Merchant Integration Guide v. 1.1.0

ERNEX Loyalty .NET API (1.0 and newer)

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

This document describes the basic information for using the .NET API for processing loyalty transactions. In particular, it describes the format for sending transactions and the corresponding responses you will receive. If you are interested in also being able to accept credit card payments via your online application, please refer to the .NET API document found at: https://www.eselectplus.ca/en/downloadable-content for eSelectPlus Canada and https://esplusqa.moneris.com/connect/en/download/index.php for eSelectPlus US.

It is also recommended that your review the Moneris Loyalty Integration Guide for an overview of the Moneris Loyalty transactional services available today and describe how to integrate processing entity solutions with Moneris System and Skill Requirements

2. System and Skill Requirements

In order to use .NET your system will need to have the following:

1. A web server with an SSL certificate
2. .NET Framework
3. Port 443 open for bi-directional communication

As well, you will need to have the following knowledge and/or skill set:

1. Install a dll into the global assembly cache
2. knowledge of the .NET Framework

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

You will need to follow these steps.

1. Identify the solution you want to develop. Fill out our Moneris Profile listing all the requirements for the merchant service.
2. Do the required development as outlined in this document
3. Test your solution in the test environment
4. Schedule a certification time window. Complete and pass the certification.
5. Activate your store
6. Make the necessary changes to move your solution from the test environment into production as outlined in this document

4. Transaction Types and Transaction Flow

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

Financial Transactions

A financial transaction is defined as one which modifies the balance of a loyalty card. The following financial transactions are supported.

Purchase – The Purchase transaction is used when the customer presents their loyalty card at checkout time in conjunction with payment for goods or services. For a loyalty transaction, another payment means must be used. The Purchase transaction is used to reward customer loyalty by awarding points to an active loyalty card based on the dollar value of purchases a card holder makes.

Void – (Correction) A Void transaction is used to cancel a previously completed loyalty Purchase, Activation, Capture, Redemption or Refund transaction. A Void will undo the effect of the original transaction, giving the cardholder the appearance that the original transaction was never completed.

Refund - Similar to the Void, the Refund will remove the points awarded during a previously performed Loyalty Purchase or Capture transaction. Though unlike the Void, the Refund will not cancel the previous Purchase/Capture. Refunds are primarily used to credit the benefit amount of a previously processed transaction that is in a closed batch. With respect to batch totals Refund transactions are added to the batch Refund Count and Refund Total statistics. The only transactions that can be Refunded are a Purchase and Capture. Some card programs (i.e. CardCode) may not allow Refunds

Independent Refund - The Independent Refund will credit a specified amount to the loyalty card. The Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the card number and expiry date will need to be passed. With respect to batch totals Independent Refund transactions are added to the batch Refund Count and Refund Total statistics.

Pre-Auth (Authorization) - A Pre-authorization transaction is used to collect the loyalty card number and determine the potential rewards to be awarded for a purchase that will be completed at a later date for less than, equal to or above the pre-authorized amount.

Completion - The completion transaction is used to complete a previously authorized pre-authorization transaction and actually adjusts the cards balance. The completion amount may be less than, equal to, or greater than the amount that was already pre-authorized.

Redemption - The redemption transaction allows the cardholder to receive value/purchase goods and services with their accumulated rewards.

* A Void can be performed against a transaction as long as the batch that contains the original transaction remains open. When using the automated closing feature by default Batch Close occurs daily between 10 – 11 pm EST; unless an alternative time has been arranged.

Administrative Transactions

Administrative transactions provide maintenance functions. The following administrative transactions are supported.

Initialization – Before a Loyalty transaction may be processed via eSELECTplus, an Initialization transaction must be completed. This transaction provides the merchant with the card configuration information required to identify cards that are eligible for loyalty card processing and to determine how to complete the various transactions.

Activation – A new loyalty card must be activated prior to being used. The Activation transaction of a loyalty card activates the card and stores points on the card record on the host, which can be used in exchange for goods and services. Upon successful completion of the transaction, the card balance is incremented for the processed amount.

Deactivation – (Disable) The Deactivation transaction allows the merchant to disable a loyalty card. This transaction is primarily used for loyalty cards which have become demagnetized or otherwise damaged. Upon successful completion of this transaction, the account can no longer be used.

Card Inquiry – The Card Inquiry transaction allows a merchant to retrieve details about a specific cardholder account.

Card Data Inquiry – The Card Data Inquiry transaction provides the merchant with the card configuration information required to identify cards that are eligible for loyalty card processing and to determine how to complete the various transactions.

Group – The Group transaction allows the merchant to link multiple transactions. For example, link a loyalty card transaction with a credit card transaction. This transaction is primarily used for reporting purposes.

Batch Close ** – (End of Day) When a Batch Close is performed it takes the totals from all transactions so they will be returned in the response for end of day reconciliation. This transaction also increments the batch number and resets the totals.

** If merchant has the Electronic Funds Transfer (EFT) function enabled it is necessary to close the batch.

Open Totals – (Current Batch Report) When an Open Totals transaction is performed it returns the details about all transactions within the currently open batch. This transaction is similar to the Batch Close, though it does not close the Batch and increment the batch number.

Transaction Flow

A number of the transaction request fields are mandatory or optional, depending on the card program(s) configured for the merchant. The specifics of the card program(s) supported are needed prior to processing any transactions. To begin, process a Card Data Inquiry transaction to retrieve all necessary card configuration information. This transaction should also be processed after any changes are made to the card program. Please refer to section Administrative Transaction Examples (Card Data Inquiry) for further details on this transaction type.

5. Financial Transaction Examples

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

Purchase (Award Loyalty Points)

A Purchase transaction is used to reward customer loyalty by awarding points to an active loyalty card based on the dollar value of purchases a card holder makes. It should be performed after a credit or debit purchase is complete. An option exists for Merchants to award points for an amount different than the total transaction amount. This Ernex feature is known as Benefit Amount prompt.

In the Purchase example we require several variables (store_id, api_token, order_id, total_amount, track2 and/or pan, expdate, language_code, benefit_amount and payment_type). Please refer to section 19 for further details on how to populate the 'language_code' variable. The merchant may both swipe the card using a Mag Swipe Reader (MSR) and pass the data in the 'track2' variable. Or, the card information may be manually keyed-in and passed using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction. To determine the rules regarding populating the value of the 'benefit_amount" in the request, please refer to the 'BenefitAmountPrompt' field in the Card Data Inquiry response. In the purchase transaction, the request benefit_amount should be provided in dollars and cents. In the response, the benefit will be in points.

There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. Please refer to section 14 for all variable definitions.

```
namespace Moneris
{
  using System;
    public class TestLoyaltyPurchase
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        //string cust_id = "customer1";  //optional
        string total_amount = args[3];
        string track2 = "";

        //Console.WriteLine("Please swipe card");
        //track1 = Console.ReadLine();
        //track2 = Console.ReadLine();

        string pan = args[4];
        string expdate = args[5];
        string cvd_value = args[6];
        string language_code = args[7];
        string info = args[8];
        string benefit_amount = args[9];
        string payment_type = args[10];

        ErnexLoyaltyPurchase ernexLoyaltyPurchase = new ErnexLoyaltyPurchase(order_id, total_amount,
                track2, pan, expdate, language_code, benefit_amount, payment_type);

        ernexLoyaltyPurchase.SetCvdValue(cvd_value); //optional
        ernexLoyaltyPurchase.SetInfo(info); //optional

        ErnexHttpsPostRequest mpgReq = new ErnexHttpsPostRequest(host, store_id, api_token,
                ernexLoyaltyPurchase);
        try
        {
            ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
```

```
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
          Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
              Console.WriteLine("recordType = " + recordType);
              Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
              Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                        recordType));
              Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                        recordType));
              Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }
        }
        }
        catch (Exception e)
        {
           Console.WriteLine(e);
        }
      }
    } // end TestLoyaltyPurchase
}
```

Void

The Void transaction is used to cancel a transaction that was performed in the current Batch. The account will be reversed to the state prior to the incorrectly performed transaction. Void transactions are treated as a correction with respect to batch totals; Void transactions are added to the batch Correction Count Statistics. When processing a Void, no amount is required because a Void is always for 100% of the original transaction. The only transactions that can be Voided are Purchase, Activation, Refund, Capture and Redemption.

To send a Void the order_id and txn_number from the Purchase, Activation or Refund are required. There is also 1 extra variable (cvd_value) that may be either mandatory or optional, depending on the card program (i.e. CardCode). To determine whether the 'cvd_value' is mandatory, please refer to the 'CVCprompt' field which was returned in the Card Data Inquiry response – please see section 15 for further details. Please refer to section 14 for all variable definitions.

```
namespace Moneris
{
  using System;
    public class TestLoyaltyVoid
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        string cvd_value = args[3];
        string txn_number = args[4];

        ErnexLoyaltyVoid ernexLoyaltyVoid = new ErnexLoyaltyVoid(order_id, txn_number);

        ernexLoyaltyVoid.SetCvdValue(cvd_value); //optional

        ErnexHttpsPostRequest mpgReq =
            new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyVoid);

        try
        {
          ErnexReceipt receipt = mpgReq.GetReceipt();

          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
          Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
          Console.WriteLine("TransTime = " + receipt.GetTransTime());
          Console.WriteLine("TransDate = " + receipt.GetTransDate());
          Console.WriteLine("TransType = " + receipt.GetTransType());
          Console.WriteLine("Message = " + receipt.GetMessage());
          Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
          Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
          Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
          Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
          Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
          Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
          Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
          Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
          Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
          Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
          Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
          Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
          Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
          Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
          Console.WriteLine("Benefit = " + receipt.GetBenefit());
          Console.WriteLine("Language = " + receipt.GetLanguage());
          Console.WriteLine("Error Code = " + receipt.GetErrorCode());
          Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
          Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
          Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
          Console.WriteLine("CardStatus = " + receipt.GetCardStatus());
```

```
          foreach (string cardCode in receipt.GetCardCodes())
          {
            Console.WriteLine("cardCode = " + cardCode);
            Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
            Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
            Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
            Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
            Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
            Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
            Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
            Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
            Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
            Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
            Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
            Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
            Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
            Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
            Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
            Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
            Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
            Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
            Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

            foreach (string recordType in receipt.GetRecordType(cardCode))
            {
                Console.WriteLine("recordType = " + recordType);
                Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
                Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                  recordType));
                Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                  recordType));
                Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode,
                                  recordType));
                Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
                Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
              }

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

    } // end TestLoyaltyVoid
}
```

Refund

Similar to the Void, the Refund will remove the points awarded during a previously performed Loyalty Purchase or Capture transaction. Though unlike the Void, the Refund will not cancel the previous Purchase/Capture. Refunds are primarily used to credit the benefit amount of a previously processed transaction that is in a closed batch. With respect to batch totals Refund transactions are added to the batch Refund Count and Refund Total statistics. The only transactions that can be Refunded are a Purchase and Capture. Some card programs (i.e. CardCode) may not allow Refunds; the merchant must first check the 'RefundAllowed' parameter of the card program in the Card Data Inquiry transaction. Please refer to section 15 for further details.

