "5",                    //quantity
        "item1's product code", //product code
        "1.01"                  //extended amount
    );
    */

    Hashtable i2 = new Hashtable();

    i2.Add("name", "item2's name");
    i2.Add("quantity", "7");
    i2.Add("product_code", "item2's product code");
    i2.Add("extended_amount", "5.01");

    custInfo.SetItem(i2);

    usResPurchaseCC.SetCustInfo(custInfo);

    usResPurchaseCC.SetDynamicDescriptor("FromQA");

    HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseCC);
```

```
                /**********************   REQUEST  ************************/

                try
                {
                        Receipt receipt = mpgReq.GetReceipt();

                        Console.WriteLine("DataKey = " + receipt.GetDataKey());
                        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                        Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                        Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                        Console.WriteLine("Message = " + receipt.GetMessage());
                        Console.WriteLine("TransDate = " + receipt.GetTransDate());
                        Console.WriteLine("TransTime = " + receipt.GetTransTime());
                        Console.WriteLine("TransType = " + receipt.GetTransType());
                        Console.WriteLine("Complete = " + receipt.GetComplete());
                        Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                        Console.WriteLine("CardType = " + receipt.GetCardType());
                        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                        Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                        Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                        //ResolveData
                        Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                        Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                        Console.WriteLine("Email = " + receipt.GetResDataEmail());
                        Console.WriteLine("Note = " + receipt.GetResDataNote());
                        Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                        Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                        Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                        Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                        Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                        Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        } // end TestDrive Item
}
```

ResPurchaseAch (with Customer and Order details)

This transaction is processed as an ResPurchaseAch transaction with CustInfo attached.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResPurchaseAch
         {
           public static void Main(string[] args)
           {

                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa002";
                  string api_token = "qatoken";
                  string data_key = "1sZH2p215HdQV616909x217";
                  string order_id = "res_puchase_ach_2";
                  string amount = "1.00";
                  string cust_id = "customer1";

                  USResPurchaseAch usResPurchaseAch = new USResPurchaseAch(data_key, order_id, cust_id, amount);

                  //CustInfo Variables
                  CustInfo custInfo = new CustInfo();

                  custInfo.SetEmail("nick@widget.com");
                  custInfo.SetInstructions("Make it fast!");

                  Hashtable b = new Hashtable();

                  b.Add("first_name", "Bob");
                  b.Add("last_name", "Smith");
                  b.Add("company_name", "Widget Company Inc.");
                  b.Add("address", "111 Bolts Ave.");
                  b.Add("city", "Toronto");
                  b.Add("province", "Ontario");
                  b.Add("postal_code", "M8T 1T8");
                  b.Add("country", "Canada");
                  b.Add("phone", "416-555-5555");
                  b.Add("fax", "416-555-5555");
                  b.Add("tax1", "123.45");       //federal tax
                  b.Add("tax2", "12.34");        //prov tax
                  b.Add("tax3", "15.45");        //luxury tax
                  b.Add("shipping_cost", "456.23");   //shipping cost

                  custInfo.SetBilling(b);

                  /* OR you can pass the individual args.
                  custInfo.SetBilling(
                                "Bob",                  //first name
                                "Smith",                //last name
                                "Widget Company Inc.",  //company name
                                "111 Bolts Ave.",       //address
                                "Toronto",              //city
                                "Ontario",              //province
                                "M8T 1T8",              //postal code
                                "Canada",               //country
                                "416-555-5555",         //phone
                                "416-555-5555",         //fax
                                "123.45",               //federal tax
                                "12.34",                //prov tax
                                "15.45",                //luxury tax
                                "456.23"                //shipping cost
                  );
                  */

                  Hashtable s = new Hashtable();

                  s.Add("first_name", "Bob");
                  s.Add("last_name", "Smith");
                  s.Add("company_name", "Widget Company Inc.");
                  s.Add("address", "111 Bolts Ave.");
                  s.Add("city", "Toronto");
                  s.Add("province", "Ontario");
                  s.Add("postal_code", "M8T 1T8");
                  s.Add("country", "Canada");
                  s.Add("phone", "416-555-5555");
```

```
s.Add("fax", "416-555-5555");
s.Add("tax1", "123.45");       //federal tax
s.Add("tax2", "12.34");        //prov tax
s.Add("tax3", "15.45");        //luxury tax
s.Add("shipping_cost", "456.23");   //shipping cost

custInfo.SetShipping(s);

/* OR you can pass the individual args.
custInfo.SetShipping(
              "Bob",                   //first name
              "Smith",                 //last name
              "Widget Company Inc.",  //company name
              "111 Bolts Ave.",        //address
              "Toronto",               //city
              "Ontario",               //province
              "M8T 1T8",               //postal code
              "Canada",                //country
              "416-555-5555",          //phone
              "416-555-5555",          //fax
              "123.45",                //federal tax
              "12.34",                 //prov tax
              "15.45",                 //luxury tax
              "456.23"                 //shipping cost
);
*/

Hashtable i1 = new Hashtable();

i1.Add("name", "item1's name");
i1.Add("quantity", "5");
i1.Add("product_code", "item1's product code");
i1.Add("extended_amount", "1.01");

custInfo.SetItem(i1);

/* OR you can pass the individual args.
custInfo.SetItem(
                 "item1's name",         //name
                 "5",                    //quantity
                 "item1's product code", //product code
                 "1.01"                  //extended amount
);
*/

Hashtable i2 = new Hashtable();

i2.Add("name", "item2's name");
i2.Add("quantity", "7");
i2.Add("product_code", "item2's product code");
i2.Add("extended_amount", "5.01");

custInfo.SetItem(i2);

usResPurchaseAch.SetCustInfo(custInfo);


HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseAch);


try
{
 Receipt receipt = mpgReq.GetReceipt();

Console.WriteLine("DataKey = " + receipt.GetDataKey());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("AVSResponse = " + receipt.GetAvsResultCode());
Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

//ResolveData
Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
```

```
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("Sec = " + receipt.GetResDataSec());
                 Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                 Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                 Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                 Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                 Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                 Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                 Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                 Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
                 Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                 Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                 Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());

                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        }
}
```

