Merchant Integration Guide

.NET API – Convenience Fees

v 1.1.1

Moneris SOLUTIONS

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

The Convenience Fee program was designed to allow merchants to offer the convenience of an alternative payment channel to the cardholder at a charge. This applies only when providing a true "convenience" in the form of an alternative payment channel outside the merchant's customary face-to-face payment channels. The convenience fee will be a separate charge on top of what the consumer is paying for the goods and/or services they were given, and this charge will appear as a separate line item on the consumer's statement.

This document describes the basic information for using the .NET API for processing Convenience Fee credit card and ACH transactions. In particular, it describes the format for sending transactions with the appropriate convenience fee amount and the corresponding responses you will receive.

2. System and Skill Requirements

In order to use the .NET API 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://esplusqa.moneris.com/connect/en/download/index.php to download the PCI-DSS Implementation Guide.

3. Verified by Visa

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

If you have enrolled in Verified by Visa (VbV) with Moneris and eSELECTplus, please also refer to the *.NET* VbV / SecureCode MPI document available at: https://esplusqa.moneris.com/connect/en/download/index.php

4. MasterCard SecureCode

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

Before approving a transaction, eSELECTplus and the Bank that issued the MasterCard will authenticate the cardholder through the use of this password. For merchants who have enrolled in SecureCode, please also refer to the .NET VbV / SecureCode MPI document available at:
https://esplusqa.moneris.com/connect/en/download/index.php

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

You will need to follow these steps.

1. Do the required development as outlined in this document
2. Test your solution in the test environment
3. Activate your store
4. Make the necessary changes to move your solution from the test environment into production as outlined in this document

6. Transaction Types and Transaction Flow

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

Note: A convenience fee transaction is made up of two amounts.

1. The total transaction amount for the goods or services sold to the cardholder. For the scope of this document this amount will be refered to as 'Transaction Amount'.
2. The convenience fee amount which is the separate charge applied to the transaction. For the scope of this document this amount will be refered to as 'Convenience Fee Amount'.

Basic Transactions

Purchase – (sale) The Purchase transaction verifies funds for both the transaction amount and convenience fee amount on the customer's card. It then removes the funds and readies the transaction amount for deposit into the merchant's account.

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

Refund – (Credit) A Refund can be performed against a Purchase to refund any part, or all of the transaction. The Refund transaction will only credit the transaction amount.

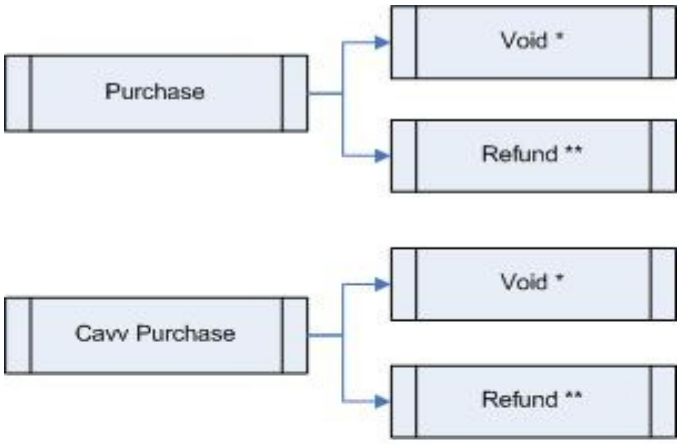
Independent Refund – (Credit) An Independent Refund can be performed to credit money to a Credit Card. This transaction does not require a prior Purchase. Please note, this transaction will only credit funds from the merchant's account.

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

Open Totals – (Current Batch Report) When an Open Totals is performed it returns the details about the currently open Batch. This transaction is similar to the Batch Close, though it does not close the Batch for settlement. Please note, this transaction will only return the details about the merchant's batch.

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

Process Flow for Basic Credit Card Transactions



Transactions with no Follow-on required

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

ACH Transactions

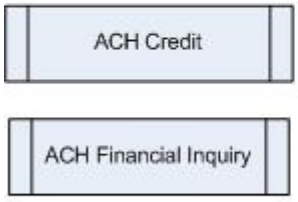
ACH Debit – The ACH Debit transaction verifies and collects the customer's bank account information. It then removes the funds for both the transaction amount and convenience fee amount directly from the customer's bank account and readies them transaction amount for deposit into the merchant's account.

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

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

Process Flow for ACH Transactions

Transactions with no Follow-on required



* Prior or After the Batch closing

7. Basic Transaction Examples

Included below is the sample code that can be found in the "Examples" folder of the Convenience Fee .NET API download.

Purchase (basic)

In the Purchase example we require several variables (store_id, api_token, order_id, amount, pan, expiry_date, and crypt). Also, the ConvFeeInfo object must be properly populated and added to the transaction. For examples on how to calculate this amount, please refer to Section 14 - How Do I Calculate The Convenience Fee Amount?

There are also two optional Level 2 variables (commcard_invoice and commcard_tax_amount) available for Corporate Purchasing Cards. Please refer to Appendix A. Definition of Request Fields for variable definitions.

```
namespace USMoneris
{
    using System;
      public class TestPurchase
      {
        public static void Main(string[] args)
        {
          /****************** REQUEST VARIABLES*******************************/
          string host = "esplusqa.moneris.com";
          string store_id = "monusqa138";
          string api_token = "qatoken";

          /****************** TRANSACTION VARIABLES ****************************/
          string order_id;               //will prompt user for input
          string amount = "1.00";
          string pan = "4242424242424242";
          string expdate = "0812";
          string crypt_type = "7";
          string commcard_invoice = "INVC090";
          string commcard_tax_amount = "1.00";
          Console.Write ("Please enter an order ID: ");
          order_id = Console.ReadLine();

          USPurchase usPurchase = new USPurchase(order_id, amount, pan, expdate, crypt_type, commcard_invoice,
                                          commcard_tax_amount);
          /****************** Convenience Fee ******************************/
          ConvFeeInfo convFeeInfo = new ConvFeeInfo();
          convFeeInfo.SetConvenienceFee("1.00");
          usPurchase.SetConvFeeInfo(convFeeInfo);

          HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, usPurchase);
          try
          {
                Receipt receipt = mpgReq.GetReceipt();
                Console.WriteLine("CardType = " + receipt.GetCardType());
                Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                Console.WriteLine("TransType = " + receipt.GetTransType());
                Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Ticket = " + receipt.GetTicket());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                Console.WriteLine("FeeType = " + receipt.GetFeeType());
          }
          catch (Exception e)
          {
                Console.WriteLine(e);
          }
        }
      }
}
```

As part of the Convenience Fee response there will be five additional reponse methods. Please refer to Appendix B. Definitions of Response Fields for further details.

Void

The Void (USPurchaseCorrection) transaction is used to cancel a transaction that was performed in the current batch. No amount is required because a Void is always for 100% of the original transaction. The only transactions that can be Voided are Purchases. To send a 'USPurchaseCorrection' the order_id and txn_number from the 'USPurchase' are required.