To send a Refund the order_id and txn_number from the Purchase or Capture are required. The benefit_amount and total_amount are also required. The total_amount is the financial amount that was refunded and the benefit_amount is the portion of the total_amount to which loyalty is applicable.

There are two extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. Please refer to section 14 for all variable definitions. . To determine the rules regarding populating the value of the 'benefit_amount" in the request, please refer to the 'Benefit Amount prompt' field in the Card Data Inquiry response.

Please note, in the Refund response the amount that has been credited will be returned in the 'Benefit' field as a negative value.

When performing multiple refunds for the same order, the initial refund can be performed using the refund transaction. All subsequent refunds for the order must be submitted as an Independent Refund.

For example, if the original Purchase was for $1.00, the Refund 'Benefit' will be -1.00

```
namespace Moneris
{
  using System;
    public class TestLoyaltyRefund
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        string txn_number = args[3];
        string total_amount = args[4];
        string benefit_amount = args[5];
        string info = "";

        ErnexLoyaltyRefund ernexLoyaltyRefund = new ErnexLoyaltyRefund(order_id, txn_number, total_amount,
                                              benefit_amount);

        ernexLoyaltyRefund.SetInfo(info); //optional

        ErnexHttpsPostRequest mpgReq =
            new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyRefund);

        try
        {
          ErnexReceipt receipt = mpgReq.GetReceipt();

          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
          Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
          Console.WriteLine("TransTime = " + receipt.GetTransTime());
          Console.WriteLine("TransDate = " + receipt.GetTransDate());
          Console.WriteLine("TransType = " + receipt.GetTransType());
```

```
          Console.WriteLine("Message = " + receipt.GetMessage());
          Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
          Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
          Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
          Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
          Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
          Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
          Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
          Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
          Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
          Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
          Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
          Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
          Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
          Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
          Console.WriteLine("Benefit = " + receipt.GetBenefit());
          Console.WriteLine("Language = " + receipt.GetLanguage());
          Console.WriteLine("Error Code = " + receipt.GetErrorCode());
          Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
          Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
          Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
          Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

          foreach (string cardCode in receipt.GetCardCodes())
          {
            Console.WriteLine("cardCode = " + cardCode);
            Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
            Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
            Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
            Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
            Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
            Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
            Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
            Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
            Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
            Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
            Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
            Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
            Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
            Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
            Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
            Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
            Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
            Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
            Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

            foreach (string recordType in receipt.GetRecordType(cardCode))
            {
              Console.WriteLine("recordType = " + recordType);
              Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
              Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                recordType));
              Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                recordType));
              Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            }
          }
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
      }

    } // end TestLoyaltyRefund
}
```

Independent Refund

The Independent Refund will credit a specified amount to the loyalty card. The Independent Refund does not require an existing order to be logged in the eSELECTplus gateway; however, the card number and expiry date will need to be passed. With respect to batch totals Independent Refund transactions are added to the batch Refund Count and Refund Total statistics.

Some card programs (i.e. CardCode) may not allow Refunds; the merchant must first check the 'RefundAllowed' parameter of the card program in the Card Data Inquiry transaction. Please refer to section 15 for further details.

To send an Independent Refund we require several variables (store_id, api_token, order_id, total_amount, track2 and/or pan, expdate, language_code, and benefit_amount). Similar to the Purchase transaction, the card may be either swiped or manually keyed-in. The 'reference_number' variable is used to refer back to a previously processed Purchase – this may have been processed via eSELECTplus or a different gateway. For transactions processed via eSELECTplus, the 'reference_number' is returned in the 'HostReferenceNum' field in the Purchase transaction response. Please refer to section 19 for further details on how to populate the 'language_code' variable. To determine the rules regarding populating the value of the 'benefit_amount" in the request, please refer to the 'Benefit Amount prompt' field in the Card Data Inquiry response.

There are also 3 extra variables (info, cvd_value and reference_number) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. To determine if the reference_number is mandatory in the request, please refer to the "RefundAllowed" field. Please refer to section 14 for all variable definitions.

Please note, in the Independent Refund response the amount that has been credited will be returned in the 'Benefit' field as a negative value.

```
namespace Moneris
{
  using System;
    public class TestLoyaltyIndRefund
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        string cust_id = "customer1";  //optional
        string total_amount = args[3];
        string track2 = "";
        //Console.WriteLine("Please swipe card");
        //track1 = Console.ReadLine();
        //track2 = Console.ReadLine();

        string pan = args[4];
        string expdate = args[5];
        string cvd_value = args[6];
        string info = args[7];
        string language_code = args[8];
        string reference_number = args[9];
        string benefit_amount = args[10];

        ErnexLoyaltyIndRefund ernexLoyaltyIndRefund = new ErnexLoyaltyIndRefund(order_id, cust_id,
                total_amount, track2, pan, expdate, language_code, benefit_amount);

        ernexLoyaltyIndRefund.SetCvdValue(cvd_value); //optional
        ernexLoyaltyIndRefund.SetInfo(info); //optional
        ernexLoyaltyIndRefund.SetReferenceNumber(reference_number); //optional

        ErnexHttpsPostRequest mpgReq =
            new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyIndRefund);
        try
        {
          ErnexReceipt receipt = mpgReq.GetReceipt();
          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
```

```
            Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
            Console.WriteLine("TransTime = " + receipt.GetTransTime());
            Console.WriteLine("TransDate = " + receipt.GetTransDate());
            Console.WriteLine("TransType = " + receipt.GetTransType());
            Console.WriteLine("Message = " + receipt.GetMessage());
            Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
            Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
            Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
            Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
            Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
            Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
            Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
            Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
            Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
            Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
            Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
            Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
            Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
            Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
            Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
            Console.WriteLine("Benefit = " + receipt.GetBenefit());
            Console.WriteLine("Language = " + receipt.GetLanguage());
            Console.WriteLine("Error Code = " + receipt.GetErrorCode());
            Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
            Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
            Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
            Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

            foreach (string cardCode in receipt.GetCardCodes())
            {
              Console.WriteLine("cardCode = " + cardCode);
              Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
              Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
              Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
              Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
              Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
              Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
              Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
              Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
              Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
              Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
              Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
              Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
              Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
              Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
              Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
              Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
              Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
              Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
              Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

              foreach (string recordType in receipt.GetRecordType(cardCode))
              {
                  Console.WriteLine("recordType = " + recordType);
                  Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
                  Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                recordType));
                  Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                recordType));
                  Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode,
                                recordType));
                  Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
                  Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
              }
            }
          }
          catch (Exception e)
          {
            Console.WriteLine(e);
          }
        }

    } // end TestLoyaltyIndRefund
}
```

Pre-Authorization

A Pre-authorization (Authorization) transaction is used to collect the loyalty card number and determine the potential rewards to be awarded for a purchase that will be completed at a later date for less than, equal to or above the pre-authorized amount.

In the Pre-Auth example we require several variables (store_id, api_token, order_id, total_amount, track2 and/or pan, expdate, language_code and benefit amount). Please refer to section 19 for further details on how to populate the 'language_code' variable. The merchant may both swipe the card using a Mag Swipe Reader (MSR) and pass the data in the 'track2' variable, or, the card information may be manually keyed-in and passed using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction. To determine the rules regarding populating the value of the 'benefit_amount" in the request, please refer to the 'BenefitAmountPrompt' field in the Card Data Inquiry response.

There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. Please refer to section 14 for all variable definitions

The Pre-Auth response will indicate the CurrentBalance and LifetimeBalance including the points that will be awarded upon the full completion of the capture transaction.

```
namespace Moneris
    {
    using System;
    public class TestLoyaltyPreauth
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        //string cust_id = "customer1";  //optional
        string total_amount = args[3];
        string track2 = "";

        //Console.WriteLine("Please swipe card");
        //track1 = Console.ReadLine();
        //track2 = Console.ReadLine();

        string pan = args[4];
        string expdate = args[5];
        string cvd_value = args[6];
        string language_code = args[7];
        string info = args[8];
        string benefit_amount = args[9];
        string payment_type = args[10];

        ErnexLoyaltyPreauth ernexLoyaltyPreauth = new ErnexLoyaltyPreauth(order_id, total_amount,
        track2, pan, expdate, language_code, benefit_amount, payment_type);

        ernexLoyaltyPreauth.SetCvdValue(cvd_value); //optional
        ernexLoyaltyPreauth.SetInfo(info); //optional

        ErnexHttpsPostRequest mpgReq =
            new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyPreauth);

        try
        {
            ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
```

```
    Console.WriteLine("TransTime = " + receipt.GetTransTime());
    Console.WriteLine("TransDate = " + receipt.GetTransDate());
    Console.WriteLine("TransType = " + receipt.GetTransType());
    Console.WriteLine("Message = " + receipt.GetMessage());
    Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
    Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
    Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
    Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
    Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
    Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
    Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
    Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
    Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
    Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
    Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
    Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
    Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
    Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
    Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
    Console.WriteLine("Benefit = " + receipt.GetBenefit());
    Console.WriteLine("Language = " + receipt.GetLanguage());
    Console.WriteLine("Error Code = " + receipt.GetErrorCode());
    Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
    Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
    Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
    Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

    foreach (string cardCode in receipt.GetCardCodes())
    {
        Console.WriteLine("cardCode = " + cardCode);
        Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
        Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
        Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
        Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
        Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
        Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
        Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
        Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
        Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
        Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
        Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
        Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
        Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
        Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
        Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
        Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
        Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
        Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
        Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

        foreach (string recordType in receipt.GetRecordType(cardCode))
        {
            Console.WriteLine("recordType = " + recordType);
            Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
            Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                recordType));
            Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                recordType));
            Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode,
                                recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
        }
    }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
  }

} // end TestLoyaltyPreauth
}
```

Completion

The completion transaction is used to complete a previously authorized pre-authorization transaction and actually adjusts the cards balance. The completion amount may be less than, equal to, or greater than the amount that was already pre-authorized.