ResPurchasePinless (with Customer and Order details)

This transaction will perform a USResPurchasePinless with customer information added.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;
        public class TestResPurchasePinless
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "UIV4907Qf1Er2198R951h0B";
                string order_id = "res_purchase_pinless_10";
                string amount = "1.00";
                string cust_id = "customer1";
                string intended_use = "1";
                string p_account_number = "23456789";

                USResPurchasePinless usResPurchasePinless = new USResPurchasePinless(data_key, order_id, cust_id,
                amount, intended_use, p_account_number);

                //CustInfo Variables
                CustInfo custInfo = new CustInfo();

                custInfo.SetEmail("nick@widget.com");
                custInfo.SetInstructions("Make it fast!");

                Hashtable b = new Hashtable();

                b.Add("first_name", "Bob");
                b.Add("last_name", "Smith");
                b.Add("company_name", "Widget Company Inc.");
                b.Add("address", "111 Bolts Ave.");
                b.Add("city", "Toronto");
                b.Add("province", "Ontario");
                b.Add("postal_code", "M8T 1T8");
                b.Add("country", "Canada");
                b.Add("phone", "416-555-5555");
                b.Add("fax", "416-555-5555");
                b.Add("tax1", "123.45");       //federal tax
                b.Add("tax2", "12.34");        //prov tax
                b.Add("tax3", "15.45");        //luxury tax
                b.Add("shipping_cost", "456.23");   //shipping cost

                custInfo.SetBilling(b);

                /* OR you can pass the individual args.
                custInfo.SetBilling(
                              "Bob",                   //first name
                              "Smith",                 //last name
                              "Widget Company Inc.",   //company name
                              "111 Bolts Ave.",        //address
                              "Toronto",               //city
                              "Ontario",               //province
                              "M8T 1T8",               //postal code
                              "Canada",                //country
                              "416-555-5555",          //phone
                              "416-555-5555",          //fax
                              "123.45",                //federal tax
                              "12.34",                 //prov tax
                              "15.45",                 //luxury tax
                              "456.23"                 //shipping cost
                );
                */