Please note, this transaction type will reverse both the transaction amount and the corresponding convenience fee amount so they do not show on the cardholder's statement.

```
namespace USMoneris
{
    using System;
        public class TestPurchaseCorrection
        {
          public static void Main(string[] args)
          {
                  /****************** REQUEST VARIABLES*****************************/

                  string host = "esplusqa.moneris.com";
                  string store_id = "monusqa138";
                  string api_token = "qatoken";

                  /****************** TRANSACTION VARIABLES ****************************/

                  string order_id;                //will prompt user for input
                  string txn_number;
                  string crypt_type = "7";

                  Console.Write ("Please enter an order ID: ");
                  order_id = Console.ReadLine();

                  Console.Write ("Please enter a txn number: ");
                  txn_number= Console.ReadLine();

                  HttpsPostRequest mpgReq =
                    new HttpsPostRequest(host, store_id, api_token,
                                new USPurchaseCorrection(order_id, txn_number, crypt_type));

               try
               {
                   Receipt receipt = mpgReq.GetReceipt();

                    Console.WriteLine("CardType = " + receipt.GetCardType());
                    Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                    Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                    Console.WriteLine("TransType = " + receipt.GetTransType());
                    Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                    Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                    Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                    Console.WriteLine("Message = " + receipt.GetMessage());
                    Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                    Console.WriteLine("Complete = " + receipt.GetComplete());
                    Console.WriteLine("TransDate = " + receipt.GetTransDate());
                    Console.WriteLine("TransTime = " + receipt.GetTransTime());
                    Console.WriteLine("Ticket = " + receipt.GetTicket());
                    Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                    Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                    Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
               }
               catch (Exception e)
               {
                   Console.WriteLine(e);
               }
          }

        }
}
```

As part of the Convenience Fee response there will be two additional reponse methods. Please refer to Appendix B. Definitions of Response Fields for further details.

Refund

The Refund will credit a specified amount to the cardholder's credit card. A Refund can be sent up to the full value of the original Purchase's transaction amount. To send a 'USRefund' you will require the order_id and txn_number from the original 'USPurchase'.

Please note, this transaction type will credit a partial value, or up to the full value, of the transaction amount from the merchant's bank account to the cardholder. This will not reverse the convenience fee amount.

```
namespace Moneris
{
    using System;
        public class TestRefund
        {
          public static void Main(string[] args)
          {
                /****************** REQUEST VARIABLES*******************************/

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa138";
                string api_token = "qatoken";

                /****************** TRANSACTION VARIABLES ****************************/

                string order_id;          //will prompt user for input
                string amount = "1.00";
                string txn_number;
                string crypt_type = "7";

                Console.Write ("Please enter an order ID: ");
                order_id = Console.ReadLine();

                Console.Write ("Please enter a txn number: ");
                txn_number= Console.ReadLine();


               HttpsPostRequest mpgReq =
                   new HttpsPostRequest(host, store_id, api_token,
                               new USRefund(order_id, amount, txn_number, crypt_type));

               try
               {
                   Receipt receipt = mpgReq.GetReceipt();

                    Console.WriteLine("CardType = " + receipt.GetCardType());
                    Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                    Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                    Console.WriteLine("TransType = " + receipt.GetTransType());
                    Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                    Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                    Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                    Console.WriteLine("Message = " + receipt.GetMessage());
                    Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                    Console.WriteLine("Complete = " + receipt.GetComplete());
                    Console.WriteLine("TransDate = " + receipt.GetTransDate());
                    Console.WriteLine("TransTime = " + receipt.GetTransTime());
                    Console.WriteLine("Ticket = " + receipt.GetTicket());
                    Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                    Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
               }
               catch (Exception e)
               {
                   Console.WriteLine(e);
               }
          }

        }
}
```

As part of the Convenience Fee response there will be one additional reponse method. Please refer to Appendix B. Definitions of Response Fields for further details.

Independent Refund

The Independent Refund (USIndependentRefund) will credit a specified amount from the merchant's bank account to the cardholder's credit card. The Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the credit card number and expiry date will need to be passed. The transaction format is almost identical to a Purchase but it does not require a convenience fee amount.

Please note, the independent refund transaction does not require a convenience fee. This transaction only allows the merchant to transfer funds from their own bank account to the cardholder.

```
namespace USMoneris
{
    using System;
        public class TestIndependentRefund
        {
          public static void Main(string[] args)
          {

                /****************** REQUEST VARIABLES*******************************/

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa138";
                string api_token = "qatoken";

                /****************** TRANSACTION VARIABLES ***************************/

                string order_id;          //will prompt user for input
                string cust_id = "Ced_Benson32";
                string amount = "5.00";
                string pan = "4005554444444403";
                string expdate = "0812";
                string crypt_type= "7";

                Console.Write ("Please enter an order ID: ");
                order_id = Console.ReadLine();

                HttpsPostRequest mpgReq =
                         new HttpsPostRequest(host, store_id, api_token,
                                 new USIndependentRefund(order_id, cust_id, amount, pan, expdate, crypt_type));

                try
                {
                      Receipt receipt = mpgReq.GetReceipt();

                     Console.WriteLine("CardType = " + receipt.GetCardType());
                     Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                     Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                     Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                     Console.WriteLine("TransType = " + receipt.GetTransType());
                     Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                     Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                     Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                     Console.WriteLine("Message = " + receipt.GetMessage());
                     Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                     Console.WriteLine("Complete = " + receipt.GetComplete());
                     Console.WriteLine("TransDate = " + receipt.GetTransDate());
                     Console.WriteLine("TransTime = " + receipt.GetTransTime());
                     Console.WriteLine("Ticket = " + receipt.GetTicket());
                     Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                     Console.WriteLine("CfStatus = " + receipt.GetCfStatus());

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

As part of the Convenience Fee response there will be one additional reponse method. Please refer to Appendix B. Definitions of Response Fields for further details.

8. Basic Transactions with Extra Features - Examples

In the previous section the instructions were provided for the basic transaction set. eSELECTplus also provides several extra features/functionalities for the basic transactions. These features include storing customer and order details, Verified by Visa / MasterCard SecureCode and verifying AVS and CVD details. Verified by Visa / MasterCard SecureCode must be added to your account, please call the Service Centre at 1-866-423-8475 to have your profile updated.

Purchase (with Customer and Order details)

Below is an example of sending a Convenience Fee Purchase with the customer and order details. If one piece of information is sent then all fields must be included in the request. Unwanted fields need to be blank. Please see Appendix C. CustInfo Fields for description of each of the fields. Customer details can only be sent with Purchase, and can be used in conjunction with other extra features such as VBV/MCSC and AVS/CVD. ***Please note that the mpgCustInfo fields are not used for any type of address verification or fraud check.***

```
namespace USMoneris
{
    using System;
        public class TestPurchase
        {
          public static void Main(string[] args)
          {
                /****************** REQUEST VARIABLES*******************************/
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa138";
                string api_token = "qatoken";

                /****************** TRANSACTION VARIABLES ***************************/
               string order_id;          //will prompt user for input
               string amount = "5.00";
               string pan = "4005554444444403";
               string expdate = "0812";
               string crypt_type = "7";
               string commcard_invoice = "INVC090";
               string commcard_tax_amount = "1.00";

                Console.Write ("Please enter an order ID: ");
                order_id = Console.ReadLine();

                USPurchase P =  new USPurchase(order_id,
                                               amount,
                                               pan,
                                               expdate,
                                               crypt_type,
                                               commcard_invoice,
                                               commcard_tax_amount);