To send a Completion there are several variables (e.g. order_id, txn_number, total_amount and benefit_amount) from the Pre-Auth are required. There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. Please refer to section 14 for all variable definitions. . To determine the rules regarding populating the value of the 'benefit_amount" in the request, please refer to the 'BenefitAmountPrompt' field in the Card Data Inquiry response.

```
namespace Moneris
{
  using System;
    public class TestLoyaltyCompletion
    {
      public static void Main(string[] args)
      {
        string host = "esqa.moneris.com";
        string store_id = args[0];
        string api_token = args[1];
        string order_id = args[2];
        string txn_number = args[3];
        string total_amount = args[4];
        string benefit_amount = args[5];
        string info = "";

        ErnexLoyaltyCompletion ernexLoyaltyCompletion = new ErnexLoyaltyCompletion(order_id, txn_number,
        total_amount, benefit_amount);

        ernexLoyaltyCompletion.SetInfo(info); //optional

        ErnexHttpsPostRequest mpgReq =
            new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyCompletion);

        try
        {
          ErnexReceipt receipt = mpgReq.GetReceipt();

          Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
          Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
          Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
          Console.WriteLine("TransTime = " + receipt.GetTransTime());
          Console.WriteLine("TransDate = " + receipt.GetTransDate());
          Console.WriteLine("TransType = " + receipt.GetTransType());
          Console.WriteLine("Message = " + receipt.GetMessage());
          Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
          Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
          Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
          Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
          Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
          Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
          Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
          Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
          Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
          Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
          Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
          Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
          Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
          Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
          Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
          Console.WriteLine("Benefit = " + receipt.GetBenefit());
          Console.WriteLine("Language = " + receipt.GetLanguage());
          Console.WriteLine("Error Code = " + receipt.GetErrorCode());
```

```
          Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
          Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
          Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
          Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

          foreach (string cardCode in receipt.GetCardCodes())
          {
            Console.WriteLine("cardCode = " + cardCode);
            Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
            Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
            Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
            Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
            Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
            Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
            Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
            Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
            Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
            Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
            Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
            Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
            Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
            Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
            Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
            Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
            Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
            Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
            Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

            foreach (string recordType in receipt.GetRecordType(cardCode))
            {
                Console.WriteLine("recordType = " + recordType);
                Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
                Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                                recordType));
                Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode,
                                recordType));
                Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode,
                                recordType));
                Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
                Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            }

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

Redemption

The redemption transaction allows the cardholder to receive value/purchase goods and services with their accumulated rewards.

In the Redemption example we require several variables (store_id, api_token, order_id, track2 and/or pan, expdate, language_code and benefit and/or total_amount). Please refer to section 19 for further details on how to populate the 'language_code' variable. The merchant may both swipe the card using a Mag Swipe Reader (MSR) and pass the data in the 'track2' variable. Or, the card information may be manually keyed-in and passed using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction. To determine the rules regarding populating the value of the "benefit" or "total_amount" in the request, please refer to the 'RedemptionType' field in the Card Data Inquiry response.

There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program. To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response. Please refer to section 14 for all variable definitions

```
namespace Moneris
{
  using System;
  public class TestLoyaltyRedemption
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string order_id = args[2];
      //string cust_id = "customer1";  //optional
      string total_amount = args[3];
      string track2 = "";

      //Console.WriteLine("Please swipe card");
      //track1 = Console.ReadLine();
      //track2 = Console.ReadLine();

      string pan = args[4];
      string expdate = args[5];
      string cvd_value = args[6];
      string language_code = args[7];
      string info = args[8];
      string benefit = args[9];

      ErnexLoyaltyRedemption ernexLoyaltyRedemption = new ErnexLoyaltyRedemption(order_id, total_amount,
      track2, pan, expdate, language_code, benefit);

      ernexLoyaltyRedemption.SetCvdValue(cvd_value); //optional
      ernexLoyaltyRedemption.SetInfo(info); //optional

      ErnexHttpsPostRequest mpgReq =
          new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyRedemption);

      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
```

```
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
          Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
            Console.WriteLine("recordType = " + recordType);
            Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
            Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                recordType));
            Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode, recordType));
            Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }
        }
      }
      catch (Exception e)
      {
          Console.WriteLine(e);
      }
    }
  } // end TestLoyaltyRedemption
}
```

6. Administrative Transaction Examples

In the previous section the instructions were provided for the financial transaction set. eSELECTplus also provides several administrative transactions. These transactions allow the merchant to perform maintenance functions.

Initialization

Before Loyalty transactions can be processed, an initialization transaction must be completed. This transaction will return the card configuration information in the 'InitData' section of the response. This information can be used by the merchant to identify cards that are eligible for loyalty card processing and to determine how to complete the various transactions - what fields are mandatory or optional within the request.

The Initialization transaction may generate more than one response message. Each response message will include card configuration information for one card program – each message will include a specific CardCode to identify it. If there is more than one card setup for this merchant account, there will be more than one response message.

For each card program setup at Moneris, three response messages will be returned. Each message includes a different RecordType value to allow the merchant to identify the information. Each block of data also includes the Card Code which is the primary identifier for the data.

```
namespace Moneris
{
  using System;
  public class TestInitialization
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string ecr_no = args[2];

      ErnexHttpsPostRequest mpgReq =
          new ErnexHttpsPostRequest(host, store_id, api_token,
          new ErnexInitialization (ecr_no));
      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());
```

```
        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
             Console.WriteLine("recordType = " + recordType);
             Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }
        }
      }
      catch (Exception e)
      {
          Console.WriteLine(e);
      }
    }

  } // end TestInitialization
}
```

Activation

An Activation transaction allows the merchant to enable a new loyalty card. With respect to batch totals Activation transactions are added to the batch Activation Count and Activation Total statistics. To perform an Activation transaction, the merchant may both swipe the card using a Mag Swipe Reader (MSR) and pass the data in the 'track2' variable or the card information may be manually keyed-in and passed using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction. As seen in the example below, the card has been swiped and the 'pan' and 'expdate' fields have been left blank.

In an Activation transaction several of the variables are mandatory (host, store_id, api_token, order_id, track2 and/or pan, expdate, initial_amount and language_code). Please refer to section 19 for further details on how to populate the 'language_code' variable. There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program (i.e. CardCode). To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response.

Once the Activation has been successfully processed, please refer to the 'ActivationCharge' field in the response to verify the amount that the customer has to be charged.

```
namespace Moneris
{
  using System;

  public class TestLoyaltyActivation
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string order_id = args[2];
      string cust_id = "customer1"; //optional
      string track2 = "";
      //Console.WriteLine("Please swipe card");
      //track1 = Console.ReadLine();
      //track2 = Console.ReadLine();
      string initial_amount = args[3];
      string info = args[4];
      string language_code = args[5];
      string cvd_value = args[6];
      string pan = args[7];
      string expdate = args[8];

      ErnexLoyaltyActivation ernexLoyaltyActivation = new ErnexLoyaltyActivation (order_id, cust_id,
      initial_amount, track2, pan, expdate, language_code);

      ernexLoyaltyActivation.SetCvdValue(cvd_value); //optional
      ernexLoyaltyActivation.SetInfo(info); //optional

      ErnexHttpsPostRequest mpgReq =
          new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyActivation);

      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
```

```
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
          Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
            Console.WriteLine("recordType = " + recordType);
            Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
            Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                recordType));
            Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode, recordType));
            Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }

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

  } // end TestLoyaltyActivation
}
```

Deactivation

A Deactivation transaction allows the merchant to disable a loyalty card that may have been demagnetized or otherwise damaged. Please note, once a card has been deactivated it may no longer be reused. Also, the awards (if existent on the card) are not removed. To perform a Deactivation the merchant may both swipe the card using a Mag Swipe Reader (MSR) and pass the data in the 'track2' variable, or the card information may be manually keyed-in and passed using the 'pan' and 'expdate' variables. If all three fields are submitted, the track2 details will be used to process the transaction.

In the example below, several of the variables are mandatory (host, store_id, api_token, order_id, track2 and/or pan, expdate, language_code). Please refer to section 19 for further details on how to populate the 'language_code' variable. There are also 2 extra variables (info and cvd_value) that may be either mandatory or optional, depending on the card program (i.e. CardCode). To determine whether the merchant must prompt for the 'info' field, please refer to the 'InfoPrompt' returned in the Card Data Inquiry response – please see section 15 for further details. If the merchant prompts for this field, the user may still choose not to populate it. In this case the 'info' field will be sent in with an empty value. To determine whether the 'cvd_value' is mandatory in the request, please refer to the 'CVCprompt' field in the Card Data Inquiry response.

Once the card has been successfully Deactivated please refer to the 'RemainingBalance' field in the response to verify the points which are now unusable. For example, if the card was Deactivated due to loss or damage the merchant may choose to award the "RemainingBalance" points to a new card.

```
namespace Moneris
{
  using System;
  public class TestLoyaltyDeactivation
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string order_id = args[2];
      //Console.WriteLine("Please swipe card");
      //track1 = Console.ReadLine();
      //track2 = Console.ReadLine();
      string track2 = "";
      string info = args[3];
      string pan = args[4];
      string expdate = args[5];
      string language_code = args[6];
      string cvd_value = args[7];

      ErnexLoyaltyDeactivation ernexLoyaltyDeactivation = new ErnexLoyaltyDeactivation (order_id, track2,
      pan, expdate, language_code);

      ernexLoyaltyDeactivation.SetCvdValue(cvd_value); //optional
      ernexLoyaltyDeactivation.SetInfo(info); //optional

      ErnexHttpsPostRequest mpgReq =
          new ErnexHttpsPostRequest(host, store_id, api_token, ernexLoyaltyDeactivation);

      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
```

```
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("LifetimeBalance = " + receipt.GetLifetimeBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("AmountPrompt = " + receipt.GetAmountPrompt(cardCode));
          Console.WriteLine("BenefitPrompt = " + receipt.GetBenefitPrompt(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
            Console.WriteLine("recordType = " + recordType);
            Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
            Console.WriteLine("BenefitDescription = " + receipt.GetBenefitDescription(cardCode,
                              recordType));
            Console.WriteLine("BenefitPromptText = " + receipt.GetBenefitPromptText(cardCode, recordType));
            Console.WriteLine("AmountPromptText = " + receipt.GetAmountPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }
        }
      }
      catch (Exception e)
      {
          Console.WriteLine(e);
      }
    }

  } // end TestLoyaltyDeactivation
}
```

Card Inquiry

The Card Inquiry transaction allows a merchant to retrieve details about a specific cardholder account. This transaction is primarily used to determine the current balance and lifetime balance for a card, but can also be used to determine a cardholder's name and telephone number. The Card Inquiry transaction also returns the 'InitData' section in the response, similar to the Card Data Inquiry. It allows the merchant to verify card configuration information without having to re-initialize the terminal. For example, the 'CVCPrompt' field will be returned which determines if the 'cvd_value' is mandatory in the request.

```
namespace Moneris
{
  using System;
  public class TestCardInquiry
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string track2 = "";
      //Console.WriteLine("Please swipe card");
      //track1 = Console.ReadLine();
      //track2 = Console.ReadLine();
      string pan = args[2];
      string expdate = args[3];
      string language_code = args[4];
      string cvd_value = args[5];

      ErnexGiftCardInq ernexGiftCardInq = new ErnexGiftCardInq (track2, pan, expdate, language_code);

      ernexGiftCardInq.SetCvdValue(cvd_value);

      ErnexHttpsPostRequest mpgReq =new ErnexHttpsPostRequest(host,store_id,api_token,ernexGiftCardInq);

      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
```

```
            Console.WriteLine("cardCode = " + cardCode);
            Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
            Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
            Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
            Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
            Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
            Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
            Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
            Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
            Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
            Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
            Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
            Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
            Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
            Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
            Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
            Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
            Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

            foreach (string recordType in receipt.GetRecordType(cardCode))
            {
              Console.WriteLine("recordType = " + recordType);
              Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
              Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
            }

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

    }
}
```

Card Data Inquiry

The Card Data Inquiry transaction allows a merchant to retrieve configuration details about a specific card. This transaction is primarily used to determine which cards are eligible for loyalty card processing and to determine how to complete the various transactions – which fields are mandatory or optional within the request. The card configuration details will be returned in the 'InitData' section of the response.