                Hashtable s = new Hashtable();

                s.Add("first_name", "Bob");
                s.Add("last_name", "Smith");
                s.Add("company_name", "Widget Company Inc.");
                s.Add("address", "111 Bolts Ave.");
                s.Add("city", "Toronto");
                s.Add("province", "Ontario");
                s.Add("postal_code", "M8T 1T8");
                s.Add("country", "Canada");
```

```
s.Add("phone", "416-555-5555");
s.Add("fax", "416-555-5555");
s.Add("tax1", "123.45");       //federal tax
s.Add("tax2", "12.34");        //prov tax
s.Add("tax3", "15.45");        //luxury tax
s.Add("shipping_cost", "456.23");   //shipping cost

custInfo.SetShipping(s);

/* OR you can pass the individual args.
custInfo.SetShipping(
              "Bob",                  //first name
              "Smith",                //last name
              "Widget Company Inc.",  //company name
              "111 Bolts Ave.",       //address
              "Toronto",              //city
              "Ontario",              //province
              "M8T 1T8",              //postal code
              "Canada",               //country
              "416-555-5555",         //phone
              "416-555-5555",         //fax
              "123.45",               //federal tax
              "12.34",                //prov tax
              "15.45",                //luxury tax
              "456.23"                //shipping cost
);
*/

Hashtable i1 = new Hashtable();

i1.Add("name", "item1's name");
i1.Add("quantity", "6");
i1.Add("product_code", "item1's product code");
i1.Add("extended_amount", "1.01");

custInfo.SetItem(i1);

/* OR you can pass the individual args.
custInfo.SetItem(
                 "item1's name",         //name
                 "5",                    //quantity
                 "item1's product code", //product code
                 "1.01"                  //extended amount
);
*/

Hashtable i2 = new Hashtable();

i2.Add("name", "item2's name");
i2.Add("quantity", "7");
i2.Add("product_code", "item2's product code");
i2.Add("extended_amount", "5.01");

custInfo.SetItem(i2);

usResPurchasePinless.SetCustInfo(custInfo);

HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchasePinless);



try
{
Receipt receipt = mpgReq.GetReceipt();

Console.WriteLine("DataKey = " + receipt.GetDataKey());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

//ResolveData
Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
```

```
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                 Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                 Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                 Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());

                }
                catch (Exception e)
                {
                        Console.WriteLine(e);
                }
          }

        }
}
```

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 1-866-423-8475 to have your profile updated.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestResPurchaseCC
        {
          public static void Main(string[] args)
          {

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
                string order_id = "qa31050025";
                string amount = "1.00";
                string cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be used
                string crypt_type = "2";
                string commcard_invoice = "invoice";
                string commcard_tax_amount = "1.00";

                USResPurchaseCC usResPurchaseCC = new USResPurchaseCC(data_key, order_id, cust_id, amount, crypt_type);

                /**********************   OPTIONAL VARIABLES  ************************/
                usResPurchaseCC.SetCommcardInvoice(commcard_invoice);
                usResPurchaseCC.SetCommcardTaxAmount(commcard_tax_amount);

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

                string recur_unit = "month";   //valid values are (day,week,month,eom)
                string start_now = "true";
                string start_date = "2011/12/01";
                string num_recurs = "12";
                string period = "1";
                string recur_amount = "30.00";

                /************************* Recur Object Option1 ******************************/

                Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_recurs, period, recur_amount);

                usResPurchaseCC.SetRecur(recurring_cycle);

                usResPurchaseCC.SetDynamicDescriptor("FromQA");

                /**********************   REQUEST  ************************/
                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseCC);

               try
               {
                        Receipt receipt = mpgReq.GetReceipt();

                        Console.WriteLine("DataKey = " + receipt.GetDataKey());
                        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                        Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                        Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                        Console.WriteLine("Message = " + receipt.GetMessage());
                        Console.WriteLine("TransDate = " + receipt.GetTransDate());
                        Console.WriteLine("TransTime = " + receipt.GetTransTime());
                        Console.WriteLine("TransType = " + receipt.GetTransType());
                        Console.WriteLine("Complete = " + receipt.GetComplete());
                        Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                        Console.WriteLine("CardType = " + receipt.GetCardType());
                        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                        Console.WriteLine("AVSResponse = " + receipt.GetAvsResultCode());
                        Console.WriteLine("CVDResponse = " + receipt.GetCvdResultCode());
                        Console.WriteLine("RecurSuccess = " + receipt.GetRecurSuccess());
                        Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                        Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                        //ResolveData
                        Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                        Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                        Console.WriteLine("Email = " + receipt.GetResDataEmail());
                        Console.WriteLine("Note = " + receipt.GetResDataNote());
                        Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                        Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                        Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                        Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                        Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                        Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                        Console.WriteLine(e);
                }
           }

        } // end TestDrive Item
}
```