               /******************** Billing/Shipping Variables ***************************/
               string first_name = "Bob";
               string last_name = "Smith";
               string company_name = "ProLine Inc.";
               string address = "623 Bears Ave";
               string city = "Chicago";
               string province = "Illinois";
               string postal_code = "M1M2M1";
               string country = "Canada";
               string phone = "777-999-7777";
               string fax = "777-999-7778";
               string tax1 = "10.00";
               string tax2 = "5.78";
               string tax3 = "4.56";
               string shipping_cost = "10.00";

               /******************** Order Line Item Variables ****************************/
               string[] item_description = new string[] { "Chicago Bears Helmet", "Soldier Field Poster" };
               string[] item_quantity = new string[] { "1", "1" };
               string[] item_product_code = new string[] { "CB3450", "SF998S" };
               string[] item_extended_amount = new string[] { "150.00", "19.79" };

               /********************* Customer Information Object ************************/
               CustInfo customer = new CustInfo();

               /********************* Set Customer Billing Information *********************/
```

```
                customer.SetBilling(first_name, last_name, company_name, address, city,
                       province, postal_code, country, phone, fax, tax1, tax2,
                       tax3, shipping_cost);

                /******************* Set Customer Shipping Information **********************/
                customer.SetShipping(first_name, last_name, company_name, address, city,
                       province, postal_code, country, phone, fax, tax1, tax2,
                       tax3, shipping_cost);

                /*************************** Order Line Items  *****************************/
                customer.SetItem(item_description[0], item_quantity[0],
                    item_product_code[0], item_extended_amount[0]);

                customer.SetItem(item_description[1], item_quantity[1],
                    item_product_code[1], item_extended_amount[1]);

                P.SetCustInfo(customer);

                /*************************** Convenience Fee ******************************/
                                ConvFeeInfo convFeeInfo = new ConvFeeInfo();
                convFeeInfo.SetConvenienceFee("1.00");
                P.SetConvFeeInfo(convFeeInfo);

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

               try
               {
                    Receipt receipt = mpgReq.GetReceipt();

                    Console.WriteLine("CardType = " + receipt.GetCardType());
                    Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                    Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                    Console.WriteLine("TransType = " + receipt.GetTransType());
                    Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                    Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                    Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                    Console.WriteLine("Message = " + receipt.GetMessage());
                    Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                    Console.WriteLine("Complete = " + receipt.GetComplete());
                    Console.WriteLine("TransDate = " + receipt.GetTransDate());
                    Console.WriteLine("TransTime = " + receipt.GetTransTime());
                    Console.WriteLine("Ticket = " + receipt.GetTicket());
                    Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                    Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                    Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                    Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                    Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                    Console.WriteLine("FeeType = " + receipt.GetFeeType());
               }
               catch (Exception e)
               {
                        Console.WriteLine(e);
               }
          }

        }
}
```

Purchase (with Verified by Visa / MasterCard SecureCode)

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

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

        public class TestPurchaseVBV
        {
          public static void Main(string[] args)
          {
                /****************** REQUEST VARIABLES*******************************/
                string host = "esplusqa.moneris.com";
                string store_id = "monusqa138";
                string api_token = "qatoken";

                /****************** TRANSACTION VARIABLES ****************************/
                string order_id;          //will prompt user for input
                string cust_id = "B_Urlac_54";
                string amount = "1.00";
                string pan = "4005554444444403";
                string expdate = "0812";
                string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
                string commcard_invoice = "COINV982";
                string commcard_tax_amount = "1.00";

                Console.Write ("Please enter an order ID: ");
                order_id = Console.ReadLine();

               /************************ Transaction Object Option1 ************************/
                USCavvPurchase cavvPurchase = new USCavvPurchase (order_id,
                                                                  cust_id,
                                                                  amount,
                                                                  pan,
                                                                  expdate,
                                                                  cavv,
                                                                  commcard_invoice,
                                                                  commcard_tax_amount);

               /************************ Transaction Object Option2 ************************/
               Hashtable cavvParams = new Hashtable();     //transaction hashtable option
                cavvParams.Add("order_id",order_id);
                cavvParams.Add("cust_id",cust_id);
                cavvParams.Add("amount",amount);
                cavvParams.Add("pan", pan);
                cavvParams.Add("expdate",expdate);
                cavvParams.Add("cavv",cavv);
                cavvParams.Add("commcard_invoice",commcard_invoice);
                cavvParams.Add("commcard_tax_amount",commcard_tax_amount);

               USCavvPurchase cavvPurchase2 = new USCavvPurchase(cavvParams); //single paramater hashtable construtor

               /*************** Address Verification Service **********************/
                AvsInfo avsCheck = new AvsInfo();

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

                cavvPurchase.SetAvsInfo (avsCheck);

                /****************** Card Validation Digits *************************/
                CvdInfo cvdCheck = new CvdInfo();
                cvdCheck.SetCvdIndicator ("1");
                cvdCheck.SetCvdValue ("099");

                cavvPurchase.SetCvdInfo (cvdCheck);

                /****************** Convenience Fee ******************************/
                ConvFeeInfo convFeeInfo = new ConvFeeInfo();
                convFeeInfo.SetConvenienceFee("1.00");
                cavvPurchase.SetConvFeeInfo(convFeeInfo);
```

```
            /************************** Https Post Request *****************************/
            HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, cavvPurchase);

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

                 Console.WriteLine("CardType = " + receipt.GetCardType());
                 Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                 Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                 Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                 Console.WriteLine("TransType = " + receipt.GetTransType());
                 Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                 Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                 Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                 Console.WriteLine("Message = " + receipt.GetMessage());
                 Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                 Console.WriteLine("Complete = " + receipt.GetComplete());
                 Console.WriteLine("TransDate = " + receipt.GetTransDate());
                 Console.WriteLine("TransTime = " + receipt.GetTransTime());
                 Console.WriteLine("Ticket = " + receipt.GetTicket());
                 Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                 Console.WriteLine("Avs Response = " + receipt.GetAvsResultCode());
                 Console.WriteLine("Cvd Response = " + receipt.GetCvdResultCode());
                 Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                 Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                 Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                 Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                 Console.WriteLine("FeeType = " + receipt.GetFeeType());
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }

    }
}
```

Purchase (with CVD and AVS - eFraud)

Below is an example of a Purchase transaction with CVD and AVS information. These values can be sent in conjunction with other additional variables such as customer information. With this feature enabled in your merchant profile, you will be able to pass in these fields for either 'USPurchase' or 'USCavvPurchase'. To form mpgCvdInfo please refer to Appendix H. Card Validation Digits (CVD), to form mpgAvsInfo please refer to Appendix I. Address Verification Service (AVS). To have the eFraud feature added to your profile, please call the Service Centre at 1-866-423-8475 to have your profile updated.