```
namespace Moneris
{
  using System;
  public class TestCardDataInquiry
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = "monca00002";
      string api_token = "giftguy";
      string track2 = "";
      //Console.WriteLine("Please swipe card");
      //track1 = Console.ReadLine();
      //track2 = Console.ReadLine();
      string pan = "0311040000001000234";
      string expdate = "1111";

      ErnexHttpsPostRequest mpgReq =
          new ErnexHttpsPostRequest(host, store_id, api_token,
          new ErnexCardDataInq (track2, pan, expdate));
      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();

        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("HostReferenceNum = " + receipt.GetHostReferenceNum());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TransCardCode = " + receipt.GetTransCardCode());
        Console.WriteLine("TransCardType = " + receipt.GetTransCardType());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
        Console.WriteLine("HostTotals = " + receipt.GetHostTotals());
        Console.WriteLine("DisplayText = " + receipt.GetDisplayText());
        Console.WriteLine("ReceiptText = " + receipt.GetReceiptText());
        Console.WriteLine("CardHolderName = " + receipt.GetCardHolderName());
        Console.WriteLine("VoucherType = " + receipt.GetVoucherType());
        Console.WriteLine("VoucherText = " + receipt.GetVoucherText());
        Console.WriteLine("InitialAmount = " + receipt.GetInitialAmount());
        Console.WriteLine("InitialBalance = " + receipt.GetInitialBalance());
        Console.WriteLine("BatchNo = " + receipt.GetBatchNo());
        Console.WriteLine("CurrentBalance = " + receipt.GetCurrentBalance());
        Console.WriteLine("Benefit = " + receipt.GetBenefit());
        Console.WriteLine("Language = " + receipt.GetLanguage());
        Console.WriteLine("Error Code = " + receipt.GetErrorCode());
        Console.WriteLine("Error Message = " + receipt.GetErrorMessage());
        Console.WriteLine("ActivationCharge = " + receipt.GetActivationCharge());
        Console.WriteLine("RemainingBalance = " + receipt.GetRemainingBalance());
        Console.WriteLine("CardStatus = " + receipt.GetCardStatus());

        foreach (string cardCode in receipt.GetCardCodes())
        {
          Console.WriteLine("cardCode = " + cardCode);
          Console.WriteLine("CardCardType = " + receipt.GetCardCardType(cardCode));
          Console.WriteLine("CheckMod10 = " + receipt.GetCheckMod10(cardCode));
          Console.WriteLine("CheckLanguage = " + receipt.GetCheckLanguage(cardCode));
          Console.WriteLine("CVCPrompt = " + receipt.GetCVCPrompt(cardCode));
          Console.WriteLine("InfoPrompt = " + receipt.GetInfoPrompt(cardCode));
          Console.WriteLine("InitialAmountPrompt = " + receipt.GetInitialAmountPrompt(cardCode));
          Console.WriteLine("RefundAllowed = " + receipt.GetRefundAllowed(cardCode));
```

```
          Console.WriteLine("CardLengthMinimum = " + receipt.GetCardLengthMinimum(cardCode));
          Console.WriteLine("CardLengthMaximum = " + receipt.GetCardLengthMaximum(cardCode));
          Console.WriteLine("LowBIN1 = " + receipt.GetLowBIN1(cardCode));
          Console.WriteLine("HighBIN1 = " + receipt.GetHighBIN1(cardCode));
          Console.WriteLine("LowBIN2 = " + receipt.GetLowBIN2(cardCode));
          Console.WriteLine("HighBIN2 = " + receipt.GetHighBIN2(cardCode));
          Console.WriteLine("LowBIN3 = " + receipt.GetLowBIN3(cardCode));
          Console.WriteLine("HighBIN3 = " + receipt.GetHighBIN3(cardCode));
          Console.WriteLine("LowBIN4 = " + receipt.GetLowBIN4(cardCode));
          Console.WriteLine("HighBIN4 = " + receipt.GetHighBIN4(cardCode) + "\n");

          foreach (string recordType in receipt.GetRecordType(cardCode))
          {
            Console.WriteLine("recordType = " + recordType);
            Console.WriteLine("CardDescription = " + receipt.GetCardDescription(cardCode, recordType));
            Console.WriteLine("InfoPromptText = " + receipt.GetInfoPromptText(cardCode, recordType));
          }

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

Batch Close

At the end of every day the Batch needs to be closed in order for the merchant to be able to perform their end-of-day reconciliation. *By default eSELECTplus will close your Batch automatically for you daily whenever there are transaction totals 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 for each card program (i.e. CardCode).

The break down of the Batch Totals is as follows:

Field Name	*Field Value*
PurchaseCount	Total number of Purchases.
PurchaseTotal	Total dollar ($) amount of Purchases.
PurchaseBenefitTotal	Total points awarded during purchase transactions
RefundCount	Total number of Refunds and Independent Refunds.
RefundTotal	Total dollar ($) amount of Refunds and Independent Refunds.
RefundBenefitTotal	Total points refunded
ActivationCount	Count = Total number of Activations - the number of voided Activations
ActivationTotal	Total Amount = Total dollar ($) amount of Activations - the $ amount of voided Activations
CorrectionCount	Total number of Voids = Voided Purchases + Voided Activations + Voided Refunds
CorrectionTotal	Defaults to 0
RedemptionCount	Total number of Redemption transactions.
RedemptionTotal	Total dollar ($) amount of Redemptions
RedemptionBenefitTotal	Total points redeemed

```
namespace Moneris
{
using System;

public class TestGiftBatchClose
{
  public static void Main(string[] args)
  {
  string host = "esqa.moneris.com";
  string store_id = args[0];
  string api_token = args[1];
  string ecr_no = args[2];

  ErnexHttpsPostRequest mpgReq =
      new ErnexHttpsPostRequest(host, store_id, api_token,
      new ErnexBatchclose (ecr_no));
  try
  {
      ErnexReceipt receipt = mpgReq.GetReceipt();
      Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());

      if ( (receipt.GetReceiptId()).Equals("Global Error Receipt") )
      {
        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("TimedOut = " + receipt.GetTimedOut());
      }
      else
      {
        foreach (string term_id in receipt.GetTerminalIDs())
        {
          Console.WriteLine("Terminal = " + term_id);
```

```
            Console.WriteLine("Closed = " + receipt.GetClosed(term_id) + "\n");

            foreach (string cardCode in receipt.GetEcrCardCodes(term_id))
            {
             Console.WriteLine("CardCode = " + cardCode);
             Console.WriteLine("PurchaseCount = " + receipt.GetPurchaseCount(term_id, cardCode));
             Console.WriteLine("PurchaseTotal = " + receipt.GetPurchaseTotal(term_id, cardCode));
             Console.WriteLine("PurchaseBenefitTotal = " + receipt.GetPurchaseBenefitTotal(term_id,
                               cardCode));
             Console.WriteLine("RefundCount = " + receipt.GetRefundCount(term_id, cardCode));
             Console.WriteLine("RefundTotal = " + receipt.GetRefundTotal(term_id, cardCode));
             Console.WriteLine("RefundBenefitTotal = " + receipt.GetRefundBenefitTotal(term_id, cardCode));
             Console.WriteLine("RedemptionCount = " + receipt.GetRedemptionCount(term_id, cardCode));
             Console.WriteLine("RedemptionTotal = " + receipt.GetRedemptionTotal(term_id, cardCode));
             Console.WriteLine("RedemptionBenefitTotal = " + receipt.GetRedemptionBenefitTotal(term_id,
                               cardCode));
             Console.WriteLine("ActivationCount = " + receipt.GetActivationCount(term_id, cardCode));
             Console.WriteLine("ActivationTotal = " + receipt.GetActivationTotal(term_id, cardCode));
             Console.WriteLine("CorrectionCount = " + receipt.GetCorrectionCount(term_id, cardCode));
             Console.WriteLine("CorrectionTotal = " + receipt.GetCorrectionTotal(term_id, cardCode));
             Console.WriteLine("Error Code = " + receipt.GetErrorCode());
             Console.WriteLine("Error Message = " + receipt.GetErrorMessage() + "\n");
            }
        }
      }
    }
    catch (Exception e)
    {
      Console.WriteLine(e);
    }
    }

} // end TestGiftBatchClose
}
```

Open Totals

Open Totals allows the merchant to retrieve details about all loyalty 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 and incrementing the Batch number.

```
namespace Moneris
{
  using System;
  public class TestGiftOpenTotals
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string ecr_no = args[2];
      ErnexHttpsPostRequest mpgReq =
      new ErnexHttpsPostRequest(host, store_id, api_token,
      new ErnexOpentotals (ecr_no));
      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();
        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        if ( (receipt.GetReceiptId()).Equals("Global Error Receipt") )
        {
           Console.WriteLine("TransType = " + receipt.GetTransType());
           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("TimedOut = " + receipt.GetTimedOut());
        }
        else
        {
          foreach (string term_id in receipt.GetTerminalIDs())
          {
            Console.WriteLine("Terminal = " + term_id);
            Console.WriteLine("Closed = " + receipt.GetClosed(term_id) + "\n");

            foreach (string cardCode in receipt.GetEcrCardCodes(term_id))
            {
              Console.WriteLine("CardCode = " + cardCode);
              Console.WriteLine("PurchaseCount = " + receipt.GetPurchaseCount(term_id, cardCode));
              Console.WriteLine("PurchaseTotal = " + receipt.GetPurchaseTotal(term_id, cardCode));
              Console.WriteLine("PurchaseBenefitTotal = " + receipt.GetPurchaseBenefitTotal(term_id,
                                cardCode));
              Console.WriteLine("RefundCount = " + receipt.GetRefundCount(term_id, cardCode));
              Console.WriteLine("RefundTotal = " + receipt.GetRefundTotal(term_id, cardCode));
              Console.WriteLine("RefundBenefitTotal = " + receipt.GetRefundBenefitTotal(term_id,
                                cardCode));
              Console.WriteLine("RedemptionCount = " + receipt.GetRedemptionCount(term_id, cardCode));
              Console.WriteLine("RedemptionTotal = " + receipt.GetRedemptionTotal(term_id, cardCode));
              Console.WriteLine("RedemptionBenefitTotal = " + receipt.GetRedemptionBenefitTotal(term_id,
                                cardCode));
              Console.WriteLine("ActivationCount = " + receipt.GetActivationCount(term_id, cardCode));
              Console.WriteLine("ActivationTotal = " + receipt.GetActivationTotal(term_id, cardCode));
              Console.WriteLine("CorrectionCount = " + receipt.GetCorrectionCount(term_id, cardCode));
              Console.WriteLine("CorrectionTotal = " + receipt.GetCorrectionTotal(term_id, cardCode));
              Console.WriteLine("Error Code = " + receipt.GetErrorCode());
              Console.WriteLine("Error Message = " + receipt.GetErrorMessage() + "\n");
            }
          }
        }
      }catch (Exception e)
      {Console.WriteLine(e);
      }
    }
  } // end TestGiftOpenTotals
}
```

Group

The Group transaction is primarily intended to allow the merchant to link multiple transactions. If the merchant is capable of processing both loyalty card and credit card transactions, they may wish to link these two types of transactions for reporting purposes. To register a Group the following variables must be submitted: host, store_id, api_token, order_id, txn_number, group_ref_num and group_type. Please refer to section 14 for all request variable definitions.