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.

ResPurchaseAch (with Recurring Billing)

This transaction is processed as a ResPurchaseAch with recur.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResPurchaseAch
        {
          public static void Main(string[] args)
          {

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "1sZH2p215HdQV6l6909x217";
                string order_id = "res_puchase_ach_3";
                string amount = "1.00";
                string cust_id = "customer1";

                USResPurchaseAch usResPurchaseAch = new USResPurchaseAch(data_key, order_id, cust_id, amount);

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

                string recur_unit = "month";
                string start_now = "true";
                string start_date = "2008/12/01";
                string num_recurs = "12";
                string period = "1";
                string recur_amount = "30.00";

                /************************* Recur Object Option1 ******************************/

                Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_recurs, period, recur_amount);

                usResPurchaseAch.SetRecur(recurring_cycle);

                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseAch);

                /**********************   REQUEST  *****************************************/
                try
                {
                 Receipt receipt = mpgReq.GetReceipt();

                Console.WriteLine("DataKey = " + receipt.GetDataKey());
                Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("TransType = " + receipt.GetTransType());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                Console.WriteLine("CardType = " + receipt.GetCardType());
                Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("RecurSuccess = " + receipt.GetRecurSuccess());
                Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("Sec = " + receipt.GetResDataSec());
                Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
                Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
                Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
                Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
                Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
                Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
                Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
                Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
```

```
                 Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
                 Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
                 Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());

                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        }
}
```

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.

ResPurchasePinless (with Recurring Billing)

This transaction will process a ResPurchasePinless with recur.

```
namespace USMoneris
{
        using System;
        using System.Text;
        using System.Collections;

        public class TestResPurchasePinless
        {
          public static void Main(string[] args)
          {

                 string host = "esplusqa.moneris.com";
                 string store_id = "monusqa002";
                 string api_token = "qatoken";
                 string data_key = "UIV4907Qf1Er2198R951h0B";
                         string order_id = "res_purchase_pinless_3";
                 string amount = "1.00";
                 string cust_id = "customer1";
                         string intended_use = "1";
                 string p_account_number = "23456789";

                         USResPurchasePinless usResPurchasePinless = new USResPurchasePinless(data_key, order_id,
                         cust_id, amount, intended_use, p_account_number);

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

                 string recur_unit = "month";
                 string start_now = "true";
                 string start_date = "2008/12/01";
                 string num_recurs = "12";
                 string period = "1";
                 string recur_amount = "30.00";

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

                 Recur recurring_cycle = new Recur(recur_unit, start_now, start_date,
                                   num_recurs, period, recur_amount);

                 usResPurchasePinless.SetRecur(recurring_cycle);

                 HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchasePinless);

                 /**********************   REQUEST  *******************************************/

                try
                {
                         Receipt receipt = mpgReq.GetReceipt();

                 Console.WriteLine("DataKey = " + receipt.GetDataKey());
                 Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                 Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("TransType = " + receipt.GetTransType());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                 Console.WriteLine("CardType = " + receipt.GetCardType());
                 Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("RecurSuccess = " + receipt.GetRecurSuccess());
                 Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                 Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                 //ResolveData
                 Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                 Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                 Console.WriteLine("Email = " + receipt.GetResDataEmail());
                 Console.WriteLine("Note = " + receipt.GetResDataNote());
                 Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                 Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                 Console.WriteLine("Presentation Type = " + receipt.GetResDataPresentationType());
                 Console.WriteLine("P Account Number = " + receipt.GetResDataPAccountNumber());