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

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

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;

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

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

                string host = "esplusqa.moneris.com";
                string store_id = "monusqa138";
                string api_token = "qatoken";

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

               string order_id;          //will prompt for user input
               string amount = "5.00";
               string pan = "4005554444444403";
               string expdate = "0812";
               string crypt_type = "7";
               string commcard_invoice = "INV98798";
               string commcard_tax_amount = "1.00";

                Console.Write ("Please enter an order ID: ");
                order_id = Console.ReadLine();

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

                AvsInfo avsCheck = new AvsInfo();

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

                USPurchase purchaseTxn = new USPurchase(order_id, amount, pan, expdate, crypt_type, commcard_invoice,
                                         commcard_tax_amount);

                purchaseTxn.SetAvsInfo (avsCheck);

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

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

                purchaseTxn.SetCvdInfo (cvdCheck);

                /****************** Convenience Fee ********************************/
```

```
                ConvFeeInfo convFeeInfo = new ConvFeeInfo();
                convFeeInfo.SetConvenienceFee("1.00");
                purchaseTxn.SetConvFeeInfo(convFeeInfo);

                /************************* Request ************************/

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

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

                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

                    Console.WriteLine("CardType = " + receipt.GetCardType());
                    Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                    Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                    Console.WriteLine("TransType = " + receipt.GetTransType());
                    Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                    Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                    Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
                    Console.WriteLine("Message = " + receipt.GetMessage());
                    Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                    Console.WriteLine("Complete = " + receipt.GetComplete());
                    Console.WriteLine("TransDate = " + receipt.GetTransDate());
                    Console.WriteLine("TransTime = " + receipt.GetTransTime());
                    Console.WriteLine("Ticket = " + receipt.GetTicket());
                    Console.WriteLine("TimedOut = " + receipt.GetTimedOut ());
                    Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                    Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                    Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                    Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                    Console.WriteLine("FeeType = " + receipt.GetFeeType());
                    Console.WriteLine("Avs Response = " + receipt.GetAvsResultCode());
                    Console.WriteLine("Cvd Response = " + receipt.GetCvdResultCode());

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

        }
}
```

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

9. ACH Transaction Examples

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

ACH Debit

In the ACH Debit example we require several mandatory variables: store_id, api_token, order_id, amount, sec, routing_num, account_num, and account_type. Also, the ConvFeeInfo object must be properly populated and added to the transaction. For examples on how to calculate this amount, please refer to Section 14 - How Do I Calculate The Convenience Fee Amount?

There are also many optional variables, such as the cust_id, check_num and the customer details. Please refer to Appendix A. Definition of Request Fields for all request variables and Appendix D. AchInfo Fields for all ACH variables. ***Please note that the AchInfo fields are not used for any type of address verification or fraud check.***

ACH Debit (Check not present)

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

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

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

                string host = "esplusqa.moneris.com";
                string order_id = "convfeetest1";
                string store_id = "monusqa138";
                string api_token = "qatoken";
                string amount = "1.00";

                //ACHInfo Variables
                string sec = "ppd";
                string cust_first_name = "Christian";
                string cust_last_name = "M";
                string cust_address1 = "3300 Main 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";
                string micr = "";

                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, micr);

                ACHDebit achdebit = new ACHDebit(order_id, amount, achinfo);

                //Convenience Fee
                ConvFeeInfo convFeeInfo = new ConvFeeInfo();
                convFeeInfo.SetConvenienceFee("1.00");
                achdebit.SetConvFeeInfo(convFeeInfo);

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

                //Cust_id Variable
                string cust_id = "customer1";
                achdebit.SetCustId(cust_id);

                ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, achdebit);
```

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

                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

                     Console.WriteLine("CardType = " + receipt.GetCardType());
                     Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                     Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                     Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                     Console.WriteLine("TransType = " + receipt.GetTransType());
                     Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                     Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                     Console.WriteLine("Message = " + receipt.GetMessage());
                     Console.WriteLine("Complete = " + receipt.GetComplete());
                     Console.WriteLine("TransDate = " + receipt.GetTransDate());
                     Console.WriteLine("TransTime = " + receipt.GetTransTime());
                     Console.WriteLine("Ticket = " + receipt.GetTicket());
                     Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                     Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                     Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                     Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                     Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                     Console.WriteLine("FeeType = " + receipt.GetFeeType());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
            }

        } // end TestACHDebit
}
```

As part of the Convenience Fee response there will be five additional reponse methods. Please refer to Appendix B. Definitions of Response Fields for further details.

ACH Credit

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

NOTE	Please note, the ACH Credit transaction may only be submitted with a SEC Code of 'ppd' or 'ccd'. This transaction type allows the merchant to transfer funds from their own bank account directly to the customer's bank account.

```
namespace USMoneris
{
    using System;
        public class TestACHCredit
        {
          public static void Main(string[] args)
          {
                string host = "esplusqa.moneris.com";
                string order_id = "dotnetachcredittest4123421";
                string store_id = "monusqa138";
                string api_token = "qatoken";
                string amount = "1.00";
                //ACHInfo Variables
                string sec = "ppd";
                string cust_first_name = "Christian";
                string cust_last_name = "M";
                string cust_address1 = "3300 Main 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";
                string micr = "";
                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, micr);

                ACHCredit achcredit = new ACHCredit(order_id, amount, achinfo);
                //************************OPTIONAL VARIABLES**************************
                //Cust_id Variable
                string cust_id = "customer1";
                achcredit.SetCustId(cust_id);
                ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, achcredit);
                /*********************   REQUEST  ***********************/
                try
                {         Receipt receipt = mpgReq.GetReceipt();
                          Console.WriteLine("CardType = " + receipt.GetCardType());
                          Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                          Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                          Console.WriteLine("TransType = " + receipt.GetTransType());
                          Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                          Console.WriteLine("Message = " + receipt.GetMessage());
                          Console.WriteLine("Complete = " + receipt.GetComplete());
                          Console.WriteLine("TransDate = " + receipt.GetTransDate());
                          Console.WriteLine("TransTime = " + receipt.GetTransTime());
                          Console.WriteLine("Ticket = " + receipt.GetTicket());
                          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                          Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
          }

        }
}
```

As part of the Convenience Fee response there will be one additional reponse method. Please refer to Appendix B. Definitions of Response Fields for further details.

ACH Reversal

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

Please note, this transaction type will reverse both the transaction amount and the corresponding convenience fee amount so they do not show on the cardholder's statement.

```
namespace USMoneris
{
    using System;

    public class TestACHReversal
    {
        public static void Main(string[] args)
        {
            string host = "esplusqa.moneris.com";
            string order_id = "dotnetachdebitcustinfotest1";
            string store_id = "monusqa138";
            string api_token = "qatoken";
            string txn_number = "132-0_5";

            ACHReversal achreversal = new ACHReversal(order_id, txn_number);

            ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, achreversal);

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

            try
            {
                Receipt receipt = mpgReq.GetReceipt();

                Console.WriteLine("CardType = " + receipt.GetCardType());
                Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                Console.WriteLine("TransType = " + receipt.GetTransType());
                Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Ticket = " + receipt.GetTicket());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
}
```

As part of the Convenience Fee response there will be two additional reponse methods. Please refer to Appendix B. Definitions of Response Fields for further details.

ACH FI Enquiry

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

```
namespace USMoneris
{
    using System;

    public class TestACHFiInquiry
    {
        public static void Main(string[] args)
        {
            string host = "esplusqa.moneris.com";
            string store_id = "monusqa138";
            string api_token = "qatoken";

            string routing_num = "11000015";

            ACHFiInquiry achfiinquiry = new ACHFiInquiry(routing_num);

            ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, achfiinquiry);

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

            try
            {
                Receipt receipt = mpgReq.GetReceipt();