NOTE	The purpose of the Group is to link transactions so they may be displayed together within the Merchant Resource Centre (MRC) reports. If the merchant does not require the MRC's reports, they may then choose to omit this transaction type.

There are two scenarios for registering the Group: ERNEX and Financial (i.e. Credit or Debit card)

Scenario 1: ERNEX
This scenario requires the 'order_id' and 'txn_number' to refer to a previously submitted ERNEX loyalty card transaction (i.e. Purchase, Activation, etc). The 'group_type' must then state "ernex". The 'group_ref_num' variable allows the merchant to submit any identifier for this particular group.

For example, the merchant may process many different ERNEX transactions on the same card that they wish to link together. Firstly, the merchant processes the ERNEX transactions as follows:

1. Process an ERNEX Activation on card A
   - request order_id = ernex_activ_20092007
   - in the response the TxnNumber = 1234-1189610340337-00016927_5
2. Process an ERNEX Purchase on card A for $10
   - request order_id = ernex_purch_20092007_1
   - in the response the TxnNumber = 2345-1189610340337-00016927_5
3. Process another ERNEX Purchase on card A for $100
   - request order_id = ernex_purch_20092007_2
   - in the response the TxnNumber = 3456-1189610340337-00016927_5

Now, to link all three ERNEX transactions together we submit three Group transactions as follows (note, all three submit the same 'group_ref_num' and the 'group_type' is always set to "ernex"):

1. Submit a Group with the following:
   - order_id = 'ernex_activ_20092007'
   - txn_number = '1234-1189610340337-00016927_5'
   - **group_ref_num = 'my unique reference number'**
   - **group_type = 'ernex'**
2. Submit a 2nd Group with the following:
   - order_id = 'ernex_purch_20092007_1'
   - txn_number = '2345-1189610340337-00016927_5'
   - **group_ref_num = 'my unique reference number'**
   - **group_type = 'ernex'**
3. Submit a 3rd Group with the following:
   - order_id = 'ernex_purch_20092007_2'
   - txn_number = '3456-1189610340337-00016927_5'
   - **group_ref_num = 'my unique reference number'**
   - **group_type = 'ernex'**

Within the Merchant Resource Centre the Order History page will now display the details for all three ERNEX transactions, no matter which order_id is selected (i.e. 1234-1189610340337-00016927_5).

Scenario 2: Financial
This scenario requires the 'order_id' and 'txn_number' to refer back to a previously submitted financial transaction (i.e. Credit Card Purchase, PreAuth (Authorization), ACH, etc). The 'group_type' must then state "financial". The 'group_ref_num' variable allows the merchant to submit any identifier for this particular group.

For example, the merchant may process an ERNEX Purchase transaction and then follow that with a Credit Card Purchase to cover the remainder of the purchase amount. Firstly the merchant will process the Purchase transactions as follows:

1. Process an ERNEX Purchase on card A for $10 (the full balance of the card)
   - request order_id = ernex_purchase_20092007
   - in the response the TxnNumber = 9999-1189610340337-00016927_5
2. Process a Credit Card Purchase on card B for $40 (the remainder of the customer's bill)
   - request order_id = credit_card_purchase_20092007
   - in the response the TxnNumber = 904-0_16

Now, to link these two transactions together we submit 2 Group transactions as follows (note, both submit the same 'group_ref_num' but the 'group_type' varies to define the type of the original transaction "ernex" or "financial"):

1. Submit a Group with the following:
   - order_id = 'ernex_purchase_20092007'
   - txn_number = '9999-1189610340337-00016927_5'
   - **group_ref_num = 'some unique ref num'**
   - **group_type = 'ernex'**
2. Submit a 2nd Group with the following:
   - order_id = 'credit_card_purchase_20092007'
   - txn_number = '904-0_16'
   - **group_ref_num = 'some unique ref num'**
   - **group_type = 'financial'**

NOTE	Please note the Group transaction is intended to link transactions related to the ERNEX program. This is not intended for linking multiple financial transactions alone. At least one "ernex" group_type must be submitted.

In the example below, we see a Group being created where the original transaction was an ERNEX transaction and this group's reference number (group_ref_num) is "group1".

```
namespace Moneris
{
  using System;
  public class TestGroup
  {
    public static void Main(string[] args)
    {
      string host = "esqa.moneris.com";
      string store_id = args[0];
      string api_token = args[1];
      string order_id = args[2];
      string txn_number = args[3];
      string group_ref_num = args[4];
      string group_type = args[5];

      ErnexHttpsPostRequest mpgReq =  new ErnexHttpsPostRequest(host, store_id, api_token,
          new Group (order_id, txn_number, group_ref_num, group_type));
      try
      {
        ErnexReceipt receipt = mpgReq.GetReceipt();
        Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
        Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
        Console.WriteLine("TransTime = " + receipt.GetTransTime());
        Console.WriteLine("TransDate = " + receipt.GetTransDate());
        Console.WriteLine("TransType = " + receipt.GetTransType());
        Console.WriteLine("Complete = " + receipt.GetComplete());
        Console.WriteLine("Message = " + receipt.GetMessage());
        Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
        Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
      }
      catch (Exception e)
      {
          Console.WriteLine(e);
      }
    }
  } // end TestInitialization
}
```

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

When printing a receipt for a transaction, there are a number of fields that must be included. The actual look and feel of the receipt is up to the merchant as long as the specified fields are printed. Receipts must be printed for approved and declined transactions. Receipt printing for the Card Inquiry transaction can be optional.

Sample receipts are provided in section 21. The following information must be printed:

	Field	Description
1	Location description	Merchant's store / business name and address. If this is a web based business, it must also include the web site address.
2	Terminal ID	This field is also referred to as the ECR number (for example, 00016927). It is a 6-8 digit number that identifies the specific terminal that processed the transaction. Returned as part of the 'TxnNumber' field in the transaction response.
3	Transaction description	The type of transaction that was performed: - Purchase - Refund - Void - Activation *or* Activate - Deactivation *or* Deactivate
4	Card description	The text for this field is returned in the Card Data Inquiry response in the 'CardDescription' field.
5	Card number	The customer's copy of the receipt must have all but the last 4 digits of card number masked out.
6	Transaction amount	The total amount of the transaction.
7	Benefit Amount	The amount of the transaction that is eligible for points to be awarded or redeemed on the loyalty card. This is the amount that is returned in the transaction response 'Benefit' field.
8	Current card balance	Optional. This amount is to be included if it is returned in the 'CurrentBalance' field in the transaction response.
9	Batch number	Optional. This field is to be included if it is returned in the 'BatchNo' field of the transaction response.
10	Reference number	Optional. This field is to be included if it is returned in the 'HostReferenceNum' field of the transaction response.
11	Info	Optional. It is to be included if it is configured and had been submitted in the original transaction request.
12	Receipt message	Optional. To be included if it is returned in the 'ReceiptText' field in the transaction response. Please note, the 'ReceiptText' field may include formatting codes that will outline how the text is to be printed on the receipt. Please refer to section 17 for further details.
13	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.
14	Voucher text	Optional. To be included if it is returned in the 'VoucherText' field in the transaction response. Please note, the 'VoucherText' field may include formatting codes that will outline how the text is to be printed on the receipt. Please refer to section 17 for further details.

15	Transaction Points	The total amount of points that were awarded/redeemed during the transaction
16	Lifetime Balance	This is the current lifetime balance for the card. If the value is non-zero, it should be printed on the receipt. Not all programs maintain a lifetime balance. Some programs use this field as an annual balance. The text printed for this field on a receipt should be drawn from the Totals Description field received in the Initialization transaction response.

Voucher

A voucher should be printed whenever the response includes a non-zero value in the 'VoucherType' field. Vouchers are used for instant win and coupons. The 'VoucherType' field indicates whether the voucher is for instant win or a coupon and allows the merchant to format the voucher differently.

The voucher should be printed after the receipt and include a dividing line of stars or other characters.

The following information should be included on an Instant Win Voucher:

	Field	Description
1	Voucher text	This field is returned in the 'VoucherText' field in the transaction response. Please note, the 'VoucherText' field may include formatting codes that will outline how the text is to be printed on the receipt. Please refer to section 17 for further details.
2	Card description	The text for this field is returned in the Card Data Inquiry response in the 'CardDescription' field.
3	Card number	The customer's copy of the receipt must have all but the last 4 digits of card number masked out.
4	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.
5	Reference number	This field is to be included if it is returned in the 'HostReferenceNum' field of the transaction response. To redeem this voucher, the merchant will be required to process yet another Purchase transaction with this Reference number submitted in the "info" field. Please refer to section 20 for further details.
6	Received By line	This is a line that allows the cardholder to sign once they receive the voucher. This is similar to a credit/debit receipt's Signature Line.

The following information should be included on a Coupon Voucher:

	Field	Description
1	Voucher text	This field is returned in the 'VoucherText' field in the transaction response. Please note, the 'VoucherText' field may include formatting codes that will outline how the text is to be printed on the receipt. Please refer to section 17 for further details.
2	Card description	The text for this field is returned in the Card Data Inquiry response in the 'CardDescription' field.
3	Card number	The customer's copy of the receipt must have all but the last 4 digits of card number masked out.
4	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.
5	Reference number	This field is to be included if it is returned in the 'HostReferenceNum' field of the transaction response. To redeem this voucher, the merchant will be required to process yet another Purchase transaction with this Reference number submitted in the "info" field. Please refer to section 20 for further details.

8. 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 section 15.

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	Approved
301-812 (inclusive)	Declined
null	Incomplete

For a full list of response codes and the associated message please refer to the Response Code table available in section 22.

9. Certification Requirements?

Once you have completed the development and testing, your application must undergo a certification process where all the applicable transaction types must be demonstrated and the corresponding receipts properly generated. Please contact our Integration Support Help Desk for the Certification Test checklist that must be completed and returned to us for verification. Be sure to include the application version of your product. Any further changes to the product after certification would require a new re-certification. Once all the certification requirements are met, we will provide you with an official certification letter.