                }
                catch (Exception e)
                {
                        Console.WriteLine(e);
                }
          }

        }
}
```

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 form CvdInfo please refer to, to form AvsInfo please refer to
To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

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

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

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.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestResPurchaseCC
        {
          public static void Main(string[] args)
          {

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa002";
                string api_token = "qatoken";
                string data_key = "GhOKrxgum6Qwxo39KNt8aTuS5";
                        string order_id = "qa31050020";
                string amount = "1.00";
                string cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be used
                string crypt_type = "1";
                string commcard_invoice = "invoice";
                string commcard_tax_amount = "1.00";

                USResPurchaseCC usResPurchaseCC = new USResPurchaseCC(data_key, order_id, cust_id, amount, crypt_type);

                /**********************   OPTIONAL VARIABLES  ***********************/
                usResPurchaseCC.SetCommcardInvoice(commcard_invoice);
                usResPurchaseCC.SetCommcardTaxAmount(commcard_tax_amount);

                /*************** Address Verification Service **********************/

                AvsInfo avsCheck = new AvsInfo();

                avsCheck.SetAvsStreetNumber("212");
                avsCheck.SetAvsStreetName("Payton Street");
                avsCheck.SetAvsZipCode("M1M1M1");

                usResPurchaseCC.SetAvsInfo(avsCheck);

                /****************** Card Validation Digits *************************/

                CvdInfo cvdCheck = new CvdInfo();
                cvdCheck.SetCvdIndicator("1");
                cvdCheck.SetCvdValue("099");
                usResPurchaseCC.SetCvdInfo(cvdCheck);

                usResPurchaseCC.SetDynamicDescriptor("FromQA");

                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseCC);

                /**********************   REQUEST  ************************/

                try
                {
                         Receipt receipt = mpgReq.GetReceipt();

                         Console.WriteLine("DataKey = " + receipt.GetDataKey());
                         Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                         Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                         Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                         Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                         Console.WriteLine("Message = " + receipt.GetMessage());
                         Console.WriteLine("TransDate = " + receipt.GetTransDate());
                         Console.WriteLine("TransTime = " + receipt.GetTransTime());
                         Console.WriteLine("TransType = " + receipt.GetTransType());
                         Console.WriteLine("Complete = " + receipt.GetComplete());
                         Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                         Console.WriteLine("CardType = " + receipt.GetCardType());
                         Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                         Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                         Console.WriteLine("AVSResponse = " + receipt.GetAvsResultCode());
                         Console.WriteLine("CVDResponse = " + receipt.GetCvdResultCode());
                         Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                         Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                         //ResolveData
                         Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                         Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                         Console.WriteLine("Email = " + receipt.GetResDataEmail());
                         Console.WriteLine("Note = " + receipt.GetResDataNote());
                         Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
                         Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                         Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                         Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                         Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                         Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
           }

        } // end TestDrive Item
}
```

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

8. 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 US 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 US 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 US Hosted Tokenization feature please see the US Hosted Tokenization Integration Guide. The guide can be downloaded from the Moneris Developer Portal: https://developer.moneris.com

9. 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 4 above for these transaction examples.

Example Temporary Token Purchase

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

         public class TestResPurchaseCC
         {
           public static void Main(string[] args)
           {

                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa002";
                  string api_token = "qatoken";

                  string data_key = "b7I1EasRbIx5bvlkaSYxxtMsE";
                  string order_id = "dec16test1";
                  string amount = "1.00";
                  string cust_id = "customer1"; //if sent will be submitted, otherwise cust_id from profile will be used
                  string crypt_type = "1";
                  string commcard_invoice = "invoice";
                  string commcard_tax_amount = "1.00";
                  string dynamic_descriptor = "123456";

        USResPurchaseCC usResPurchaseCC = new USResPurchaseCC(data_key, order_id, cust_id, amount, crypt_type);