                Console.WriteLine("CardType = " + receipt.GetCardType());
                Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                Console.WriteLine("TransType = " + receipt.GetTransType());
                Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                Console.WriteLine("Message = " + receipt.GetMessage());
                Console.WriteLine("Complete = " + receipt.GetComplete());
                Console.WriteLine("TransDate = " + receipt.GetTransDate());
                Console.WriteLine("TransTime = " + receipt.GetTransTime());
                Console.WriteLine("Ticket = " + receipt.GetTicket());
                Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
            }
            catch (Exception e)
            {
                Console.WriteLine(e);
            }
        }
    }
}
```

10. ACH Transactions with Extra Features – Examples

In the previous section the instructions were provided for the ACH transaction set. eSELECTplus also provides several extra features/functionalities for the ACH transactions. These features include storing customer and order details.

ACH Debit (with Customer and Order details)

Below is an example of sending an ACH Debit with the customer and order details. If one piece of information is sent then all fields must be included in the request. Unwanted fields need to be blank. Please see Appendix C. CustInfo Fields for description of each of the fields. Customer details can only be sent with the ACH Debit transaction. ***Please note that the mpgCustInfo 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 TestACHDebit
        {
          public static void Main(string[] args)
          {

                string host = "esplusqa.moneris.com";
                string order_id = "dotnetachdebitcustinfotest1";
                string store_id = "monusqa138";
                string api_token = "qatoken";
                string amount = "1.00";

                //ACHInfo Variables
                string sec = "ppd";
                string cust_first_name = "Christian";
                string cust_last_name = "M";
                string cust_address1 = "3300 Main 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";
                string micr = "";

                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, micr);

                ACHDebit achdebit = new ACHDebit(order_id, amount, achinfo);

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

                //Cust_id Variable
                string cust_id = "customer1";
                achdebit.SetCustId(cust_id);

                //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);

                achdebit.SetCustInfo(custInfo);

                /****************** Convenience Fee *******************************/

                ConvFeeInfo convFeeInfo = new ConvFeeInfo();
                convFeeInfo.SetConvenienceFee("1.00");
                achdebit.SetConvFeeInfo(convFeeInfo);


                ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, achdebit);

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

                try
                {
                    Receipt receipt = mpgReq.GetReceipt();

                   Console.WriteLine("CardType = " + receipt.GetCardType());
                   Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                   Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                   Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                   Console.WriteLine("TransType = " + receipt.GetTransType());
                   Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                   Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                   Console.WriteLine("Message = " + receipt.GetMessage());
                   Console.WriteLine("Complete = " + receipt.GetComplete());
                   Console.WriteLine("TransDate = " + receipt.GetTransDate());
                   Console.WriteLine("TransTime = " + receipt.GetTransTime());
                   Console.WriteLine("Ticket = " + receipt.GetTicket());
                   Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                   Console.WriteLine("CfSuccess = " + receipt.GetCfSuccess());
                   Console.WriteLine("CfStatus = " + receipt.GetCfStatus());
                   Console.WriteLine("FeeAmount = " + receipt.GetFeeAmount());
                   Console.WriteLine("FeeRate = " + receipt.GetFeeRate());
                   Console.WriteLine("FeeType = " + receipt.GetFeeType());

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

        }
}
```

11. Administrative Transactions

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

Batch Close

At the end of every financial day (11pm EST) the Batch needs to be closed in order to have the Credit Card funds settled the next business day and the ACH funds settled, on average, within the next 5 business days. *By default eSELECTplus will automatically close your Batch daily, whenever there are funds in the open Batch.* Some merchants prefer to control Batch Close, and disable the automatic functionality. For these merchants we have provided the ability to close your Batch through the API. When a Batch is closed the response will include the transaction count and amount for each type of transaction. To disable automatic close please access the Merchant Resource Centre (https://esplus.moneris.com/usmpg), go to the Admin menu item and then choose Store Settings; the Batch Close options are located on this page.

Please note, the Batch Close transaction will ready the transaction amount for settlement, but it will not affect the convenience fee funds. It is recommended that the default daily auto close features be used to synchronize the settlement of the funds.

```
namespace USMoneris
{
    using System;

        public class TestBatchClose
        {
          public static void Main(string[] args)
          {
            string host = "esplusqa.moneris.com";
            string store_id = "monusqa138";
            string api_token = "qatoken";

            string ecr_no = "64000074";

            HttpsPostRequest mpgReq = new HttpsPostRequest(host,store_id, api_token, new USBatchClose (ecr_no));

            try
            {
               Receipt receipt = mpgReq.GetReceipt();
               if ( (receipt.GetReceiptId()).Equals("Global Error Receipt") )
               {
                     Console.WriteLine("CardType = " + receipt.GetCardType());
                     Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                     Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                     Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                     Console.WriteLine("TransType = " + receipt.GetTransType());
                     Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                     Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                     Console.WriteLine("BankTotals = null");
                     Console.WriteLine("Message = " + receipt.GetMessage());
                     Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                     Console.WriteLine("Complete = " + receipt.GetComplete());
                     Console.WriteLine("TransDate = " + receipt.GetTransDate());
                     Console.WriteLine("TransTime = " + receipt.GetTransTime());
                     Console.WriteLine("Ticket = " + receipt.GetTicket());
                     Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                }
                else
                {
                   foreach ( string ecr in receipt.GetTerminalIDs() )
                   {
                     Console.WriteLine("ECR: " + ecr);
                     foreach ( string CardType in receipt.GetCreditCards(ecr) )
                     {
                         Console.WriteLine("\tpan Type: " + CardType);

                         Console.WriteLine("\t\tPurchase: Count = "
                                                 + receipt.GetPurchaseCount(ecr,CardType)
                                                 + " Amount = "
                                                 + receipt.GetPurchaseAmount(ecr,CardType));

                          Console.WriteLine("\t\tRefund: Count = "
```

```
                                                    + receipt.GetRefundCount(ecr,CardType)
                                                    + " Amount = "
                                                    + receipt.GetRefundAmount(ecr,CardType));

                        Console.WriteLine("\t\tCorrection: Count = "
                                                    + receipt.GetCorrectionCount(ecr,CardType)
                                                    + " Amount = "
                                                    + receipt.GetCorrectionAmount(ecr,CardType));
                    }
                  }
                }
              }
              catch (Exception e)
              {
                 Console.WriteLine(e);
              }
            }
          }
}
```

Open Totals

Open Totals allows the merchant to retrieve details about all Credit Card transactions within the currently open Batch. The response will include the transaction count and amount for each type of transaction. Open Totals returns a similar response to the Batch Close without closing the current Batch.