10. How Do I Test My Solution?

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

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

Test IDs			
eSelectPlus Canada		eSelectPlus US	
Host URL: https://esqa.moneris.com/gateway2/servlet/MpgRequest		Host URL: https://esplusqa.moneris.com/gateway_us/servlet/MpgRequest	
monca00002	yesguy	monusqa003	qatoken

11. How Do I Activate My Store?

Once you have received your activation letter/fax refer to the appropriate link below 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. This is where you may locate your production API Token.

eSelectPlus Canada	eSelectPlus US
https://www3.moneris.com/connect/en/activate/index.php	https://esplus.moneris.com/usmpg/activate

12. 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", refer to the appropriate link below. You will also need to change the store_id to reflect your production store ID and 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. You can use the store administrator ID you created during the activation process and then create additional users as needed.

eSelectPlus Canada	eSelectPlus US
https://www3.moneris.com/gateway2/servlet/MpgRequest	https://esplus.moneris.com/gateway_us/servlet/MpgRequest

13. How Do I Get Help?

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

For technical support: Phone: 1-866-319-7450 (Technical Difficulties) For integration support: Phone: 1-866-562-4354 Email: eselectplus@moneris.com	For technical support: Phone: 1-866-696-0488 (Technical Difficulties) 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. **For security reasons, please do not send us your API Token combined with your store ID, or your merchant number and terminal number in the same email.**

14. 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, Activation and Deactivation attempt. For Refunds and Voids the order_id must reference the original transaction.
cust_id	50 / an	This is an optional field that can be sent as part of a Purchase or Activation request. It is searchable from the Moneris Merchant Resource Centre. It is commonly used for policy number, membership number, student ID or invoice number.
track2		This is a string that is retrieved from the magstripe of a card by swiping the credit card through a magnetic card reader (MSR).
pan	20 / num	Card Number - no spaces or dashes. The length of the card may vary. The minimum and maximum length for the card will be returned in the 'CardLengthMinimum' and 'CardLengthMaximum' fields in the Card Data Inquiry response.
expdate	4 / num	Expiry Date - format **YYMM** no spaces or slashes.
initial_amount	9 / decimal	This is the sale amount for the loyalty card and is only used for Activation transactions. This must contain 3 digits with two penny values. Before submitting the initial_amount, the merchant must check the InitialAmountPrompt field from the Card Data Inquiry response. If the InitialAmountPrompt is "1" the minimum value passed can be 0.01 and the maximum 9999999.99. Otherwise if the InitialAmountPrompt is "0", the amount must be entered as 0.00.
total_amount	9 / decimal	This is the total amount of the transaction. This must contain 3 digits with two penny values. The minimum value passed can be 0.01 and the maximum 9999999.99. It must be equal to or greater than the "benefit_amount"
benefit	9/num	This is the number of points to be redeemed.
benefit_amount	9/decimal	This is the dollar amount on which the benefit will be calculated. The number of points associated with the transaction will be returned in the Benefit response field. This must be less than or equal to "total_amount". The benefit_amount must be submitted as a positive amount for all applicable transactions including refunds and voids.
info	9 / an	This is optional information that may be provided by the merchant. The "info" field is optional in the request, but the merchant must first verify the InfoPrompt in the Card Data Inquiry response to verify whether the prompt for this field is mandatory. InfoPrompt is "0" – merchant does not need to prompt and "info" is optional InfoPrompt is "1" – merchant is required to prompt for the "info" field, but the user may choose not to populate it (in this case the "info" field would be submitted with an empty value) This field is free form and the contents are a function of the loyalty card program. An example use of this field could be for a voucher redemption code – please refer to section 20 for further details.
language_code	1 / num	This will specify the merchant's language (i.e. terminal language) and will be used by Moneris to determine the language in which to return the response. Valid values are: 1 – English 2 – French Please refer to section 19 for further details on the 'language_code' variable.

cvd_value	6 / num	The card validation code is similar to a personal identification number (PIN). Some cards (depending on the card program) may have a CVD printed on them. This number is entered by the merchant and validated by Moneris. The CVD is a fraud prevention measure and is designed to prevent the fraudulent manufacture of a card when only the card number is known. **The CVD must never be logged/stored.** To see whether the "cvd_value" field is mandatory or optional in the request, the merchant must first verify the CVCPrompt in the Card Data Inquiry response. CVCPrompt is "0" - "cvd_value" is an optional field CVCPrompt is "1" - "cvd_value" is a mandatory field.
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 Void this must reference the Purchase, Redemption, Capture, Activation or Refund. If performing a Refund this must reference the Purchase or Capture. When performing a Capture, this must reference the Pre-Auth.
reference_number	10 / num	This field is returned in the transaction response in the 'HostReferenceNum' field – it uniquely identifies the transaction. It is used to process an Independent Refund transaction which will refer to a previously processed Purchase via eSELECTplus or a different gateway. To see whether the "reference_number" field is mandatory or optional in the request, the merchant must first verify the 'RefundAllowed' field in the Card Data Inquiry response. RefundAllowed is "1" - "reference_number" is a mandatory field. RefundAllowed is "2" - "reference_number" is an optional field.
ecr_no	8 / num	This field is also referred to as the Terminal ID (i.e. 00016927). It is an 8 digit number assigned by Moneris during the merchant account setup process; it identifies the specific terminal that processed the transaction. Primarily used for administrative transactions such as Batch Close, Open Totals and Initialization.
group_ref_num	50 / an	Merchant chosen identifier for a Group transaction. Used to identify multiple linked transactions for reporting purposes.
group_type	ernex/financial	Defines the type of transaction being referenced in the Group transaction; ERNEX or Financial (i.e. credit or debit card, etc). This field must specify either "ernex" or "financial".
payment_type	2/num	This field indicates the tender type for the financial transaction. The possible values are: 00 = Cash 01 = Cheque payment 02 = Gift payment P = Interac V = Visa M = MasterCard AX = Amex NO = Discover DC = Diners SE = Sears MS= Maestro CQ = ACH C1 = JCB (eSelectPlus Canada) C = JCB (eSelectPlus US)

15. Appendix B. Definitions of Response Fields

Response Fields		
Variable Name	**Size/Type**	**Description**
ReceiptId	50 / an	order_id specified in request
ReponseCode	3 / num	Transaction Response Code 000: Transaction approved 301-812: 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 table available in section 22.
HostReferenceNum	10 / num	This is the unique number that was assigned by the Moneris system to identify the transaction. The maximum value of this parameter is 0xFFFFFFFF (4294967295). The host can not return reference numbers greater than this value. If this field is present, it is to be included on the receipt. Please refer to section 21 for further details on receipt requirements.
TransTime	hhmmss	Processing host time stamp
TransDate	yymmdd	Processing host date stamp
TransType	2 / num	Type of transaction that was performed. Valid values are: 00 – Purchase 01 – Pre-Auth 02- Capture 04 - Refund / Independent Refund 20 - Redemption 21 - Void 23 - Activation 24 - Deactivation 25 - Card Inquiry 60 - Batch Close 65 - Open Totals 90 - Initialization
Complete	true/false	Transaction was sent to authorization host and a response was received
Message	100 / an	Response description returned from issuing institution.
TransCardCode	3 / num	This field identifies the card program and is used by Moneris and the merchant to associate device totals to host totals.
TransCardType	1 / num	This identifies the type of card. Valid values are: 0 - Loyalty 1 – Gift
TxnNumber	30 / an	Gateway Transaction identifier (ex. 2590-1189435361595-00016927_5). Required for follow-on transactions such as Void and Refund. Please note, the 3rd parameter (i.e. 00016927) represents the Terminal ID/ECR number for the device which was used to process the transaction. This is required for administrative transactions such as Batch Close.
TimedOut	true/false	Transaction failed due to a process timing out.
DisplayText	82 / an	This is a message that, if present, is to be shown to the customer on the device display. The number of lines in the text is variable (up to a maximum of 10 lines) and each line (except the last line) is terminated with a line feed. The language of the message will be determined from the “language_code” variable in the request – please refer to section 5 (Purchase) for an example.

ReceiptText	122 / an	This is a message that, if present, is to be printed on the receipt. The language of the message will be determined from the "language_code" variable in the request – please refer to section 5 (Purchase) for an example. Please refer to section 17 for the format in which this message is to be printed.
CardHolderName	40 / an	This is the name that is associated with the card in the Moneris database. If this field is present, it should be printed on the receipt. Please refer to section 21 for further details on receipt requirements.
VoucherText	255 / an	If the VoucherType field is non-zero, the text from this field should be printed in the body of the voucher. Please refer to section 17 for the format in which this message is to be printed.
VoucherType	1 / num	This field indicates the type of voucher to print. A voucher consists of a separating line on the receipt followed by text from the VoucherText field. The following values are valid: 0 - No voucher 1 - Instant win 2 – Coupon
InitialAmount	9 / num	For loyalty card activations, this is the sale amount for the card.
InitialBalance	9 / num	This is the initial balance for the card after Activation. The balance will be in points.
BatchNo	5 / num	This is the batch number, assigned by Moneris, which the transaction is a part of.
CurrentBalance	9 / num	This is the current balance for the card in points. If this field is present, it is to be printed on the receipt. If this field is not present, no balance information is to be printed on the receipt
LifetimeBalance	9/num	This is the lifetime balance for the card in points. If this field is present, it is to be printed on the receipt. If this field is not present, no balance information is to be printed on the receipt.
RemainingBalance	9/num	This is the balance on the card after deactivation. The balance will be in points.
Benefit	9 / num	This is the benefit that was generated for the transaction; the number of points for the transaction. For a Purchase or Redemption transaction, this will be a positive value. For a Refund, this will be a negative value.
ActivationCharge	9 / num	This is the amount which the merchant should charge the customer for activating the loyalty card.
Language	1 / num	This will specify the language associated with the particular transaction; merchant's (terminal) language, as well as the card's language. eSELECTplus will verify the value of the 'CheckLanguage' field and will check for the language code on the card as needed. Combining the card's language with the "language_code" provided by the merchant in the request, eSELECTplus will return one of the following codes to outline if both terminal and card language are same or if they differ. Valid values are: 3 – English (both the terminal & card languages are English) 4 – French (both the terminal & card languages are French) 5 – English terminal, French card 6 – French terminal, English card 7 – English terminal, Bilingual card 8 – French terminal, Bilingual card Please refer to section 19 for further details on language codes.
RemainingBalance	9 / num	This is the remaining balance on the card after Deactivation. The balance will be in pennies.

CardStatus	2 / an	This is the current status of the card. Valid values are: A – active C – cancelled D – deactivated H – hold (reported lost or stolen) N – new (not yet active) T – test card null – invalid response
ErrorCode ErrorMessage	1 / num 30 /an	The ErrorCode and ErrorMessage fields are used to return transaction processing errors to the merchant. The eSELECTplus gateway will perform many of the required verifications on behalf of the merchant (i.e. CVCPrompt, CheckMod10, RefundAllowed, etc) and these fields will be used to return errors when any of the verifications fail. Valid values are:

ErrorCode	ErrorMessage
1	Card length error.
2	CVC missing error.
3	CVC length error.
4	Refund not allowed error.
5	Mod 10 error.
6	Bin look up error.
7	No pan error.

HostTotals	If the HostTotals is not “null” then this indicates that either a Batch Close or Open Totals administrative transaction was performed and that the following fields will be present.	
term_id	8 / num	Terminal Identifier – Determined which ERNEX terminal ID that Host Totals are referring to and/or which terminal has had its batch closed.
closed	true/false	Batch has been successfully closed on the processing host. The batch number should now be incremented.
CardCode	3 / num	This field identifies the card program and is used by Moneris and the merchant to associate device totals to host totals.
PurchaseCount	4 / num	This is the number of purchase transactions. (i.e. 0023 = 23 purchases)
PurchaseTotal	9 / num	This is the total of purchase transactions. The total amount is represented in pennies (i.e. 000000100 = $1.00)
PurchaseBenefitTotal	9/num	This is the total number of points awarded.
RefundCount	4 / num	This is the number of refund transactions.
RefundTotal	9 / num	This is the total amount of refund transactions represented in pennies.
RefundBenefitTotal	9/num	This is the total points refunded.
ActivationCount	4 / num	This number of activation transactions. Count = Total number of Activations - the number of voided Activations
ActivationTotal	9 / num	This is the total of activation transactions represented in pennies. Total Amount = Total amount of Activations - the amount of voided Activations
CorrectionCount	4 / num	This is the number of correction and void transactions. Total number = Voided Purchases + Voided Activations + Voided Refunds + Voided Redemptions.
CorrectionTotal	9 / num	Reserved for future use. Default = 000000000.
RedemptionCount	4 / num	This is the number of redemption transactions. (i.e. 0023 = 23 redemptions)
RedepemptionTotal	9 / num	This is the total of redemption transactions. The total amount is represented in

		pennies (i.e. 000000100 = $1.00)
RedemptionBenefitTotal	9/num	This is the total number of points redeemed.