        /**********************   OPTIONAL VARIABLES  ***********************/
        usResPurchaseCC.SetCommcardInvoice(commcard_invoice);
        usResPurchaseCC.SetCommcardTaxAmount(commcard_tax_amount);
        usResPurchaseCC.SetDynamicDescriptor(dynamic_descriptor);

                  //usResPurchaseCC.SetExpdate("1601"); //must be YYMM format

        HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResPurchaseCC);

                  /**********************   REQUEST  ***********************/
```

```
                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

               Console.WriteLine("DataKey = " + receipt.GetDataKey());
               Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
               Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
               Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
               Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
               Console.WriteLine("Message = " + receipt.GetMessage());
               Console.WriteLine("TransDate = " + receipt.GetTransDate());
               Console.WriteLine("TransTime = " + receipt.GetTransTime());
               Console.WriteLine("TransType = " + receipt.GetTransType());
               Console.WriteLine("Complete = " + receipt.GetComplete());
               Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
               Console.WriteLine("CardType = " + receipt.GetCardType());
               Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
               Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
               Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
               Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

               //ResolveData
               Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
               Console.WriteLine("Phone = " + receipt.GetResDataPhone());
               Console.WriteLine("Email = " + receipt.GetResDataEmail());
               Console.WriteLine("Note = " + receipt.GetResDataNote());
               Console.WriteLine("Masked Pan = " + receipt.GetResDataMaskedPan());
               Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
               Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
               Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
               Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
               Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());


                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        }
}
```

10. 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 US 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.

```
namespace USMoneris
{
    using System;
    using System.Text;
    using System.Collections;

        public class TestResAddToken
        {
          public static void Main(string[] args)
          {

                 string host = "esplusqa.moneris.com";
        string store_id = "monusqa002";
        string api_token = "qatoken";

        string data_key = "ot-QOAnydyWYYw79pApkS5ROBzX9";
        string phone = "0000000000";
        string email = "bob@smith.com";
        string note = "my note";
        string cust_id = "customer1";
        string crypt_type = "7";

        AvsInfo avsCheck = new AvsInfo();
        avsCheck.SetAvsStreetNumber("212");
        avsCheck.SetAvsStreetName("Payton Street");
        avsCheck.SetAvsZipCode("M1M1M1");


        USResAddToken usResAddToken = new USResAddToken(data_key, crypt_type);

        //***********************OPTIONAL VARIABLES***************************

        usResAddToken.SetCustId(cust_id);
        usResAddToken.SetPhone(phone);
        usResAddToken.SetEmail(email);
        usResAddToken.SetNote(note);
        usResAddToken.SetAvsInfo(avsCheck);

        HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usResAddToken);

                 /**********************   REQUEST  ************************/

                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

                Console.WriteLine("DataKey = " + receipt.GetDataKey());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
                Console.WriteLine("PaymentType = " + receipt.GetPaymentType());