Please note, the Open Totals transaction will only provide the details about the transaction amount funds, it will not provide convenience fee details.

```
namespace USMoneris
{
    using System;
        public class TestOpenTotals
        {
          public static void Main(string[] args)
          {
                 string host = "esplusqa.moneris.com";
                 string store_id = "monusqa138";
                 string api_token = "qatoken";
                 string ecr_no = "64000074";

                HttpsPostRequest mpgReq = new HttpsPostRequest(host, store_id, api_token, new USOpenTotals (ecr_no));
                try
                {
                    Receipt receipt = mpgReq.GetReceipt();
                    if ( (receipt.GetReceiptId()).Equals("Global Error Receipt") )
                    {
                             Console.WriteLine("CardType = " + receipt.GetCardType());
                             Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
                             Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
                             Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
                             Console.WriteLine("TransType = " + receipt.GetTransType());
                             Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
                             Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
                             Console.WriteLine("BankTotals = null");
                             Console.WriteLine("Message = " + receipt.GetMessage());
                             Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
                             Console.WriteLine("Complete = " + receipt.GetComplete());
                             Console.WriteLine("TransDate = " + receipt.GetTransDate());
                             Console.WriteLine("TransTime = " + receipt.GetTransTime());
                             Console.WriteLine("Ticket = " + receipt.GetTicket());
                             Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
                    }
                    else
                    {
                            foreach ( string ecr in receipt.GetTerminalIDs() )
                            {
                                Console.WriteLine("ECR: " + ecr);
                                foreach ( string CardType in receipt.GetCreditCards(ecr) )
                                {
                                    Console.WriteLine("\tpan Type: " + CardType);

                                    Console.WriteLine("\t\tPurchase: Count = "
                                                      + receipt.GetPurchaseCount(ecr,CardType)
                                                      + " Amount = "
                                                      + receipt.GetPurchaseAmount(ecr,
                                                                                  CardType));
                                    Console.WriteLine("\t\tRefund: Count = "
                                                      + receipt.GetRefundCount(ecr,CardType)
                                                      + " Amount = "
                                                      + receipt.GetRefundAmount(ecr,CardType));

                                    Console.WriteLine("\t\tCorrection: Count = "
                                                      + receipt.GetCorrectionCount(ecr,CardType)
                                                      + " Amount = "
                                                      + receipt.GetCorrectionAmount(ecr,CardType));
                                }
                            }
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine(e);
                }
                 }

        }
}
```

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

For each transaction you will receive a response message. For a full description of each field please refer to Appendix B. Definitions of Response Fields.

To determine whether a transaction is successful or not the field that must be checked is ResponseCode. See the table below to determine the transaction result.

Response Code	Result
0 – 49 (inclusive)	Approved
50 – 999 (inclusive)	Declined
null	Incomplete

For a full list of response codes and the associated message please refer to the Response Code document available at https://esplusqa.moneris.com/connect/en/download/index.php

13. 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
monusqa138	qatoken	demouser	abc1234

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

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

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

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

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

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


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.

14. How Do I Calculate The Convenience Fee Amount?

A merchant account may be setup for one of the following 2 convenience fee programs:

1. Fixed Fee Amount
2. Percentage Fee

To properly calculate the convenience fee, we recommend that the merchant be aware of their Convenience Fee profile setup. In case convenience fee setup changes were to occur on the merchant's profile, the convenience fee setup will also be returned in the transaction response. Please note, the convenience fee must be less than the total amount for goods or services.

For the convenience fee setup, please refer to the following response variables:

Variable Name	Size/Type	Description
FeeAmount	9 / decimal	The expected Convenience Fee amount. This field will return the amount submitted by the merchant for a successful transaction. For an unsuccessful transaction, it will return the expected convenience fee amount.
FeeRate	9 / decimal	The convenience fee rate that has been defined on the merchant's profile. For example: 1.00 – a fixed amount or 10.0 - a percentage amount
FeeType	AMT / PCT	This field defines the type of convenience fee the merchant account has been setup for. Available options are: AMT – fixed amount PCT – percentage

Convenience Fee Examples

Description	Convenience Fee Examples

In the example to the right the merchant account is setup for a Fixed Convenience Fee with a per transaction rate of 5.75.
FeeType = AMT
FeeRate = 5.75
 	
For this merchant setup, each Purchase transaction that is submitted will require a convenience fee amount of $5.75 to be submitted as well.
	`string order_id = "testing";`
	`string amount = "100.00";`
	`string pan = "4242424242424242";`
	`string expdate = "1206";`
	`string crypt_type = "7";`
	`string commcard_invoice = "INVC090";`
	`string commcard_tax_amount = "1.00";`
	
	`USPurchase usPurchase = new USPurchase(order_id,amount,`
	`pan,expdate,crypt_type,commcard_invoice,commcard_tax_amount);`
	
	`ConvFeeInfo convFeeInfo = new ConvFeeInfo();`
	`convFeeInfo.SetConvenienceFee("5.75");`
	`usPurchase.SetConvFeeInfo(convFeeInfo);`

In the example to the right the merchant account is setup for a Percentage Convenience Fee. This merchant's rate is set at 10% of the total transaction amount.
FeeType = PCT
FeeRate = 10.0
 	
For this merchant setup, each Purchase transaction that is submitted will require the merchant to calculate 10% of the 'amount' field. For example, for an 'amount' of 20.00 the 'convenience_fee' will be submitted as 2.00.
	`string order_id = "testing";`
	`string amount = "20.00";`
	`string pan = "4242424242424242";`
	`string expdate = "1206";`
	`string crypt_type = "7";`
	`string commcard_invoice = "INVC090";`
	`string commcard_tax_amount = "1.00";`
	
	`USPurchase usPurchase = new USPurchase(order_id,amount,`
	`pan,expdate,crypt_type,commcard_invoice,commcard_tax_amount);`
	
	`ConvFeeInfo convFeeInfo = new ConvFeeInfo();`
	`convFeeInfo.SetConvenienceFee("2.00");`
	`usPurchase.SetConvFeeInfo(convFeeInfo);`

15. Convenience Fee Regulations

When processing a convenience fee transaction, Visa and MasterCard expect certain details be displayed to the cardholder prior to completing the transaction.

The following details must be disclosed/provided to the cardholder prior to checkout:

1. Transaction Amount – amount of goods or services
2. Convenience Fee Amount – amount to be charged for offering a convenient payment method
3. Option to cancel the transaction
4. Details for other available payment options (i.e. in store or call in details)

16. What Do I Need to Include in the Receipt?

Visa and MasterCard expect that certain variables be returned to the cardholder and presented as a receipt when a transaction is approved. These fields vary depending on what type of transaction was performed:

- Credit Card Transaction – please refer to Appendix K. Basic Transaction Receipt

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

- ACH Transaction (Check not present) – please refer to Appendix L. ACH Transaction Receipt

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

17. How Do I Activate My Store?

Once you have received your activation letter/fax go to https://esplus.moneris.com/usmpg/activate/ as instructed in the letter/fax. You will need to input your store ID and merchant ID then click on 'Activate'. In this process you will need to create an administrator account that you will use to log into the Merchant Resource Centre to access and administer your eSELECTplus store. You will need to use the Store ID and API Token to send transactions through the API.

Once you have created your first Merchant Resource Centre user, please log on to the Interface by clicking the "eSELECTplus" button. Once you have logged in please proceed to ADMIN and then STORE SETTINGS. At the top of the page you will locate your production API Token.

18. How Do I Configure My Store For Production?

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

Once you are in production you will access the Merchant Resource Centre at https://esplus.moneris.com/usmpg . You can use the store administrator ID you created during the activation process and then create additional users as needed.

For further information on how to use the Merchant Resource Centre please see the HELP button found in the top left corner of the website.