InitData	If InitData is not “null” then this indicates that some or all of the following fields will be present. This data will include information particular to each card program (CardCode).	
CardCode	3 / num	This field identifies the card program and is used by Moneris and the merchant to associate device totals to host totals.
CardType	1 / num	This identifies the type of card. Valid values are: 0 - Loyalty 1 – Gift
CheckMod10	1 / num	This specifies whether the merchant should subject the card to a modulus 10 check. This check will automatically be performed by the eSELECTplus gateway, but it also an option as an additional verification for the merchant. Valid values are: 0 - Do not check mod 10 1 - Perform mod 10 check
CheckLanguage	1 / num	This check will automatically be performed by the eSELECTplus gateway and the result will be returned to the merchant in the ‘Language’ field, but it also an option as an additional verification for the merchant. This field specifies whether the card has a language code encoded on the magstripe. If the card does not have a language code encoded, the merchant should use their language for the cardholder language. Valid values are: 0 - No language code 1 - Language code encoded as last digit before stop sentinel. 2 - Language code encoded as last digit before stop sentinel. If valid language not read, treat card as bilingual. Please refer to section 18 for further track 2 layout details. Please refer to section 19 for further details on language codes.
CVCPrompt	1 / num	This specifies whether the merchant should prompt for a CVC (card validation code). Valid values are: 0 - No prompt 1 - Prompt
InfoPrompt	1 / num	This specifies whether the merchant should prompt for a value for the “info” variable in the request – please refer to section 5 (Purchase) for an example. The text for the prompt should be taken from the InfoPromptText Field. Valid values are: 0 - No prompt 1 - Prompt
AmountPrompt	1/num	This specifies whether the merchant should prompt for a value for the “benefit_amount” field when processing a Loyalty card. The text for the amount should be taken from the AmountPromptText field. Valid values are: 0 - No prompt (set “benefit_amount” = “total_amount”) 1 - Prompt
BenefitPrompt	1/num	This specifies whether the merchant should prompt for a value for the “benefit” field when processing a Loyalty card. The text for the amount should be taken from the BenefitPromptText field. Valid values are: 0 - No prompt 1 - Prompt

InitialAmountPrompt	1 / num	This specifies whether the merchant should prompt for an "initial_amount" when activating a card - please refer to section 6 (Activation) for an example. Valid values are: 0 - No prompt (default amount of 0.00 is to be sent in the Activation request) 1 - Prompt The amount entered is sent in the "initial_amount" field of the transaction request.
RedemptionAmount	9/num	This is the dollar amount of the points redeemed. This field is only used for a redemption that was completed by specifying a dollar amount in the request (RedemptionType =1). If the request specified the number of points, this field should be ignored in the response.
RedemptionType	1/num	This specifies how the merchant should process a redemption transaction. Valid values are: 0 – Prompt for benefit total (point value redemption) 1 – Prompt for dollar amount (dollar value redemption) When prompting for a dollar amount, the amount should be sent in the "benefit_amount" and "total_amount" fields. When prompting for benefit total, the value should be sent in the "benefit" field.
RefundAllowed	1 / num	Flag indicating whether or not Refunds are allowed for the CardCode. This check will automatically be performed by the eSELECTplus gateway, but it also an option as an additional verification for the merchant. Valid values are: 0 - No 1 – Yes with original "reference_number" required 2 – Yes with original "reference_number" optional The "reference_number" pertains to the Independent Refund transaction.
CardLengthMinimum	2 / num	This is the minimum length of a card number. The minimum and maximum values may be the same if variable length card numbers are not supported.
CardLengthMaximum	2 / num	This is the maximum length of a card number. The minimum and maximum values may be the same if variable length card numbers are not supported.
LowBIN1	10 / an	This check will automatically be performed by the eSELECTplus gateway, but it also an option as an additional verification for the merchant. This is the low end of the BIN range for the card program. BIN ranges are always specified in pairs (a low and a high). When validating a card number it must fall between the low and the high. The number of characters in a BIN may be less than the number of characters in the card. This field will be right filled with spaces. There can be up to four BIN ranges for each card program.
HighBIN1	10 / an	This check will automatically be performed by the eSELECTplus gateway, but it also an option as an additional verification for the merchant. This is the high end of the BIN range for the card program. BIN ranges are always specified in pairs (a low and a high). When validating a card number it must fall between the low and the high. The number of characters in a BIN may be less than the number of characters in the card. This field will be right filled with spaces. There can be up to four BIN ranges for each card program.
LowBIN2	10 / an	Please refer to LowBIN 1 for description.
HighBIN2	10 / an	Please refer to HighBIN 1 for description.
LowBIN3	10 / an	Please refer to LowBIN 1 for description.
HighBIN3	10 / an	Please refer to HighBIN 1 for description.
LowBIN4	10 / an	Please refer to LowBIN 1 for description.
HighBIN4	10 / an	Please refer to HighBIN 1 for description.

RecordType	2 / an	This field is used in the Initialization and Card Data Inquiry responses and identifies the type of configuration data that follows. This field should be used by the merchant to know how to interpret the information. Valid values are: 01 - Card configuration block 1 02 - Card configuration block 2 03 - Card configuration block 3
CardDescription	20 / an	This is a description of the card that should be printed on receipts and reports.
InfoPromptText	20 / an	This is the text of the message to display when prompting the user to enter a value for the “info” variable in the request - please refer to section 5 (Purchase) for an example. For example, “Bonus Code”.
BenefitDescription	20/an	This is a description of the unit of benefit which should be used for prompts, receipts and reports.
AmountPromptText	20/an	This is the text of the message to display when prompting the user to enter a value for the “benefit_amount” field in the request. Please refer to section 5 (Purchase) for an example.
BenefitPromptText	20/an	This is the text of the message to display when prompting the user to enter a value for the “benefit” field in the request. Please refer to section 5 (Redemption) for an example.
TotalsDescription	20/an	This is a description of the LifetimeBalance field.

16. Appendix C. 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 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

17. Appendix D. Formatting Codes

Each text field has a maximum length. Within the maximum length, the text message can be organized into many lines (up to a maximum of 10). Each line is terminated with a line feed character (0x0A). The last line of the message may not include a line feed.

The first character or each line may be a formatting code. The merchant should then act on the formatting code and print all text up to the next line feed (or end of message) according to the specified formatting code. The following formatting codes are defined.

0x07 - Normal height, left justified
0x08 - Normal height, centred
0x0F - Normal height, right justified
0x0B - Double height, left justified
0x0C - Double height, centred
0x0E - Double height, right justified

If a formatting code is not included in the message text, or if the merchant is unable to print using the specified format, then the merchant may choose their own default format.

18. Appendix E. Track 2 Layout

The Moneris Loyalty card has a unique format in the discretionary data field of track 2 on the magstripe. This information is used by the host during an Activation transaction and is designed to reduce the risk of fraud.

The track 2 data for a Moneris Loyalty card will have one of the following formats:

No Language Code

Field	Value	Length	Type
Start Sentinel	;	1	Fixed
Card Number	Numeric	19	Variable
Field Separator	=	1	Fixed
Expiry Date	YYMM	4	Fixed
Service Code	000	3	Fixed
Unused		5	Fixed
End Sentinel	?	1	Fixed

Including Language Code

Field	Value	Length	Type
Start Sentinel	;	1	Fixed
Card Number	Numeric	19	Variable
Field Separator	=	1	Fixed
Expiry Date	YYMM	4	Fixed
Service Code	000	3	Fixed
Unused		5	Fixed
Language Code	1=English, 2=French, 3=Bilingual	1	Fixed
End Sentinel	?	1	Fixed

Including Language Code and End Zero

Field	Value	Length	Type
Start Sentinel	;	1	Fixed
Card Number	Numeric	19	Variable
Field Separator	=	1	Fixed
Expiry Date	YYMM	4	Fixed
Service Code	000	3	Fixed
Unused		5	Fixed
Language Code	1=English, 2=French, 3=Bilingual	1	Fixed
End Zero	0	1	Fixed
End Sentinel	?	1	Fixed

19. Appendix F. Language Codes

For many of the transactions the “language_code” must be submitted in the request. This “language_code” identifies the merchant’s language of choice (i.e. terminal language).

Valid “language_code” values are:
1 – English
2 – French

When a transaction is submitted, eSELECTplus determines whether the card has a language code encoded on the magstripe (track 2) by examining the configuration data received in the Card Data Inquiry transaction. The ‘CheckLanguage’ field indicates whether the card has a language code encoded.

Valid values for ‘CheckLanguage’ are:
0 - No language code
1 - Check for language code
2 - Check for language code and if not found/valid, assume card is bilingual

Most cards do not have a language code encoded on the magstripe. But, if the card does happen to have a language code encoded, eSELECTplus then attempts to retrieve the cardholder’s language from the card’s magstripe. This can only be done if the card is swiped and a track 2 is provided in the transaction request. Please refer to section 18 for further details on the layout of a card’s magstripe (track 2).

Valid magstripe language code indicators are:
1 – English
2 – French
3 – Bilingual

Lastly, eSELECTplus combines what the merchant provides in the transaction request (“language_code”) with the language indicator from the magstripe, to return a ‘Language’ indicator in the response.

Valid response ‘Language’ values are:
3 – English (both the terminal & card languages are English, or no card language was used)
4 – French (both the terminal & card languages are French, or no card language was used)
5 – English terminal, French card
6 – French terminal, English card
7 – English terminal, Bilingual card
8 – French terminal, Bilingual card

The value returned in the response ‘Language’ field is to be used by the merchant to determine the language of the receipt. In some situations it is desirable for the merchant to produce a bilingual receipt. This ensures that the cardholder will be able to read the receipt.

20. Appendix G. Vouchers - Coupon and Instant Win

If the card program is set up to include vouchers, either Coupons or Instant Win, then the merchant must be able to receive these messages and properly handle them.

Printing a Voucher

In the initial transaction response the merchant must verify the 'VoucherType' field to determine whether the transaction includes a voucher. For example, a Purchase transaction may include a voucher in the response.

Valid response 'VoucherType' values are:
0 – No voucher
1 – Instant win
2 – Coupon

If the 'VoucherType' field returns a 1 or 2, the merchant must then retrieve the text from the 'VoucherText' field within the response. This text is to be printed on the receipt presented to the customer. Please note, this field may include formatting codes that are to be used when printing this text on a receipt. Please refer to section 17 for further details on possible formatting codes.