                //ResolveData
                Console.WriteLine("\nCust ID = " + receipt.GetResDataCustId());
                Console.WriteLine("Phone = " + receipt.GetResDataPhone());
                Console.WriteLine("Email = " + receipt.GetResDataEmail());
                Console.WriteLine("Note = " + receipt.GetResDataNote());
                Console.WriteLine("MaskedPan = " + receipt.GetResDataMaskedPan());
                Console.WriteLine("Exp Date = " + receipt.GetResDataExpdate());
                Console.WriteLine("Crypt Type = " + receipt.GetResDataCryptType());
                Console.WriteLine("Avs Street Number = " + receipt.GetResDataAvsStreetNumber());
                Console.WriteLine("Avs Street Name = " + receipt.GetResDataAvsStreetName());
                Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        }
}
```

11. How Do I Test My Solution?

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

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

Test IDs			
store_id	**api_token**	**Username**	**Password**
monusqa002*	qatoken	demouser	abc1234
monusqa003	qatoken	demouser	abc1234
monusqa004	qatoken	demouser	abc1234

monusqa005	qatoken	demouser	abc1234
monusqa006	qatoken	demouser	abc1234

* test store 'monusqa002' is intended for testing the Pinless Debit transactions

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

Test Card Numbers	
Card Plan	**Card Number**
MasterCard	5454545454545454
Visa	4242424242424242 or 4005554444444403
Amex	373599005095005
Pinless Debit	4496270000164824

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

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

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

The test environment will approve and decline credit card transactions based on the penny value of the amount field.
For example, a transaction made for the amount of $9.00 or $1.00 will approve since the .00 penny value is set to approve in the test environment. Transactions in the test environment should not exceed $11.00. This limit does not exist in the production environment. For a list of all current test environment responses for various penny values, please see the Test Environment Penny Response table as well as the Test Environment eFraud Response table, available at https://developer.moneris.com

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

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

12. How Do I Get Help?

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

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

For technical 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.**

13. Appendix A. Definition of Request Fields

Request Fields		
Variable Name	**Size/Type**	**Description**
order_id	50 / an	Merchant defined unique transaction identifier - must be unique for every ResPurchase, ResPreAuth and ResIndRefund attempt. Characters allowed for Order ID: **a-z A-Z 0-9 _ - : . @ spaces**
data_key	23 / 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, ResAddACH or ResAddPinless transactions.
pan	20 / variable	Credit or Pinless Debit Card Number - no spaces or dashes. Most credit/pinless debit 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
enc_track2		This is a string that is retrieved by swiping or keying in a credit card through a Moneris provided encrypted mag swipe card reader. It is part of an encrypted keyed or swiped transaction only. This string must be retrieved by a specific device. Please refer to device_type for the list of current available devices.
device_type	an	Defines the encrypted mag swipe reader that was used for swiping or keying in the credit card. Plesase note, this device must be provided by Moneris Solutions so that the values are properly encrypted and decrypted. This field is case sensitive. Available values are: device_type="idtech"
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.
intended_use	1 / num	Identifies the party who initiated the transaction. - "0" = Merchant initiated the payment - "1" = Customer initiated the payment
p_account_number	25 / an	The billing invoice number – no spaces or dashes. The length of the account number varies with a maximum length of 25 digits. This is field is mandatory to properly process a Pinless Debit financial transaction. It must either be registered in the profile or submitted at the time of the ResPurchasePinless transaction.
presentation_type	1 / alpha	Identifies how merchants obtain the Pinless Debit account. This field is a mandatory field required when adding the Pinless Debit profile. - 'X' for Telephone/VRU - 'W' for Internet
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 **Error! Reference source not found.** for further details. Note: The CVD value supplied by the cardholder should simply be passed to the eSELECTplus payment gateway. Under no circumstances should it be stored for subsequent uses or displayed as part of the receipt information.
cvd_indicator	1 / num	CVD presence indicator (1 digit – refer to **Error! Reference source not found.** for values)
commcard_invoice	17 / an	Level 2 Invoice Number for the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). Characters allowed for commcard_invoice: **a-z A-Z 0-9 spaces**
commcard_tax_amount	9 / decimal	Level 2 Tax Amount of the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum is 9999999.99
dynamic_descriptor	20 / an	Merchant defined description sent on a per-transaction basis that will appear on the credit card statement. Dependent on the card Issuer, the statement will typically show the dynamic desciptor appended to the merchant's existing business name separated by the "/" character. Please note that the combined length of the merchant's business name, forward slash "/" character, and the dynamic descriptor may not exceed 22 characters.

NOTE

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