19. How Do I Get Help?

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

For technical support:
Phone: 1-866-696-0488

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

When sending an email support request please be sure to include your name and phone number, a clear description of the problem, as well as the type of API that you are using (i.e. .NET Convenience Fee API). **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.**

20. Appendix A. Definition of Request Fields

Request Fields		
Variable Name	**Size/Type**	**Description**
order_id	50 / an	Merchant defined unique transaction identifier - must be unique for every Purchase, Independent Refund, ACH Debit, and ACH Credit attempt. For Refunds, Voids and ACH Reversals the order_id must reference the original transaction. Characters allowed for Order ID: **a-z A-Z 0-9 _ - : . @ spaces**
pan	20 / variable	Credit Card Number - no spaces or dashes. Most credit card numbers today are 16 digits in length but some 13 digits are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and/or potential support of private label card ranges.
expiry_date	4 / num	Expiry Date - format **YYMM** no spaces or slashes. PLEASE NOTE THAT THIS IS REVERSED FROM THE DATE DISPLAYED ON THE PHYSICAL CARD WHICH IS MMYY
amount	9 / decimal	Amount of the transaction – cost of goods or services. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
convenience_fee	9 / decimal	Amount of the convenience fee. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99. Please note, the 'convenience_fee' must be lower than the 'amount'.
crypt	1 / an	E-Commerce Indicator: 1 - Mail Order / Telephone Order - Single 4 - Mail Order / Telephone Order - Unknown Classification 5 - Authenticated E-commerce Transaction (VBV/MCSC) 6 – Non Authenticated E-commerce Transaction (VBV/MCSC) 7 - SSL enabled merchant 8 - Non Secure Transaction (Web or Email Based) 9 - SET non - Authenticated transaction
txn_number	255 / varchar	Used when performing follow on transactions - this must be filled with the value that was returned as the TxnNumber in the response of the original transaction. When performing a Refund or a Void this must reference the Purchase.
cust_id	50/an	This is an optional field that can be sent as part of a Purchase or ACH Debit request. It is searchable from the Moneris Merchant Resource Centre. It is commonly used for policy number, membership number, student ID or invoice number.
cavv		This is a value that is provided by the Moneris MPI or by a third party MPI. It is part of a VBV/MCSC transaction.
commcard_invoice	17 / an	Level 2 Invoice Number for the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). Characters allowed for commcard_invoice: **a-z A-Z 0-9 spaces**
commcard_tax_amount	9 / decimal	Level 2 Tax Amount of the transaction. Used for Corporate Credit Card transactions (Commercial Purchasing Cards). This must contain 3 digits with two penny values. The minimum value passed can be 0.00 and the maximum is 9999999.99

avs_street_number avs_street_name	19 / an	Street Number & Street Name (max – 19 digit limit for street number and street name combined). This must match the address that the issuing bank has on file.
avs_zipcode	9 / an	Zip or Postal Code – This must match what the issuing banks has on file.
cvd_indicator	1 / num	CVD presence indicator (1 digit – refer to Appendix H. Card Validation Digits (CVD) for values)
cvd_value	4 / num	Credit Card CVD value – this number accommodates either 3 or 4 digit CVD values. Refer to Appendix H. Card Validation Digits (CVD) for further details. Note: The CVD value supplied by the cardholder should simply be passed to the eSELECTplus payment gateway. Under no circumstances should it be stored for subsequent uses or displayed as part of the receipt information.
dynamic_descriptor	25 / an	Merchant defined description sent on a per-transaction basis that will appear on the credit card statement appended to the merchant's business name. Please note, the combined length of the merchant's business name and dynamic_descriptor may not exceed 25 characters.

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 _ - : . @ $ =** /

21. Appendix B. Definitions of Response Fields

<table>
<tr><th colspan="3">Response Fields</th></tr>
<tr><th>Variable Name</th><th>Size/Type</th><th>Description</th></tr>
<tr><td>ReceiptId</td><td>50 / an</td><td>order_id specified in request</td></tr>
<tr><td>ReferenceNum</td><td>18 / num</td><td>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.</td></tr>
<tr><td>ReponseCode</td><td>3 / num</td><td>Transaction Response Code
< 50: Transaction approved
>= 50: Transaction declined
NULL: Transaction was not sent for authorization

* If you would like further details on the response codes that are returned please see the Response Codes document available at https://esplusqa.moneris.com/connect/en/download/index.php

Other possible Convenience Fee response codes:
<table><tr><th>Response Code</th><th>Definition</th></tr><tr><td>973</td><td>Unable to locate merchant CF details</td></tr><tr><td>977</td><td>Invalid amount</td></tr><tr><td>978</td><td>Failed CF transaction</td></tr><tr><td>984</td><td>Data error: (optional: field name)</td></tr><tr><td>987</td><td>Invalid transaction</td></tr><tr><td>null</td><td>Error: Malformed XML</td></tr></table></td></tr>
<tr><td>AuthCode</td><td>8 / an</td><td>Authorization code returned from the issuing institution</td></tr>
<tr><td>TransTime</td><td>##:##:##</td><td>Processing host time stamp</td></tr>
<tr><td>TransDate</td><td>yyyy-mm-dd</td><td>Processing host date stamp</td></tr>
<tr><td>TransType</td><td>an</td><td>Type of transaction that was performed</td></tr>
<tr><td>Complete</td><td>true/false</td><td>Transaction was sent to authorization host and a response was received</td></tr>
<tr><td>Message</td><td>100 / an</td><td>Response description returned from issuing institution.</td></tr>
<tr><td>TransAmount</td><td>9 / decimal</td><td>‘amount’ specified in request – amount for goods or services sold</td></tr>
<tr><td>CardType</td><td>2 / alpha</td><td>Credit Card Type</td></tr>
<tr><td>TxnNumber</td><td>20 / an</td><td>Gateway Transaction identifier</td></tr>
<tr><td>TimedOut</td><td>true/false</td><td>Transaction failed due to a process timing out</td></tr>
<tr><td>Ticket</td><td>n/a</td><td>reserved</td></tr>
<tr><td>AvsResultCode</td><td>1/alpha</td><td>Indicates the address verification result. Refer to Appendix I. Address Verification Service (AVS).</td></tr>
<tr><td>CvdResultCode</td><td>2/an</td><td>Indicates the CVD validation result. Refer to Appendix H. Card Validation Digits (CVD).</td></tr>
</table>

FeeAmount	9 / decimal	The expected Convenience Fee amount. This field will return the amount submitted by the merchant for a successful transaction. For an unsuccessful transaction, it will return the expected convenience fee amount.
FeeRate	9 / decimal	The convenience fee rate that has been defined on the merchant's profile. For example: 1.00 – a fixed amount or 10.0 - a percentage amount
FeeType	AMT / PCT	The type of convenience fee that has been defined on the merchant's profile. Available options are: AMT – fixed amount PCT – percentage
CfSuccess	true/false	Indicates whether the Convenience Fee transaction processed successfully.
CfStatus	2/an	Indicates the status of the merchant and convenience fee transactions. The CfStatus field provides details about the transaction behavior and should be referenced when contacting Moneris Customer Support. Possible values are: 1 or 1F – Completed 1st purchase transaction 2 or 2F – Completed 2nd purchase transaction 3 – Completed void transaction 4A or 4D – Completed refund transaction 7 or 7F – Completed merchant independent refund transaction 8 or 8F – Completed merchant refund transaction 9 or 9F – Completed 1st void transaction 10 or 10F – Completed 2nd void transaction 11A or 11D – Completed refund transaction

22. 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. For an example, please refer to Section 8 - Purchase (with Customer and Order details).		
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. For an example, please refer to Section 8 - Purchase (with Customer and Order details).		
item_description	30 / an	
item_quantity	10 / num	You must send a quanitity > 0 or the item will not be added to the item list (ie. minimum 1, maximum 9999999999)
item_product_code	30 / an	
item_extended_amount	9 /decimal	This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99
Extra Details		
email	50 / an	
instructions	50 / an	

NOTE

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

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

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

23. Appendix D. 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 Please refer to Appendix E. ACH Sec Codes for full sec code description
routing_num	9 / num	The first number in the MICR, or magnetic ink character recognition, line at the bottom of a check is the bank's check routing number. It is exactly nine digits long and always starts with 0, 1, 2 or 3.
account_num	50 / num	The account number may appear before or after the check number in the check's MICR line at the bottom of the check. The length of the account number varies with a maximum length of 50 digits.
check_num	16 / num	The sequential number for checks appears in both the MICR line at the bottom of the check and the upper right corner of the check. The check number length may vary; the maximum length is 16 digits. This is an optional field.
account_type	savings / checking	Identifies the type of bank account. The account type must be submitted as either ‘savings’ or ‘checking’. This field is case sensitive.
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.