Redeeming a Voucher

Once a customer chooses to redeem their voucher, either a Coupon or an Instant Win promotion, the merchant must then process yet another Purchase transaction (voucher redemption purchase[1]) with certain fields being submitted.

To redeem a voucher the Purchase request fields must be populated as follows:

store_id & ***api_token***: must reference the same store that processed the initial purchase transaction.
order_id: any unique order ID.
total_amount: submit as 0.00. If the customer wishes to pay for other goods and service while redeeming the voucher this may then be any other amount[2].
track2: optional if ***pan*** and ***expdate*** are submitted; otherwise it must be the same card that was used to process the initial purchase.
pan & ***expdate***: optional if ***track2*** is submitted; otherwise it must be the same card that was used to process the initial purchase.
cvd_value: proper CVD value for the Loyalty card submitted in ***track2*** and/or ***pan***.
info: must include the 'HostReferenceNum' received in the initial purchase's response (ex. 385593). This number will identify the initial purchase and the voucher associated with it.
language_code: same as in the initial purchase – please refer to section 19 for further details.

When redeeming a voucher, certain error messages may be returned in the voucher redemption purchase response.

ResponseCode	*Message*
338	Benefit has expired Benefit already redeemed Benefit has been cancelled Benefit not transferable Can not reverse a benefit redemption

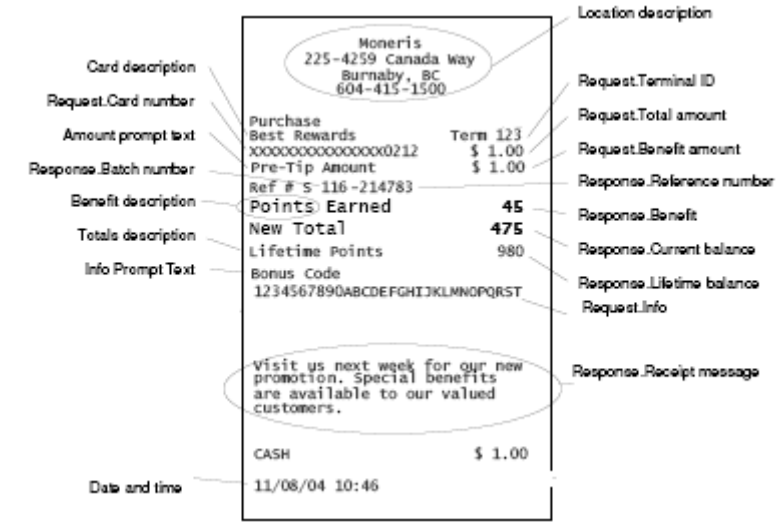
For a list of all other ResponseCode's, please refer to the table available in section 22.

[1] For our example, we will refer to the two Purchase transactions as Initial Purchase and Voucher Redemption Purchase.
[2] Please note, if the voucher redemption portion fails, then the purchase transaction will also be declined and must be reprocessed. Also, if the purchase declines, then the coupon will remain active until another voucher redemption purchase is submitted.

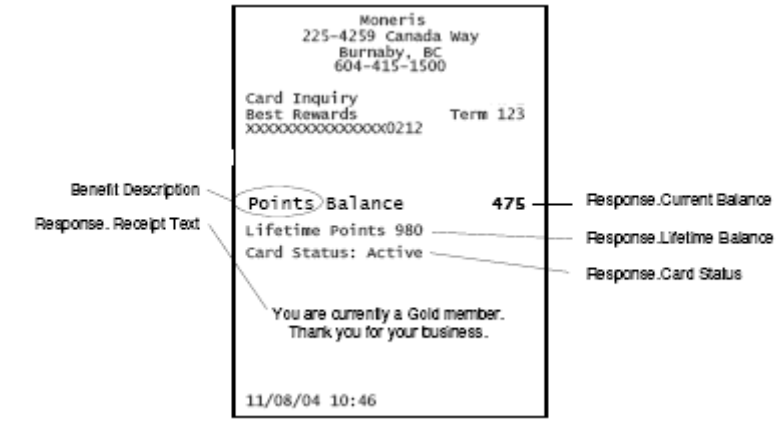
21. Appendix H. Sample Receipts

When printing a receipt for a transaction, there are a number of fields that must be included. Below are a number of sample receipts that display the required fields. For a full list and description of the mandatory fields, please refer to section 7.

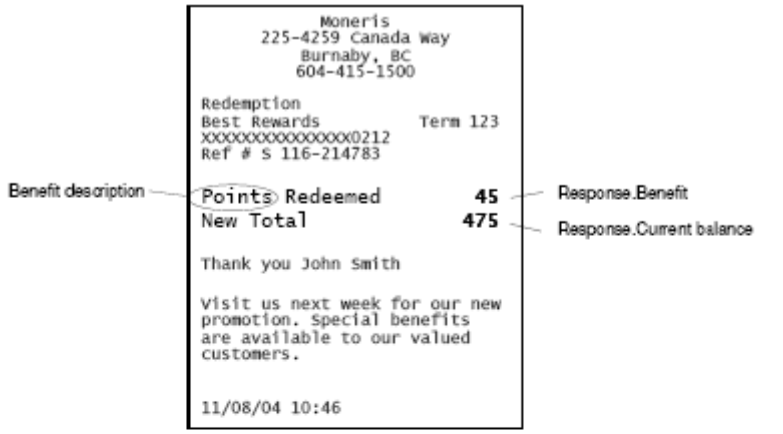
Sample Purchase Receipts

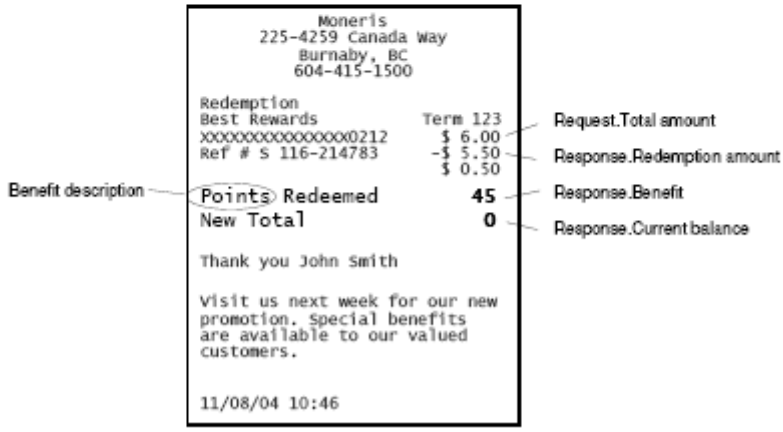


Sample Card Inquiry Receipt



Sample Redemption Receipts





22. Appendix I. Response Codes

When processing a transaction, to verify whether or not it has been successful the merchant should verify the result in the ResponseCode variable. The table below outlines the possible response codes and their associated message.

Code	Status	Message
0	Approved	Approved
0	Approved	Approuvee
301	Declined	Terminal ID not on file
301	Declined	Numero de terminal inconnu
302	Declined	Card Number Not On File
302	Declined	Numero de carte inconnu
303	Declined	Invalid Product ID
303	Declined	Produit Inconnu
304	Declined	No Rate Table For Merchant
304	Declined	Pas de Table de Concordance
305	Declined	Invalid Expiry Date
305	Declined	Date d'Expiration Non Valable
306	Declined	Card Number Already Exists
306	Declined	Numero de carte deja cree
308	Declined	Invalid Transmission Number
308	Declined	Numerode Transmission Non Valable
310	Declined	No Redemption Criteria
310	Declined	Pas de Critere de Rachat De Points
311	Declined	Insufficient Points For Redemption
311	Declined	Nb de Points Pour Rachat Insuffisant
316	Declined	Refund Amount Greater Than Original
316	Declined	Montant du Remb. Superieur a l'Achat
317	Declined	Unknown Card Program
317	Declined	Programme Inconnu
318	Declined	Transaction Code Invalid
318	Declined	Code De Transaction Non Valable
319	Declined	Merchant ID Invalid
319	Declined	Numero de Marchand Non Valable
320	Declined	Terminal ID Disabled
320	Declined	Terminal Desactivee
321	Declined	Merchant ID Disabled
321	Declined	Marchand Desactivee
322	Declined	Card Disabled
322	Declined	Carte Desactivee
323	Declined	Non-Loyalty Program
323	Declined	Pas Un Programme de Fidelisation
324	Declined	Invoice Number Invalid
324	Declined	Numero de Facture Non Valable
325	Declined	Invalid CVC
325	Declined	CVC Non Valable
327	Declined	Card Balance Is Zero
327	Declined	Solde de Carte Nul
328	Declined	Invalid Initial Balance
328	Declined	Solde Initial Non Valable
329	Declined	Card Already Active
329	Declined	Carte Deja Activee
330	Declined	Card Already Deactivated
330	Declined	Carte Deja Desactivee

331	Declined	Card Not Active
331	Declined	Carte Jamais Activee
332	Declined	Insufficient Points For Refund
332	Declined	Solde Insuffisant Pour Rembours
333	Declined	Matching Transaction Not Found
333	Declined	Transaction Originale Introuvable
334	Declined	Host System Error
334	Declined	Erreur Systeme Ordin. Central
335	Declined	Transaction Already Voided
335	Declined	Transaction Deja Annulee
336	Declined	Message Format Invalid
336	Declined	Format de Message Non Valable
337	Declined	System Maintenance Please Try Later
337	Declined	Maintenance en Cours Essayez Plus Tard
338 [3]	Declined	**Moneris host message. Host will return message text in the Display Message Field, in the language of the location**
340	Declined	Internal Moneris Timeout
340	Declined	Temps de reponse Moneris trop long
341	Declined	Card Number Not On File
341	Declined	Numero de carte inconnu
342	Declined	No Match For Device Name
342	Declined	Type d'appellation non valable
343	Declined	Invalid Amount
343	Declined	Montant non valable
345	Declined	Invalid Host Config
345	Declined	Config ordin central non valable
346	Declined	Device Name Already In Use
346	Declined	Type d'appellation deja utilisee
347	Declined	Invalid Input Values
347	Declined	Valeur des entrees invalides
348	Declined	Record Not Updated
348	Declined	Engeristrement non mis à jour
349	Declined	Invalid Email Address
349	Declined	Adresse électronique invalide
350	Declined	Records Not Found
350	Declined	Enregistrements non trouvés
483	Declined	Network Timeout
483	Declined	Arret temporaire du reseau
809	Declined	Invalid Transaction Code
809	Declined	Code de transaction invalide
811	Declined	System Error Please Try Later
811	Declined	Erreur du systeme, s'il-vous-plait, re-essayer plus tard
812	Declined	System Error Please Try Later
812	Declined	Erreur du systeme, s'il-vous-plait, re-essayer plus tard

[3] Please refer to section 20 for a list of some possible messages associated with this ResponseCode.