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

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

14. Appendix B. Definitions of Response Fields

Response Fields		
Variable Name	**Size/Type**	**Description**
ReceiptId	50 / an	order_id specified in request
ReferenceNum	18 / num	The reference number is an 18 character string that references the terminal used to process the transaction as well as the shift, batch and sequence number, This data is typically used to reference transactions on the host systems and must be displayed on any receipt presented to the customer. This information should be stored by the merchant. The following illustrates the breakdown of this field where "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 at https://developer.moneris.com Vault Admin Responses (i.e. ResAdd or ResDelete) 001 Successfully registered (CC|ACH|Pinless) details. Successfully updated (CC|ACH|Pinless) details. Successfully deleted (CC|ACH|Pinless) details. Successfully located (CC|ACH|Pinless) 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 Refer to **Error! Reference source not found.**the developer portal.
CvdResultCode	2/an	Indicates the CVD validation result. Refer to **Error! Reference source not found.**the developer portal
ResSuccess	true/false	Indicates if Vault transaction was successful.
PaymentType	cc|ach|pinless	Indicates the payment type associated with a Vault profile.
DataKey	23 / 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.
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, presentation_type, p_account_number, sec, cust_first_name, cust_last_name, cust_address1, cust_address2, cust_city, cust_state, cust_zip, routing_num, masked_account_num, account_num, check_num, and account_type.

15. Appendix C. CustInfo Fields

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

NOTE

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

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

Also, the data sent in Billing and Shipping Address fields will not be used for any address verification. Please refer to the section 7 for further details about Address Verification Service (AVS).

16. 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**

string data_key = "3ixBzo1e25Zck8urjLcTbuu22";
string order_id = "monthly_purchase";
string amount = "15.00";
string crypt_type = "7";
 	
USResPurchaseCC usResPurchaseCC = new
USResPurchaseCC(data_key, order_id, cust_id,
amount, crypt_type);
 	
string recur_unit = "month";
string start_now = "false";
string start_date = "2007/01/02";
string num_recurs = "12";
string period = "2";
string recur_amount = "30.00";
 	
Recur recurring_cycle = new Recur(recur_unit,
start_now, start_date, num_recurs, period,
recur_amount);
	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.

string data_key = "3ixBzo1e25Zck8urjLcTbuu22";
string order_id = "bi-weekly_purchase";
string amount = "15.00";
string crypt_type = "7";
 	
USResPurchaseCC usResPurchaseCC = new
USResPurchaseCC(data_key, order_id, cust_id,
amount, crypt_type);
 	
string recur_unit = "week";
string start_now = "true";
string start_date = "2007/01/02";
string num_recurs = "26";
string period = "2";
string recur_amount = "30.00";
 	
Recur recurring_cycle = new Recur(recur_unit,
start_now, start_date, num_recurs, period,
recur_amount);
	In the example on the left the first charge will be billed immediately. The initial charge will be for $15.00. Then starting on Jan 2nd 2007 the credit card will be billed $30.00 every 2 weeks for 26 recurring charges. The card will be billed a total of 27 times. (1 x $15.00 (immediate) and 26 x $30.00 (recurring))

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

17. Appendix E. AchInfo Fields

AchInfo Request Fields		
Variable Name	**Size/Type**	**Description**
sec	3 / an	ACH SEC Code: ppd - Prearranged Payment and Deposit ccd - Cash Concentration or Disbursement web - Internet Initiated Entry * only PPD and CCD apply to the ResIndRefundAch transaction
routing_num	9 / num	The first number in the MICR, or magnetic ink character recognition, line at the bottom of a check is the bank's check routing number. It is exactly nine digits long and always starts with 0, 1, 2 or 3.
account_num	50 / num	The account number may appear before or after the check number in the check's MICR line at the bottom of the check. The length of the account number varies with a maximum length of 50 digits.
check_num	16 / num	The sequential number for checks appears in both the MICR line at the bottom of the check and the upper right corner of the check. The check number length may vary; the maximum length is 16 digits. This is an optional field.
account_type	savings / checking	Identifies the type of bank account. The account type must be submitted as either 'savings' or 'checking'. This field is case sensitive.
ACH Customer Information		
NOTE: The following Account Holder information fields are optional.		
cust_first_name	50 / an	
cust_last_name	50 / an	
cust_address1	50 / an	
cust_address2	50 / an	
cust_city	50 / an	
cust_state	2 / alpha	The state must be submitted as exactly 2 characters (ex. MI – Michigan)
cust_zip	15 / an	

NOTE

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

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

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

18. Appendix F. 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.

19. Appendix G. Additional Information for CVD and 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**

20. Appendix H. 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 performed and the form of payment used (i.e. Credit Card vs ACH). For further details on all receipt requirements, please refer to the full .NET API Integration Guide available at: https://developer.moneris.com

eSELECTplus™

Printed in Canada.

10 9 8 7 6 5 4 3 2 1