24. Appendix E. ACH Sec Codes

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

25. 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

26. Appendix G. Convenience Fee Response Codes

The following is a list of possible convenience fee responses.

Convenience Fee Response Codes	
RESULT VALUE	**DEFINITION**
973	Unable to locate merchant CF details
977	Invalid amount
978	Failed CF transaction
984	Data error: (optional: field name)
987	Invalid transaction
null	Error: Malformed XML

27. Appendix H. Card Validation Digits (CVD)

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

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

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

CVD Response codes:

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

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


NOTE

For American Express, a CVD Response code will not be returned; the response will be either Approved or Declined.

To have CVD for American Express added to your profile, please contact AmEx directly.

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

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

28. Appendix I. Address Verification Service (AVS)

The Address Verification Service (AVS) value refers to the cardholder's street number, street name and zip/postal code as it would appear on their statement. mpgAvsInfo is broken down into three elements:

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

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

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

29. Appendix J. Additional Information for CVD and AVS

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

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

NOTE

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

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

---

30. Appendix K. Basic Transaction Receipt

For credit card transactions, the credit card associations expect certain fields to be presented to the cardholder on a receipt.

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

REAL CONVENIENCE

150 Ocean Drive Suite 120 Chicago ID 44444
T: 1231231234 F: 1231234123 *www.monerisusa.com*

TRANSACTION APPROVED - THANK YOU

Payment Details

Transaction Type: SALE
Transaction Amount: $10.00 (USD)
Convenience Fee: $1.00
Total Amount: $11.00 (USD)
Order ID: mvt9019127877
Card Num: **** **** **** 4242
Card Type: VISA
Resp Code - Message: 001 - APPROVED 104182
Auth Code: 104182
Reference Num: 640000740011415300 M
Date/Time: Nov 19 2010 01:28PM
CVD Result: CVD was not performed. (Code: n/a)
AVS Result: AVS check was not performed. (Code: n/a)
Invoice #/Customer Code: 1234
Tax Amount: $0.10

Refund Policy: Within 30 days

SIGNATURE _____

Cardholder will pay card issuer above amount
pursuant to Cardholder Agreement

Item Details

Description	Product Code	Quantity	Price
Shoes - Red Slippers	AS123	1	$10.00
		Shipping:	$1.00
		Tax 1:	$0.50
		Tax 2:	$0.50
		Total (USD):	$10.00

Customer Details

Customer ID: My personal Customer ID
Email Address: bob@smith.com
Note: Please deliver during the day

Address Details

Billing
Bob Smith
Moneris
101 Main St
Springfield
NY

123456
USA
Phone: 555-555-5555
Fax: 555-555-5566

Shipping
Mary Smith
My Company
111 Lakeshore Blvd
Chicago
Illinois

234567
USA
Phone: 555-111-2222
Fax: 555-222-4444

31. Appendix L. ACH Transaction Receipt

For an ACH transaction, a transaction confirmation is not mandatory; though eSELECTplus does recommend that a receipt of registration of the transaction be provided to the customer. Below is a list of recommended fields and the format they are to be displayed in.

	Field	Description
1	Merchant Name	The name of the store / business.
2	Merchant URL	Web site address of the store / business.
3	Transaction Type	The type of transaction that was performed: - Check Sale (ACH Debit) - Check Refund (ACH Credit) - Check Reversal (ACH Reversal) *NOTE: The terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction type are indicated in brackets.*
4	Payment Type	Indicate that this is an ACH processed transaction.
5	SEC Code	Specify which SEC Code was submitted. Identifies how the bank account information was collected. i.e. WEB – Internet Initiated Entry
6	Transaction Amount	The total amount being debited or credited to the bank account for the goods or services.
7	**Convenience Fee Amount**	**The total amount being debited or credited to the bank account for the convenience of an alternate payment method.**
8	Transaction Date and Time	The date may be in any format, but must include the day, month and year. The time must be in 24 hour format. It is good practice to include the seconds in the time format to help with tracing transactions.
9	Reference Number	6400135 = terminal number 001 = shift 001 = batch number 001 = sequence number 0 = reserved
10	Auth Code	The authorization number is only printed if the transaction is approved. If the transaction is declined the field may be omitted.
11	Response Code	The 3 digit response code returned in the transaction (ex. 000 – 999)
12	Response Message / Result	Message indicating whether the transaction was Registered or Failed to Register.
13	Account Number	Customer's bank account number. All but the last 4 digits of the account number must be masked out.
14	Routing Number	Check routing number to identify the Financial Institution.
15	Check Number	Used for check tracking purposes.
16	Account Type	Indicate whether this is a Savings or Checking account.

REAL CONVENIENCE

150 Ocean Drive Suite 120 Chicago ID 44444
T: 1231231234 F: 1231234123 *www.monerisusa.com*

TRANSACTION REGISTERED - THANK YOU

Payment Details

Transaction Type: CHECK SALE
Payment Type: CHECK
SEC Code: WEB - Internet Initiated Entry
Transaction Amount: $10.00 (USD)
Convenience Fee: $1.00
Total Amount: $11.00 (USD)
Order ID: mch9019294755
Account Num: ***4321
Routing Num: 123456789
Check Num: 101
Account Type: Checking
Resp Code - Message: 027 - REGISTERED * =
Auth Code:
Reference Num: 002000960010200010 M
Date/Time: Nov 19 2010 01:55PM

Refund Policy: Within 30 days

ACH Customer Information

Customer Name: John Smith
Street Address 1: 101 Main St
Street Address 2: Apt 1510
City: New York
State: NY
Zip Code: 987654

Item Details

Description	Product Code	Quantity	Price
Shoes - Red Slippers	AS123	1	$1.00
Shoes - Blue Suede	BC456	1	$2.00
Shoes - Yellow Tap	CD567	2	$1.00
Shoes - Gold Salsa	DE678	3	$1.00
		Shipping:	$1.00
		Tax 1:	$0.50
		Tax 2:	$0.50
		Total (USD):	$10.00

Customer Details

Customer ID: My personal Customer ID
Email Address: bob@smith.com
Note: Please deliver during the day

Address Details

Billing

Bob Smith
Moneris
500 Royal Rd
Cityville
NC
876543
USA
Phone: 555-555-5555
Fax: 555-555-5566

Shipping

Mary Smith
Solutions
600 Hello Way
Citycentre
SC
765432
USA
Phone: 555-111-1111
Fax: 555-222-2222